CBMPy User Guide

Release 0.7.4

Brett G. Olivier

April 13, 2017

Contents:

CHAPTER

ONE

CBMPY: INSTALLATION GUIDE

1.1 Introduction

1.1.1 Support

PySCeS CBMPy is Open Source software released under the GNU GPL 3 licence (included with the source code) and is in constant development. All the latest downloads, documentation and development information is available on **SourceForge**: http://cbmpy.sourceforge.net

CBMPy is developed in the Department of Systems Bioinformatiocs at the Vrije Universiteit Amsterdam, as part of the BeBasic Metatoolkit project, by Brett Olivier (bgoli@users.sourceforge.net)

1.1.2 Python standard library modules

CBMPy is developed and tested on Python 2.7.x. but should also work on Python 3. The following Python Standard Library modules are used in CBMPy and should be available as part of any CPython distribution and not require additional installation:

```
'cPickle', 'cStringIO', 'cgi', 'copy', 'gc', 'itertools', 'locale', 'math',
'multiprocessing', 'os', 'pprint', 'random', 're', 'shutil', 'subprocess',
'time', 'urllib2', 'webbrowser', 'xml'
```

1.1.3 Required libraries (Python bindings)

Besides those mentioned above, the following packages are required for CBMPy's core functionality. Note that it is possible to install CBMPy using only *numpy* but that only very limited subset of functionality is then available. CBMPy is primarily developed on Microsoft Windows and Ubuntu Linux and where possible the package name is provided such that can be used with the software center or package manager `sudo apt-get install <package>` (please see the man pages for sudo and apt-get if you don't know what this). A comprehensive list of modules are listed at the end of this document. In the case of external C/C++ libraries the Python bindings should be installed (e.g. libSBML). Many of these are available in *batteries included* Python distributions and via the pip installer.

1.2 Overview

CBMPy has been designed to be used as a framework and can be used in different contexts. Here are the minimal requirements for each role. The fastest way to install the latest official version of CBMPy is

using PyPI: https://pypi.python.org/pypi/cbmpy while development source, binaries, tools and utilities are available from SourceForge: https://sourceforge.net/projects/cbmpy/files

1.2.1 PyPI

Try one of:

```
pip install cbmpy
easy_install cbmpy
```

1.2.2 Minimal

Read, write and convert model files as well as view, create and edit model components and annotation.

- *numpy*
- *libSBML* (Python bindings)
- *biopython*

1.2.3 Normal

In addition to the minimal requirements a linear optimizer is required

- Optimization libraries (one or more of):
- CPLEX (LP, MILP): http:://www.ibm.com
- GLPK (LP): http://tfinley.net/software/pyglpk/
- *matplotlib*
- *sympy*
- *xlwt*
- *xlrd*

1.2.4 Full

This includes graphical interface support

- *pyqt4*
- *wxPython*
- *suds*
- *scipy*
- *h5py*
- *networkx*

1.2.5 User tools

These are highly recommended user tools but are not required for using CBMPy.

- iPython
- iPython-notebook
- SCiTE

1.3 Installing on Ubuntu Linux

1.3.1 Python 2.7 (full install)

First we create a scientific Python workbench using the Ubuntu package manager:

```
sudo apt-get install python-dev python-numpy python-scipy python-
↪matplotlib  python-pip
sudo apt-get install python-sympy python-suds python-xlrd python-xlwt␣
↪python-h5py
sudo apt-get install python-biopython python-wxgtk2.8 python-qt4
sudo apt-get install ipython ipython-notebook
```

1.3.2 libSBML

Installing libSBML is now easy using PiP and PyPI, first we need some dependencies:

```
sudo apt-get install libxml2 libxml2-dev
sudo apt-get install zlib1g zlib1g-dev
sudo apt-get install bzip2 libbz2-dev
```

Then we can install the latest and greatest with:

```
sudo pip install python-libsbml
```

1.3.3 glpk/python-glpk

Currently, CBMPy still requires GLPK version 4.47 to work with glpk-0.3 so it is important *not* to install the latest using the ubuntu package manager but rather use these instructions. First download the GLPK source and bindings:

```
https://sourceforge.net/projects/cbmpy/files/tools/glpk/glpk-4.47.tar.gz
https://sourceforge.net/projects/cbmpy/files/tools/glpk/glpk-0.3.tar.bz2
```

and then install the the following dependency:

```
sudo apt-get install libgmp-dev
```

Unpack the glpk-4.47 source and make it your current directory:

```
cd GLPK source (e.g. glpk-4.47)
```

Build and install GLPK:

```
./configure --with-gmp
make
make check
sudo make install
sudo ldconfig
```

Change to the python-glpk source (glpk-0.3) and make it the current directory:

```
cd to python-glpk source (glpk-0.3):
```

Then build and install it with the following commands:

```
make
sudo make install
```

Finally, install CBMPy either using PyPI:

```
pip install cbmpy
     or
easy_install cbmpy
```

Or download a source file from either:

```
https://pypi.python.org/pypi/cbmpy
https://sourceforge.net/projects/cbmpy/files
```

and install manually:

```
python setup.py build sdist
sudo python setup.py install
```

1.4 Installing on Microsoft Windows

For the modeller that does not want to customize his installation and install all of the individual packages by hand there are some *batteries included* Python distributions which have many (if not all) of the required packages required.

CBMPy is developed using a 64 bit version of *Anaconda Python 2.7* (http://continuum.io) has also been installed on 32 bit *Python(x,y)* and Enthought Python Distribution (EPD) http://www.enthought.com

1.4.1 Anaconda

Most of the required packages can be installed using the *conda* package manager for example:

```
conda install sympy pyqt4
```

or using pip (libSBML) or setuptools

1.4.2 Python(x,y)

Select the following packages from the *Python* branch of the Python(x,y) installation directory:

- WxPython
- Sympy
- NetworkX
- xlrd
- xlwt
- h5py
- wxPython
- PyQT4

1.4.3 Installing CBMPy

There are two ways to install CBMPy either download the latest release as source bundle or binary from http://cbmpy.sourceforge.net and unzip or execute from a a temporary directory (recommended). Or, if you want the latest (greatest and potentially broken) version grab the latest revision from the the CBMPy Subversion repository:

```
svn co http://sourceforge.net/p/cbmpy/code/HEAD/tree/trunk/cbmpy cbmpy
```

In both cases you should should now have a directory that contains a file *setup.py* which can install by simply typing the following into a Windows shell (command line):

```
python setup.py build
python setup.py install
```

1.4.4 Installing libSBML with Python bindings

It is highly recommended to install libSBML which CBMPy uses to provide support for the Systems Biology Markup Language (SBML). First go to the libSBML download page http://sbml.org/Software/libSBML page follow the *Download libSBML –> Stable –> Windows –> 32bit* path and download libSBML (e.g. libSBML-5.10.0-win-x86.exe). The latest stable version can be found at http://sbml.org/Software/libSBML

> http://sourceforge.net/projects/sbml/files/libsbml/5.10.0/stable/Windows/32-bit/libSBML-5.10.0-win-x86.exe/download

Run the installer and make sure you select the Python Bindings during installation or install the appropriate Python bindings that match your Python(x,y) version directly e.g. (libSBML-5.10.0-win-py2.7-x86.exe)

1.4.5 Optmization (1): IBM cplex optimization studio (Academic)

If you have access to the the IBM CPLEX solver. It is a a good idea to use the latest available version. Again choose the appropriate 32 or 64 bit version and an installation path that suites your setup.

- Run **cplex_studio126.win-x86-32.exe**
- Select English language and accept licence
- Set "Program" install directory to C:\ILOG\CPLEX_Studio126
- Allow default associations to be set and PATH update

Once installation is complete we need to install the Python bindings

- Open a terminal
- Execute `cd c:\\ILOG\\CPLEX_Studio126\\cplex\\python\\x86_win32`
- Execute `python setup.py install`

1.4.6 Optmization (2): GLPK

CBMPy 0.7.4 includes support for the free, Open Source GLPK solver. This allows access to CBMPy's LP functionality (MILP's requires CPLEX). A port of PyGLPK 0.3 is maintained by the OpenCOBRA project which is mirrored here:

https://sourceforge.net/projects/cbmpy/files/tools/glpk/

Select the binary or source distribution you require and either execute the binary:

- Execute `glpk-0.3.win32-py2.7.exe`

1.4.7 Testing your new installation

If everything has gone according to plan you can test your installation:

- Open a terminal
- Execute `ipython`
- In ipython shell, execute `import numpy,h5py,xlrd,xlwt`

No import errors should occur.

- Execute `import libsbml`
- Execute `libsbml.LIBSBML_VERSION_STRING`

A successful test should return (for example):

```
In : libsbml.LIBSBML_VERSION_STRING
Out: '51000'
```

- Execute `import cbmpy as cbm`

This should return:

```
In [1]: import cbmpy as cbm

*********************************************************************
* Welcome to CBMPy (0.7.4) - PySCeS Constraint Based Modelling      *
*                http://cbmpy.sourceforge.net                       *
* Copyright(C) Brett G. Olivier 2010 - 2015                         *
* Dept. of Systems Bioinformatics                                   *
```

```
* Vrije Universiteit Amsterdam, Amsterdam, The Netherlands         *
* CBMPy is distributed under the GNU GPL v 3.0 licence, see        *
* LICENCE (supplied with this release) for details                 *
*******************************************************************
```

Exit ipython with CTRL-D

If you installed CPLEX then try:

- Open a terminal
- Execute `ipython`
- Execute `import cplex`
- Execute `lp = cplex.Cplex()`
- Execute `lp.solve()`

A succesful test should return:

```
In : lp.solve()
Tried aggregator 1 time.
No LP presolve or aggregator reductions.
Presolve time =    0.00 sec.
```

Exit ipython with CTRL-D

If you installed GLPK then try:

- Open a terminal
- Execute `ipython`
- Execute `import glpk`
- Execute `lp = glpk.LPX()`

A succesful test should return:

```
In : glpk.LPX()
<glpk.LPX 0-by-0 at 0x036C24C8>
```

Exit ipython with CTRL-D

1.4.8 Install CBMPy (http://cbmpy.sourceforge.net)

Download the latest version of CBMPy

- Run **cbmpy-0.7.x.win32.exe** (or newer for 32 bit Windows)
- Run **cbmpy-0.7.x.amd64.exe** (or newer for 64 bit Windows)

Test installation:

- Open a terminal
- Execute `ipython`
- Execute `import cbmpy as cbm`

This should return:

```
In [1]: import cbmpy as cbm

*****
Using GLPK
*****

WX GUI tools available.
Qt4 GUI tools available

CBMPy environment
******************
Revision: r346


*********************************************************************
* Welcome to CBMPy (0.7.4) - PySCeS Constraint Based Modelling      *
*                http://cbmpy.sourceforge.net                       *
* Copyright(C) Brett G. Olivier 2010 - 2015                         *
* Dept. of Systems Bioinformatics                                   *
* Vrije Universiteit Amsterdam, Amsterdam, The Netherlands          *
* CBMPy is distributed under the GNU GPL v 3.0 licence, see         *
* LICENCE (supplied with this release) for details                  *
*********************************************************************
```

Exit ipython with CTRL-D

1.5 Linux: Ubuntu

On Linux many of the base dependencies are available as packages or from the Python Cheeseshop (http://pypi.python.org/pypi). For **libSBML**, **CPLEX** and/or **GLPK** please see the *Generic installation on Microsoft Windows (XP, 7, 2008)* for more details. For example using **Ubuntu** the base dependencies can be easily installed (depending on what functionality is required). If you don't know what these packages are please look them up before installing.

Required:

```
sudo apt-get install python-dev python-numpy

- libSBML for SBML support.
```

Please see http://sbml.org/Software/libSBML or try the following. Depending on your configurationyou need to install libxml2, bzip2 and their associated "dev" packages:

```
apt-get install libxml2 libxml2-dev
apt-get install zlib1g zlib1g-dev
apt-get install bzip2 libbz2-dev

easy_install pip

# for standard libSBML
pip install python-libsbml
```

```
# for "experimental" libSBML (for FBC V2 and Groups support)
pip install python-libsbml-experimental
```

- Optimization (at least one of):
  - IBM CPLEX: http://www.ibm.com
  - PyGLPK: https://sourceforge.net/projects/cbmpy/files/tools/glpk/

Please note that due to changes in the GLPK API the current version of PyGLPK (0.3) **only supports GLPK up until version 4.47**. If your system has a newer version of GLPK then the current workaround is to uninstall the newer version and compile 4.47 from source (also available from the above directory). Dependencies are standard Linux build tools and GMP etc:

```
tar xzf glpk-4.47.tar.gz
cd glpk-4.47
./configure --with-gmp
make
make check
sudo make install
```

Graphical interfaces (highly recommended):

```
sudo apt-get install python-wxgtk2.8 python-qt4 python-matplotlib
```

Extended IO (highly recommended):

```
sudo apt-get install python-xlrd python-xlwt python-sympy
```

Web services and database:

```
sudo apt-get install python-suds python-pysqlite2
```

Advanced functionality:

```
sudo apt-get install python-scipy python-h5py python-networkx
```

User tools (highly recommended):

```
sudo apt-get install ipython ipython-notebook scite
```

1.6 Linux: Ubuntu 14.04

1.6.1 Python2

First we create a scientific Python workbench:

```
sudo apt-get install python-dev python-numpy python-scipy
sudo apt-get install python-matplotlib  python-pip
sudo apt-get install python-sympy python-suds python-xlrd
sudo apt-get install python-xlwt python-h5py
sudo apt-get install python-wxgtk2.8 python-qt4
sudo apt-get install ipython ipython-notebook
```

1.6.2 libSBML

Installing libSBML is now easy using Pip:

```
sudo apt-get install libxml2 libxml2-dev
sudo apt-get install zlib1g zlib1g-dev
sudo apt-get install bzip2 libbz2-dev

sudo pip install python-libsbml
```

1.6.3 glpk/python-glpk

GLPK needs to be version 4.47 to work with glpk-0.3:

```
sudo apt-get install libgmp-dev
```

cd GLPK source (e.g. glpk-4.47):

```
./configure --with-gmp
make
make check
sudo make install
sudo ldconfig
```

cd to python-glpk source (glpk-0.3):

```
make
sudo make install
```

1.6.4 CBMPy

Finally, install CBMPy:

```
python setup.py build sdist
sudo python setup.py install
```

1.6.5 Installing PyscesMarinerCBM

This will install PySCeS Mariner that adds SOAP web-services capability to CBMPy. First unpack pyscesmariner-0.7.7.zip and install the cherrypy webserver:

```
sudo apt-get install python-cherrypy
```

1.6.6 Install soaplib

cd <pysces_cbm_mariner>/misc:

```
tar -xf soaplib-0.8.1.tar.gz
cd soaplib-0.8.1
python setup.py build sdist
sudo python setup.py install
```

1.6.7 Install Mariner

cd <pysces_cbm_mariner> and set mariner configuration (not needed for Ubuntu, Windows or if the server does not read SBML):

```
sudo nano /usr/local/lib/python2.7/dist-packages/pyscesmariner/
 ↪MarinerConfig.py
PATH_LIBSBMLTHREAD = '/usr/local/lib/python2.7/dist-packages/pyscesmariner/
 ↪libSBMLthread.pyc'
PATH_LIBSBML_CONVERTTHREAD = '/usr/local/lib/python2.7/dist-packages/
 ↪pyscesmariner/libSBMLConvertThread.py'
```

cd to <pysces_cbm_mariner>:

```
python setup.py build sdist
sudo python setup.py install
```

1.6.8 Test installation

Open a new terminal window:

```
# cd <pysces_cbm_mariner>/demo
python cbm_server_demo.py
```

Open another terminal and run the client demo:

```
python cbm_client_demo.py
```

Kill the server by closing the terminal window.

1.6.9 Python3

Not all dependencies are available for Python3:

```
sudo apt-get install python3-dev python3-numpy python3-scipy
sudo apt-get install python3-matplotlib  python3-pip
sudo apt-get install python3-xlrd python3-h5py

# need to find out what is going on with Python3 and xlwt suds
# easy_install3 sympy ???
# wxPython and PyQt4 not in Ubuntu P3 builds yet

sudo apt-get install ipython3 ipython3-notebook

sudo apt-get install libxml2 libxml2-dev
sudo apt-get install zlib1g zlib1g-dev
sudo apt-get install bzip2 libbz2-dev

sudo pip3 install python-libsbml

sudo apt-get install python-qt4 python-qt4-dev python-sip
sudo apt-get install python-sip-dev build-essential
```

1.7 Apple Macintosh: OS X

Installation is similar to Linux except packages are installed using distutils and pip. The first step is to install the Mac development tools `xcode`

Install `Python` packages:

```
sudo easy_install numpy ipython scipy matplotlib
sudo easy_install xlrd xlwt sympy suds pyparsing pip
```

Use pip to install advanced Ipython and libsbml:

```
sudo pip install ipython[notebook]
ARCHFLAGS=-Wno-error=unused-command-line-argument-hard-error-in-future ␣
↪pip install python-libsbml
```

For `solvers`, either install your own copy of CPLEX or build PyGLPK which requires building both the GMP and GLPK libraries.

`GMP` (https://gmplib.org/):

```
download gmp
./configure --prefix=/usr/local
make
make check
sudo make install
```

`GLPK` (http://sourceforge.net/projects/cbmpy/files/tools/glpk):

```
download glpk-4.47.tar.gz
./configure --prefix=/usr/local --with-gmp
make
sudo make install
```

`PyGLPK` (http://sourceforge.net/projects/cbmpy/files/tools/glpk):

```
download python-glpk-0.3
python setup.py build
sudo python setup.py install
```

1.8 Installing PySCeS-CBM Mariner (Microsoft Windows and Linux)

The PySCeS Mariner module exposes the CBMPy functionality as SOAP web services (e.g. as a back-end to FAME (http://F-A-M-E.org)). It is available for download from SourceForge:

- PySCeS-CBM Mariner: http://sourceforge.net/projects/cbmpy/files/release/pysces_mariner/

1.8.1 Dependencies: CherryPy, libXML and SOAPlib

PySCeS-CBM Mariner requires (pure python) soaplib 0.8.1 (supplied with it) or downloadable from:

```
https//sourceforge.net/projects/cbmpy/files/tools/soaplib/
```

Soaplib itself has two dependencies which should be installed first:

- LXML (http://lxml.de)
  - Windows: install with `easy_install lxml`
  - Linux (Ubuntu) use `sudo apt-get install python-lxml`
- CherryPy (http://www.cherrypy.org)
  - Windows: install with `easy_install cherrypy`
  - Linux (Ubuntu) use `sudo apt-get install python-cherrypy`
- SOAPLIB 0.8.1:
  - Windows: `Execute soaplib-0.8.1.win32.exe`
  - Linux: Unpack the zip archive and run `sudo python setup.py install`

Test installation:

- Open a terminal
- Execute "ipython"
- Execute "import cherrypy, lxml, soaplib" no errors or warnings should be generated
- Exit ipython with CTRL-D
- change directory to supplied soaplib tests e.g. "cd e:\cbmpy\tests\soaplib"
- Execute "python binary_test.py"
- Execute "python primitive_test.py"

All tests should pass.

1.8.2 PySCeS-CBM Mariner (http://cbmpy.sourceforge.net)

Download and install the latest version (0.7.4 or newer is required for CBMPy 0.7+):

- Windows: `Execute pyscesmariner-0.7.7.zip`
- Linux: unpack the archive and run `sudo python setup.py install`

To test installation, on Linux execute the commands in *run_server.bat* from the terminal directly.

- Open two terminals and in both
- Change directory to supplied PySCeS-CBM Mariner tests e.g. `cd e:\\cbmpy\\tests\\pyscesmariner`
- In terminal one `Execute run_server.bat`

Which should now display:

```
E:\\cbmpy\\tests\\pyscesmariner>python cbm_server_demo.py
Mariner using E:\\cbmpy\\tests\\pyscesmariner as a working directory
Mariner server name: 10.0.2.15
Mariner using port: 31313
```

```
Welcome to the PySCeS Constraint Based Modelling Toolkit (0.7.4)

<snipped>

Multiple Environment Module (0.6.2 [r1147])

PySCeSCBM/Mariner initialising ... this console is now blocked
```

In terminal two:

- Execute `python cbm_client_demo.py`

This should end without errors and display `done.` Congratulations you have successfully installed CBMPy and PySCeS-CBM Mariner!

CHAPTER

TWO

INTRODUCTION

PySCeS CBMPy is a new platform for constraint based modelling and analysis. It has been designed using principles developed in the PySCeS simulation software project: usability, flexibility and accessibility. Its architecture is both extensible and flexible using data structures that are intuitive to the biologist (metabolites, reactions, compartments) while transparently translating these into the underlying mathematical structures used in advanced analysis (LP's, MILP's).

PySCeS CBMPy implements popular analyses such as FBA, FVA, element/charge balancing, network analysis and model editing as well as advanced methods developed specifically for the ecosystem modelling: minimal distance methods, flux minimization and input selection.

To cater for a diverse range of modelling needs PySCeS CBMPy supports user interaction via:

- interactive console, scripting for advanced use or as a library for software development
- GUI, for quick access to a visual representation of the model, analysis methods and annotation tools
- SOAP based web services: using the Mariner framework much high level functionality is exposed for integration into web tools

For more information on the development and use of PySCeS CBMPy visit the website (http:cbmpy.sourceforge.net) for up to date information and feel free to contact the development team (bgoli@users.sourceforge.net).

CHAPTER

THREE

CBMPY MODULE REFERENCE

3.1 CBMPy: CBCommon module

PySCeS Constraint Based Modelling (http://cbmpy.sourceforge.net) Copyright (C) 2009-2017 Brett G. Olivier, VU University Amsterdam, Amsterdam, The Netherlands

This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program. If not, see <http://www.gnu.org/licenses/>

Author: Brett G. Olivier Contact email: bgoli@users.sourceforge.net Last edit: $Author: bgoli $ ($Id: CBCommon.py 575 2017-04-13 12:18:44Z bgoli $)

class cbmpy.CBCommon.**ComboGen**
Generate sets of unique combinations

cbmpy.CBCommon.**checkChemFormula**(*cf*, *quiet=False*)
Checks whether a string conforms to a Chemical Formula C3Br5 etc, returns True/False. Please see the SBML Level 3 specification and http://wikipedia.org/wiki/Hill_system for more information.

- *cf* a string that contains a formula to check
- *quiet* [default=False] do not print error messages

cbmpy.CBCommon.**checkId**(*s*)
Checks the validity of the string to see if it conforms to a C variable. Returns true/false

- *s* a string

cbmpy.CBCommon.**extractGeneIdsFromString**(*g*, *return_clean_gpr=False*)
Extract and return a list of gene names from a gene association string formulation

- *g* a COBRA style gene association string
- *return_clean_gpr* [default=False] in addition to the list returns the "cleaned" GPR string

cbmpy.CBCommon.**fixId**(*s*, *replace=None*)
Checks a string (Sid) to see if it is a valid C style variable. first letter must be an underscore or letter, the rest should be alphanumeric or underscore.

- *s* the string to test
- *replace* [None] default is to leave out offensive character, otherwise replace with this one

cbmpy.CBCommon.**parseGeneAssociation**(*gs*)
Parse a COBRA style gene association into a nested list.

- *gs* a string containing a gene association

cbmpy.CBCommon.**processSpeciesChargeChemFormulaAnnot**(*s*, *getFromName=False*, *overwriteChemFormula=False*, *overwriteCharge=False*)
Disambiguate the chemical formula from either the Notes or the overloaded name

- *s* a species object
- *getFromName* [default=False] whether to try strip the chemical formula from the name (old COBRA style)
- *overwriteChemFormula* [default=False]
- *overwriteCharge* [default=False]

3.2 CBMPy: CBConfig module

PySCeS Constraint Based Modelling (http://cbmpy.sourceforge.net) Copyright (C) 2009-2017 Brett G. Olivier, VU University Amsterdam, Amsterdam, The Netherlands

This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program. If not, see <http://www.gnu.org/licenses/>

Author: Brett G. Olivier Contact email: bgoli@users.sourceforge.net Last edit: $Author: bgoli $ ($Id: CBConfig.py 575 2017-04-13 12:18:44Z bgoli $)

cbmpy.CBConfig.**current_version**()
Return the current CBMPy version as a string

cbmpy.CBConfig.**current_version_tuple**()
Return the current CBMPy version as a tuple (x, y, z)

3.3 CBMPy: CBCPLEX module

PySCeS Constraint Based Modelling (http://cbmpy.sourceforge.net) Copyright (C) 2009-2017 Brett G. Olivier, VU University Amsterdam, Amsterdam, The Netherlands

This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program. If not, see <http://www.gnu.org/licenses/>

Author: Brett G. Olivier Contact email: bgoli@users.sourceforge.net Last edit: $Author: bgoli $ ($Id: CBCPLEX.py 575 2017-04-13 12:18:44Z bgoli $)

cbmpy.CBCPLEX.**cplx_FluxVariabilityAnalysis**(*fba*, *selected_reactions=None*, *pre_opt=True*, *tol=None*, *objF2constr=True*, *rhs_sense='lower'*, *optPercentage=100.0*, *work_dir=None*, *quiet=True*, *debug=False*, *oldlpgen=False*, *markupmodel=True*, *default_on_fail=False*, *roundoff_span=10*, *method='o'*)

Perform a flux variability analysis on an fba model:

- *fba* an FBA model object
- *selected reactions* [default=None] means use all reactions otherwise use the reactions listed here
- *pre_opt* [default=True] attempt to presolve the FBA and report its results in the ouput, if this is disabled and *objF2constr* is True then the rid/value of the current active objective is used
- *tol* [default=None] do not floor/ceiling the objective function constraint, otherwise round of to *tol*
- *rhs_sense* [default='lower'] means objC >= objVal the inequality to use for the objective constraint can also be *upper* or *equal*
- *optPercentage* [default=100.0] means the percentage optimal value to use for the RHS of the objective constraint: optimal_value*(optPercentage/100.0)
- *work_dir* [default=None] the FVA working directory for temporary files default = cwd+fva
- *debug* [default=False] if True write out all the intermediate FVA LP's into work_dir
- *quiet* [default=False] if enabled, supress CPLEX output
- *objF2constr* [default=True] add the model objective function as a constraint using rhs_sense etc. If this is True with pre_opt=False then the id/value of the active objective is used to form the constraint
- *markupmodel* [default=True] add the values returned by the fva to the reaction.fva_min and reaction.fva_max
- *default_on_fail* [default=False] if *pre_opt* is enabled replace a failed minimum/maximum with the solution value
- *roundoff_span* [default=10] number of digits is round off (not individual min/max values)

- *method* [default='o'] choose the CPLEX method to use for solution, default is automatic. See CPLEX reference manual for details
  - 'o': auto
  - 'p': primal
  - 'd': dual
  - 'b': barrier (no crossover)
  - 'h': barrier
  - 's': sifting
  - 'c': concurrent

Returns an array with columns: Reaction, Reduced Costs, Variability Min, Variability Max, abs(Max-Min), MinStatus, MaxStatus and a list containing the row names.

cbmpy.CBCPLEX.**cplx_MinimizeNumActiveFluxes**(*fba*, *selected_reactions=None*, *pre_opt=True*, *tol=None*, *objF2constr=True*, *rhs_sense='lower'*, *optPercentage=100.0*, *work_dir=None*, *quiet=False*, *debug=False*, *objective_coefficients=None*, *return_lp_obj=False*, *populate=None*, *oldlpgen=False*)

Minimize the sum of active fluxes, updates the model with the values of the solution and returns the value of the MILP objective function (not the model objective function which remains unchanged). If population mode is activated output is as described below:

Min: sum(Bi) Bi = 0 -> Ci Ji = 0

Such that: NJi = 0 Jbio = opt

where: Binary Bi

Arguments:

- *fba* an FBA model object
- *selected reactions* [default=None] means use all reactions otherwise use the reactions listed here
- *pre_opt* [default=True] attempt to presolve the FBA and report its results in the ouput, if this is diabled and *objF2constr* is True then the vid/value of the current active objective is used
- *tol* [default=None] do not floor/ceiling the objective function constraint, otherwise round of to *tol*
- *rhs_sense* [default='lower'] means objC >= objVal the inequality to use for the objective constraint can also be *upper* or *equal* Note this does not necessarily mean the upper or lower bound, although practically it will. If in doubt use *equal*
- *optPercentage* [default=100.0] means the percentage optimal value to use for the RHS of the objective constraint: optimal_value * (optPercentage/100.0)
- *work_dir* [default=None] the MSAF working directory for temporary files default = cwd+fva
- *debug* [default=False] if True write out all the intermediate MSAF LP's into work_dir

- *quiet* [default=False] if enabled supress CPLEX output
- *objF2constr* [default=True] add the model objective function as a constraint using rhs_sense etc. If this is True with pre_opt=False then the id/value of the active objective is used to form the constraint
- *objective_coefficients* [default=None] a dictionary of (reaction_id : float) pairs that provide the are introduced as objective coefficients to the absolute flux value. Note that the default value of the coefficient (non-specified) is +1.
- *return_lp_obj* [default=False] off by default when enabled it returns the CPLEX LP object
- *populate* [default=None] enable search algorithm to find multiple (sub)optimal solutions. Set with a tuple of (RELGAP=0.0, POPULATE_LIMIT=20, TIME_LIMIT=300) suggested values only. - *RELGAP* [default=0.0] relative gap to optimal solution - *POPULATE_LIMIT* [default=20] terminate when so many solutions have been found - *TIME_LIMIT* [default=300] terminate search after so many seconds important with higher values of *POPULATION_LIMIT*
- *with_reduced_costs* [default='uncsaled'] can be 'scaled', 'unscaled' or anything else which is None

With outputs:

- *mincnt* the objective function value OR
- *mincnt, cpx* the objective function and cplex model OR
- *populate_data, mincnt* a population data set OR
- *populate_data, mincnt, cpx* both the cps object and population data set

depending on selected flags.

cbmpy.CBCPLEX.**cplx_MinimizeSumOfAbsFluxes**(*fba*, *selected_reactions=None*, *pre_opt=True*, *tol=None*, *objF2constr=True*, *rhs_sense='lower'*, *optPercentage=100.0*, *work_dir=None*, *quiet=False*, *debug=False*, *objective_coefficients=None*, *return_lp_obj=False*, *oldlpgen=False*, *with_reduced_costs=None*, *method='o'*)

Minimize the sum of absolute fluxes sum(abs(J1) + abs(J2) + abs(J3) ... abs(Jn)) by adding two constraints per flux and a variable representing the absolute value:

Min: Ci abs_Ji Ji - abs_Ji <= 0 Ji + abs_Ji >= 0

Such that: NJi = 0 Jopt = opt

returns the value of the flux minimization objective function (not the model objective function which remains unchanged from)

Arguments:

- *fba* an FBA model object

- *selected reactions* [default=None] means use all reactions otherwise use the reactions listed here
- *pre_opt* [default=True] attempt to presolve the FBA and report its results in the ouput, if this is disabled and *objF2constr* is True then the vid/value of the current active objective is used
- *tol* [default=None] do not floor/ceiling the objective function constraint, otherwise round of to *tol*
- *rhs_sense* [default='lower'] means objC >= objVal the inequality to use for the objective constraint can also be *upper* or *equal*
- *optPercentage* [default=100.0] means the percentage optimal value to use for the RHS of the objective constraint: optimal_value*(optPercentage/100.0)
- *work_dir* [default=None] the MSAF working directory for temporary files default = cwd+fva
- *debug* [default=False] if True write out all the intermediate MSAF LP's into work_dir
- *quiet* [default=False] if enabled supress CPLEX output
- *objF2constr* [default=True] add the model objective function as a constraint using rhs_sense etc. If this is True with pre_opt=False then the id/value of the active objective is used to form the constraint
- *objective_coefficients* [default=None] a dictionary of (reaction_id : float) pairs that provide the are introduced as objective coefficients to the absolute flux value. Note that the default value of the coefficient (non-specified) is +1.
- *return_lp_obj* [default=False] off by default when enabled it returns the CPLEX LP object
- *with_reduced_costs* [default=None] if not None should be 'scaled' or 'unscaled'
- *method* [default='o'] choose the CPLEX method to use for solution, default is automatic. See CPLEX reference manual for details
  - 'o': auto
  - 'p': primal
  - 'd': dual
  - 'b': barrier (no crossover)
  - 'h': barrier
  - 's': sifting
  - 'c': concurrent

With outputs:

- *fba* an update instance of a CBModel. Note that the FBA model objective function value is the original value set as a constraint

cbmpy.CBCPLEX.**cplx_MultiFluxVariabilityAnalysis**(*lp*, *selected_reactions=None*, *tol=1e-10*, *rhs_sense='lower'*, *optPercentage=100.0*, *work_dir=None*, *debug=False*)

Perform a flux variability analysis on a multistate LP

- *lp* a multistate LP
- *selected reactions* [default=None] means use all reactions otherwise use the reactions listed here
- *pre_opt* [default=True] attempt to presolve the FBA and report its results in the ouput
- *tol* [default=1e-10] do floor/ceiling the objective function constraint, otherwise floor/ceil to *tol*
- *rhs_sense* [default='lower'] means objC >= objVal the inequality to use for the objective constraint can also be *upper* or *equal*
- *optPercentage* [default=100.0] means the percentage optimal value to use for the RHS of the objective constraint: optimal_value*(optPercentage/100.0)
- *work_dir* [default=None] the FVA working directory for temporary files default = cwd+fva
- *debug* [default=False] if True write out all the intermediate FVA LP's into work_dir
- *bypass* [default=False] bypass everything and only run the min/max on lp

and returns an array with columns:

```
Reaction, Reduced Costs, Variability Min, Variability Max, abs(Max-
↪Min), MinStatus, MaxStatus
```

and a list containing the row names.

cbmpy.CBCPLEX.**cplx_SolveMILP**(*c*, *auto_mipgap=False*)

Solve and MILP

- *auto_mipgap* auto decrease mipgap until mipgap == absmipgap

cbmpy.CBCPLEX.**cplx_WriteFVAtoCSV**(*pid*, *fva*, *names*, *Dir=None*, *fbaObj=None*)

Takes the resuls of a FluxVariabilityAnalysis method and writes it to a nice csv file. Note this method has been refactored to *CBWrite.WriteFVAtoCSV()*.

- *pid* filename_base for the CSV output
- *fva* FluxVariabilityAnalysis() OUTPUT_ARRAY
- *names* FluxVariabilityAnalysis() OUTPUT_NAMES
- *Dir* [default=None] if set the output directory for the csv files
- *fbaObj* [default=None] if supplied adds extra model information into the output tables

cbmpy.CBCPLEX.**cplx_analyzeModel**(*f*, *lpFname=None*, *return_lp_obj=False*, *with_reduced_costs='unscaled'*, *with_sensitivity=False*, *del_intermediate=False*, *build_n=True*, *quiet=False*, *oldlpgen=False*, *method='o'*)

Optimize a model and add the result of the optimization to the model object (e.g. *reaction.value*, *objectiveFunction.value*). The stoichiometric matrix is automatically generated. This is a common function available in all solver interfaces. By default returns the objective function value

- *f* an instantiated PySCeSCBM model object
- *lpFname* [default=None] the name of the intermediate LP file. If not specified no LP file is produced

- *return_lp_obj* [default=False] off by default when enabled it returns the CPLEX LP object
- *with_reduced_costs* [default='unscaled'] calculate and add reduced cost information to mode this can be: 'unscaled' or 'scaled' or anything else which is interpreted as 'None'. Scaled means s_rcost = (r.reduced_cost*rval)/obj_value
- *with_sensitivity* [default=False] add solution sensitivity information (not yet implemented)
- *del_intermediate* [default=False] redundant except if output file is produced and deleted (not useful)
- *build_n* [default=True] generate stoichiometry from the reaction network (reactions/reagents/species)
- *quiet* [default=False] suppress cplex output
- *method* [default='o'] choose the CPLEX method to use for solution, default is automatic. See CPLEX reference manual for details
  - 'o': auto
  - 'p': primal
  - 'd': dual
  - 'b': barrier (no crossover)
  - 'h': barrier
  - 's': sifting
  - 'c': concurrent

cbmpy.CBCPLEX.**cplx_constructLPfromFBA**(*fba*, *fname=None*)

Create a CPLEX LP in memory. - *fba* an FBA object - *fname* optional filename if defined writes out the constructed lp

cbmpy.CBCPLEX.**cplx_fixConSense**(*operator*)

Fixes the sense of inequality operators, returns corrected sense symbol

- *operator* the operator to check

cbmpy.CBCPLEX.**cplx_func_GetCPXandPresolve**(*fba*, *pre_opt*, *objF2constr*, *quiet=False*, *oldlpgen=False*, *with_reduced_costs='unscaled'*, *method='o'*)

This is a utility function that does a presolve for FVA, MSAF etc. Generates properly formatted empty objects if pre_opt == False

- *pre_opt* a boolean
- *fba* a CBModel object
- *objF2constr* add objective function as constraint
- *quiet* [default=False] supress cplex output
- *with_reduced_costs* [default='unscaled'] can be 'scaled' or 'unscaled'
- *method* [default='o'] choose the CPLEX method to use for solution, default is automatic. See CPLEX reference manual for details
  - 'o': auto

- 'p': primal
- 'd': dual
- 'b': barrier (no crossover)
- 'h': barrier
- 's': sifting
- 'c': concurrent

Returns: pre_sol, pre_oid, pre_oval, OPTIMAL_PRESOLUTION, REDUCED_COSTS

cbmpy.CBCPLEX.**cplx_func_SetObjectiveFunctionAsConstraint**(*cpx*, *rhs_sense*, *oval*, *tol*, *optPercentage*)

Take the objective function and "optimum" value and add it as a constraint

- *cpx* a cplex object
- *oval* the objective value
- *tol* [default=None] do not floor/ceiling the objective function constraint, otherwise round of to *tol*
- *rhs_sense* [default='lower'] means objC >= objVal the inequality to use for the objective constraint can also be *upper* or *equal*
- *optPercentage* [default=100.0] means the percentage optimal value to use for the RHS of the objective constraint: optimal_value*(optPercentage/100.0)

cbmpy.CBCPLEX.**cplx_getCPLEXModelFromLP**(*lptFile*, *Dir=None*)

Load a LPT (CPLEX format) file and return a CPLX LP model

- *lptfile* an CPLEX LP format file
- *Dir* an optional directory

cbmpy.CBCPLEX.**cplx_getDualValues**(*c*)

Get the get the dual values of the solution

- *c* a CPLEX LP

Output is a dictionary of {name : value} pairs

cbmpy.CBCPLEX.**cplx_getModelFromLP**(*lptFile*, *Dir=None*)

Load a LPT (CPLEX format) file and return a CPLX LP model

- *lptfile* an CPLEX LP format file
- *Dir* an optional directory

cbmpy.CBCPLEX.**cplx_getModelFromObj**(*fba*)

Return a CPLEX object from a FBA model object (via LP file)

cbmpy.CBCPLEX.**cplx_getOptimalSolution**(*c*)

From a CPLX model extract a tuple of solution, ObjFuncName and ObjFuncVal

cbmpy.CBCPLEX.**cplx_getOptimalSolution2**(*c*, *names*)

From a CPLX model extract a tuple of solution, ObjFuncName and ObjFuncVal

cbmpy.CBCPLEX.**cplx_getReducedCosts**(*c*, *scaled=False*)
Extract ReducedCosts from LP and return as a dictionary 'Rid' : reduced cost

- *c* a cplex LP object
- *scaled* scale the reduced cost by the optimal flux value

cbmpy.CBCPLEX.**cplx_getSensitivities**(*c*)
Get the sensitivities of each constraint on the objective function with inpt

- *c* a CPLEX LP

Output is a tuple of bound and objective sensitivities where the objective sensitivity is described in the CPLEX reference manual as:

```
... the objective sensitivity shows each variable, its reduced cost␣
↪and the range over
which its objective function coefficient can vary without forcing a␣
↪change
in the optimal basis. The current value of each objective coefficient␣
↪is
also displayed for reference.

- *objective coefficient sensitivity* {flux : (reduced_cost, lower_
↪obj_sensitivity, coeff_value, upper_obj_sensitivity)}
- *rhs sensitivity* {constraint : (low, value, high)}
- *bound sensitivity ranges* {flux : (lb_low, lb_high, ub_low, ub_
↪high)}
```

cbmpy.CBCPLEX.**cplx_getShadowPrices**(*c*)
Returns a dictionary of shadow prices containing 'Rid' : (lb, rhs, ub)

- *c* a cplex LP object

cbmpy.CBCPLEX.**cplx_getSolutionStatus**(*c*)
Returns one of:

- *LPS_OPT*: solution is optimal;
- *LPS_FEAS*: solution is feasible;
- *LPS_INFEAS*: solution is infeasible;
- *LPS_NOFEAS*: problem has no feasible solution;
- *LPS_UNBND*: problem has unbounded solution;
- *LPS_UNDEF*: solution is undefined.
- *LPS_NONE*: no solution

cbmpy.CBCPLEX.**cplx_runInputScan**(*fba*, *exDict*, *wDir*, *input_lb=-10.0*, *input_ub=0.0*, *writeHformat=False*, *rationalLPout=False*)
scans all inputs

cbmpy.CBCPLEX.**cplx_setFBAsolutionToModel**(*fba*, *lp*, *with_reduced_costs='unscaled'*)
Sets the FBA solution from a CPLEX solution to an FBA object

- *fba* and fba object
- *lp* a CPLEX LP object

- *with_reduced_costs* [default='unscaled'] calculate and add reduced cost information to mode this can be: 'unscaled' or 'scaled' or anything else which is interpreted as None. Scaled is: s_rcost = (r.reduced_cost*rval)/obj_value

cbmpy.CBCPLEX.**cplx_setMIPGapTolerance**(*c*, *tol*)
Sets the the relative MIP gap tolerance

cbmpy.CBCPLEX.**cplx_setObjective**(*c*, *pid*, *expr=None*, *sense='maximize'*, *reset=True*)
Set a new objective function note that there is a major memory leak in *c.variables.get_names()* whch is used when reset=True. If this is a problem use cplx_setObjective2 which takes *names* as an input:

- *c* a CPLEX LP object
- *pid* the r_id of the flux to be optimized
- *expr* a list of (coefficient, flux) pairs
- *sense* 'maximize'/'minimize'
- *reset* [default=True] reset all objective function coefficients to zero

cbmpy.CBCPLEX.**cplx_setObjective2**(*c*, *pid*, *names*, *expr=None*, *sense='maximize'*, *reset=True*)
Set a new objective function. This is a workaround function to avoid the e is a major memory leak in *c.variables.get_names()* whch is used in cplx_setObjective() when reset=True.

cbmpy.CBCPLEX.**cplx_setOutputStreams**(*lp*, *mode='default'*)
Sets the noise level of the solver, mode can be one of:

- *None* silent i.e. no output
- '*file'* set solver to silent and output logs to *CPLX_RESULT_STREAM_FILE* cplex_output.log
- '*iostream'* set solver to silent and output logs to *CPLX_RESULT_STREAM_IO* csio
- '*default'* or anything else noisy with full output closes STREAM_IO and STREAM_FILE (default)

cbmpy.CBCPLEX.**cplx_setSolutionStatusToModel**(*m*, *lp*)
Sets the lp solutions status to the CBMPy model

cbmpy.CBCPLEX.**cplx_singleGeneScan**(*fba*, *r_off_low=0.0*, *r_off_upp=0.0*, *optrnd=8*, *altout=False*)
Perform a single gene deletion scan

- *fba* a model object
- *r_off_low* the lower bound of a deactivated reaction
- *r_off_upp* the upper bound of a deactivated reaction
- *optrnd* [default=8] round off the optimal value
- *altout* [default=False] by default return a list of gene:opt pairs, alternatively (True) return an extended result set including gene groups, optima and effect map

cbmpy.CBCPLEX.**cplx_writeLPsolution**(*fba_sol*, *objf_name*, *fname*, *Dir=None*, *separator=', '*)
This function writes the optimal solution, produced wth *cplx_getOptimalSolution* to file

- *fba_sol* a dictionary of Flux : value pairs
- *objf_name* the objective function flux id
- *fname* the output filename
- *Dir* [default=None] use directory if not None
- *separator* [default=','] the column separator

cbmpy.CBCPLEX.**cplx_writeLPtoLPTfile**(*c*, *filename*, *title=None*, *Dir=None*)
Write out a CPLEX model as an LP format file

cbmpy.CBCPLEX.**getReducedCosts**(*fba*)
Get a dictionary of reduced costs for each reaction/flux

cbmpy.CBCPLEX.**setReducedCosts**(*fba*, *reduced_costs*)
For each reaction/flux, sets the attribute "reduced_cost" from a dictionary of reduced costs

- *fba* an fba object
- *reduced_costs* a dictionary of {reaction : value} pairs

3.4 CBMPy: CBDataStruct module

PySCeS Constraint Based Modelling (http://cbmpy.sourceforge.net) Copyright (C) 2009-2017 Brett G. Olivier, VU University Amsterdam, Amsterdam, The Netherlands

This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program. If not, see <http://www.gnu.org/licenses/>

Author: Brett G. Olivier Contact email: bgoli@users.sourceforge.net Last edit: $Author: bgoli $ ($Id: CBDataStruct.py 575 2017-04-13 12:18:44Z bgoli $)

class cbmpy.CBDataStruct.**MIRIAMannotation**
The MIRIAMannotation class MIRIAM annotations: Biological Qualifiers

addIDorgURI(*qual*, *uri*)
Add a URI directly into a qualifier collection:

- *qual* a Biomodels biological qualifier e.g. "is" "isEncodedBy"
- *uri* the complete identifiers.org uri e.g. http://identifiers.org/chebi/CHEBI:58088

addMIRIAMannotation(*qual*, *entity*, *mid*)
Add a qualified MIRIAM annotation or entity:

- *qual* a Biomodels biological qualifier e.g. "is" "isEncodedBy"
- *entity* a MIRIAM resource entity e.g. "ChEBI"
- *mid* the entity id e.g. CHEBI:17158

checkEntity(*entity*)
Check an entity entry, this is a MIRIAM resource name: "chEBI". The test is case insensitive and will correct the case of wrongly capitalised entities automatically. If the entity is not recognised then a list of possible candidates based on the first letters of the input is displayed.

•*entity* a MIRIAM resource entity e.g. "ChEBI"

checkEntityPattern(*entity*)
For an entity key compile the pattern to a regex, if necessary.

•*entity* a MIRIAM resource entity

checkId(*entity*, *mid*)
Check that a entity id e.g. CHEBI:17158

•*mid* the entity id e.g. CHEBI:17158

deleteMIRIAMannotation(*qual*, *entity*, *mid*)
Deletes a qualified MIRIAM annotation or entity:

•*qual* a Biomodels biological qualifier e.g. "is" "isEncodedBy"

•*entity* a MIRIAM resource entity e.g. "ChEBI"

•*mid* the entity id e.g. CHEBI:17158

getAllMIRIAMUris()
Return a dictionary of qualifiers that contain ID.org URL'S

getAndViewUrisForQualifier(*qual*)
Retrieve all url's associated with qualifier and attempt to open them all in a new browser tab

•*qual* the qualifier e.g. "is" or "isEncoded"

getMIRIAMUrisForQualifier(*qual*)
Return all list of urls associated with qualifier:

•*qual* the qualifier e.g. "is" or "isEncoded"

viewURL(*url*)
This will try to open the URL in a new tab of the default webbrowser

•*url* the url

class cbmpy.CBDataStruct.**StructMatrix**(*array*, *ridx*, *cidx*, *row=None*, *col=None*)
This class is specifically designed to store structural matrix information give it an array and row/col index permutations it can generate its own row/col labels given the label src.

getColsByIdx(**args*)
Return the columns referenced by index (1,3,5)

getColsByName(**args*)
Return the columns referenced by label ('s','x','d')

getIndexes(*axis='all'*)
Return the matrix indexes ([rows],[cols]) where axis='row'/'col'/'all'

getLabels(*axis='all'*)
Return the matrix labels ([rows],[cols]) where axis='row'/'col'/'all'

getRowsByIdx(**args*)
Return the rows referenced by index (1,3,5)

getRowsByName (**args*)
Return the rows referenced by label ('s','x','d')

setCol (*src*)
Assuming that the col index array is a permutation (full/subset) of a source label array by supplying that src to setCol maps the row labels to cidx and creates self.col (col label list)

setRow (*src*)
Assuming that the row index array is a permutation (full/subset) of a source label array by supplying that source to setRow it maps the row labels to ridx and creates self.row (row label list)

class cbmpy.CBDataStruct.**StructMatrixLP** (*array*, *ridx*, *cidx*, *row=None*, *col=None*, *rhs=None*, *operators=None*)
Adds some stuff to StructMatrix that makes it LP friendly

getCopy (*attr_str*, *deep=False*)
Return a copy of the attribute with name attr_str. Uses the copy module *copy.copy* or *copy.deepcopy*

- *attr_str* a string of the attribute name: 'row', 'col'
- *deep* [default=False] try to do a deepcopy. Use with caution see copy module docstring for details

3.5 CBMPy: CBGLPK module

PySCeS Constraint Based Modelling (http://cbmpy.sourceforge.net) Copyright (C) 2009-2017 Brett G. Olivier, VU University Amsterdam, Amsterdam, The Netherlands

This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program. If not, see <http://www.gnu.org/licenses/>

Author: Brett G. Olivier Contact email: bgoli@users.sourceforge.net Last edit: $Author: bgoli $ ($Id: CBGLPK.py 575 2017-04-13 12:18:44Z bgoli $)

cbmpy.CBGLPK.**getReducedCosts** (*fba*)
Get a dictionary of reduced costs for each reaction/flux

cbmpy.CBGLPK.**glpk_FluxVariabilityAnalysis** (*fba*, *selected_reactions=None*, *pre_opt=True*, *tol=None*, *objF2constr=True*, *rhs_sense='lower'*, *optPercentage=100.0*, *work_dir=None*, *quiet=True*, *debug=False*, *oldlpgen=False*, *markupmodel=True*, *default_on_fail=False*, *roundoff_span=10*, *method='s'*)

Perform a flux variability analysis on an fba model:

- *fba* an FBA model object
- *selected reactions* [default=None] means use all reactions otherwise use the reactions listed here
- *pre_opt* [default=True] attempt to presolve the FBA and report its results in the ouput, if this is disabled and *objF2constr* is True then the vid/value of the current active objective is used
- *tol* [default=None] do not floor/ceiling the objective function constraint, otherwise round of to *tol*
- *rhs_sense* [default='lower'] means objC >= objVal the inequality to use for the objective constraint can also be *upper* or *equal*
- *optPercentage* [default=100.0] means the percentage optimal value to use for the RHS of the objective constraint: optimal_value*(optPercentage/100.0)
- *work_dir* [default=None] the FVA working directory for temporary files default = cwd+fva
- *debug* [default=False] if True write out all the intermediate FVA LP's into work_dir
- *quiet* [default=False] if enabled supress CPLEX output
- *objF2constr* [default=True] add the model objective function as a constraint using rhs_sense etc. If this is True with pre_opt=False then the id/value of the active objective is used to form the constraint
- *markupmodel* [default=True] add the values returned by the fva to the reaction.fva_min and reaction.fva_max
- *default_on_fail* [default=False] if *pre_opt* is enabled replace a failed minimum/maximum with the solution value
- *roundoff_span* [default=10] number of digits is round off (not individual min/max values)
- *method* [default='s'] select the GLPK solver method, see the GLPK documentation for details
  - 's': simplex
  - 'i': interior
  - 'e': exact

Returns an array with columns Reaction, Reduced Costs, Variability Min, Variability Max, abs(Max-Min), MinStatus, MaxStatus and a list containing the row names.

cbmpy.CBGLPK.**glpk_MinimizeSumOfAbsFluxes**(*fba*, *selected_reactions=None*, *pre_opt=True*, *tol=None*, *objF2constr=True*, *rhs_sense='lower'*, *optPercentage=100.0*, *work_dir=None*, *quiet=False*, *debug=False*, *objective_coefficients={}*, *return_lp_obj=False*, *oldlpgen=False*, *with_reduced_costs=None*, *method='s'*)

Minimize the sum of absolute fluxes sum(abs(J1) + abs(J2) + abs(J3) ... abs(Jn)) by adding two

constraints per flux and a variable representing the absolute value:

Min: Ci abs_Ji Ji - abs_Ji <= 0 Ji + abs_Ji >= 0

Such that: NJi = 0 Jopt = opt

returns the value of the flux minimization objective function (not the model objective function which remains unchanged from)

Arguments:

- *fba* an FBA model object
- *selected reactions* [default=None] means use all reactions otherwise use the reactions listed here
- *pre_opt* [default=True] attempt to presolve the FBA and report its results in the ouput, if this is disabled and *objF2constr* is True then the vid/value of the current active objective is used
- *tol* [default=None] do not floor/ceiling the objective function constraint, otherwise round of to *tol*
- *rhs_sense* [default='lower'] means objC >= objVal the inequality to use for the objective constraint can also be *upper* or *equal*
- *optPercentage* [default=100.0] means the percentage optimal value to use for the RHS of the objective constraint: optimal_value*(optPercentage/100.0)
- *work_dir* [default=None] the MSAF working directory for temporary files default = cwd+fva
- *debug* [default=False] if True write out all the intermediate MSAF LP's into work_dir
- *quiet* [default=False] if enabled supress CPLEX output
- *objF2constr* [default=True] add the model objective function as a constraint using rhs_sense etc. If this is True with pre_opt=False then the id/value of the active objective is used to form the constraint
- *objective_coefficients* [default={}] a dictionary of (reaction_id : float) pairs that provide the are introduced as objective coefficients to the absolute flux value. Note that the default value of the coefficient (non-specified) is +1.
- *return_lp_obj* [default=False] off by default when enabled it returns the CPLEX LP object
- *with_reduced_costs* [default=None] if not None should be 'scaled' or 'unscaled'
- *method* [default='s'] select the GLPK solver method, see the GLPK documentation for details
  - 's': simplex
  - 'i': interior
  - 'e': exact

With outputs:

- *fba* an update instance of a CBModel. Note that the FBA model objective function value is the original value set as a constraint

cbmpy.CBGLPK.**glpk_Solve**(*lp*, *method='s'*)

Solve the LP and create a status attribute with the solution status

- *method* [default='s'] 's' = simplex, 'i' = interior, 'e' = exact

GLPK solver options can be set in the dictionary GLPK_CFG

cbmpy.CBGLPK.**glpk_analyzeModel**(*f*, *lpFname=None*, *return_lp_obj=False*, *with_reduced_costs='unscaled'*, *with_sensitivity=False*, *del_intermediate=False*, *build_n=True*, *quiet=False*, *oldlpgen=False*, *method='s'*)

Optimize a model and add the result of the optimization to the model object (e.g. *reaction.value*, *objectiveFunction.value*). The stoichiometric matrix is automatically generated. This is a common function available in all solver interfaces. By default returns the objective function value

- *f* an instantiated PySCeSCBM model object
- *lpFname* [default=None] the name of the intermediate LP file saved when this has a string value.
- *return_lp_obj* [default=False] off by default when enabled it returns the PyGLPK LP object
- *with_reduced_costs* [default='unscaled'] calculate and add reduced cost information to mode this can be: 'unscaled' or 'scaled' or anything else which is interpreted as 'None'. Scaled means s_rcost = (r.reduced_cost*rval)/obj_value
- *with_sensitivity* [default=False] add solution sensitivity information (not yet implemented)
- *del_intermediate* [default=False] delete the intermediary files after updating model object, useful for server applications
- *build_n* [default=True] generate stoichiometry from the reaction network (reactions/reagents/species)
- *quiet* [default=False] suppress glpk output
- *method* [default='s'] select the GLPK solver method, see the GLPK documentation for details
  - 's': simplex
  - 'i': interior
  - 'e': exact

cbmpy.CBGLPK.**glpk_constructLPfromFBA**(*fba*, *fname=None*)

Create a GLPK LP in memory. - *fba* an FBA object - *fname* optional filename if defined writes out the constructed lp

cbmpy.CBGLPK.**glpk_func_GetCPXandPresolve**(*fba*, *pre_opt*, *objF2constr*, *quiet=False*, *oldlpgen=True*, *with_reduced_costs='unscaled'*, *method='s'*)

This is a utility function that does a presolve for FVA, MSAF etc. Generates properly formatted empty objects if pre_opt == False

- *pre_opt* a boolean
- *fba* a CBModel object
- *objF2constr* add objective function as constraint
- *quiet* [default=False] supress cplex output
- *with_reduced_costs* [default='unscaled'] can be 'scaled' or 'unscaled'

•*method* [default='s'] select the GLPK solver method, see the GLPK documentation for details

– 's': simplex

– 'i': interior

– 'e': exact

Returns: pre_sol, pre_oid, pre_oval, OPTIMAL_PRESOLUTION, REDUCED_COSTS

cbmpy.CBGLPK.**glpk_func_SetObjectiveFunctionAsConstraint**(*cpx*, *rhs_sense*, *oval*, *tol*, *optPercentage*)

Take the objective function and "optimum" value and add it as a constraint

- *cpx* a cplex object
- *oval* the objective value
- *tol* [default=None] do not floor/ceiling the objective function constraint, otherwise round of to *tol*
- *rhs_sense* [default='lower'] means objC >= objVal the inequality to use for the objective constraint can also be *upper* or *equal*
- *optPercentage* [default=100.0] means the percentage optimal value to use for the RHS of the objective constraint: optimal_value*(optPercentage/100.0)

cbmpy.CBGLPK.**glpk_getOptimalSolution**(*c*)

From a GLPK model extract a tuple of solution, ObjFuncName and ObjFuncVal

cbmpy.CBGLPK.**glpk_getReducedCosts**(*c*, *scaled=False*)

Extract ReducedCosts from LP and return as a dictionary 'Rid' : reduced cost

•*c* a GLPK LP object

•*scaled* scale the reduced cost by the optimal flux value

cbmpy.CBGLPK.**glpk_getSolutionStatus**(*lp*)

Returns one of:

•*LPS_OPT*: solution is optimal;

•*LPS_FEAS*: solution is feasible;

•*LPS_INFEAS*: solution is infeasible;

•*LPS_NOFEAS*: problem has no feasible solution;

•*LPS_UNBND*: problem has unbounded solution;

•*LPS_UNDEF*: solution is undefined.

cbmpy.CBGLPK.**glpk_setFBAsolutionToModel**(*fba*, *lp*, *with_reduced_costs='unscaled'*)

Sets the FBA solution from a CPLEX solution to an FBA object

•*fba* and fba object

•*lp* a CPLEX LP object

- *with_reduced_costs* [default='unscaled'] calculate and add reduced cost information to mode this can be: 'unscaled' or 'scaled' or anything else which is interpreted as None. Scaled is: s_rcost = (r.reduced_cost*rval)/obj_value

cbmpy.CBGLPK.**glpk_setObjective**(*c*, *oid*, *expr=None*, *sense='maximize'*, *reset=True*)

Set a new objective function note that there is a major memory leak in *c.variables.get_names()* whch is used when reset=True. If this is a problem use cplx_setObjective2 which takes *names* as an input:

- *c* a GLPK LP object
- *oid* the r_id of the flux to be optimized
- *expr* a list of (coefficient, flux) pairs
- *sense* 'maximize'/'minimize'
- *reset* [default=True] reset all objective function coefficients to zero

cbmpy.CBGLPK.**glpk_setSingleConstraint**(*c*, *cid*, *expr=[]*, *sense='E'*, *rhs=0.0*)

Sets a new sigle constraint to a GLPK model

- *c* a GLPK instance
- *cid* the constraint id
- *expr* a list of (coefficient, name) pairs
- *sense* [default='G'] LGE
- *rhs* [default=0.0] the right hand side

cbmpy.CBGLPK.**glpk_setSolutionStatusToModel**(*m*, *lp*)

Sets the lp solutions status to the CBMPy model

cbmpy.CBGLPK.**glpk_writeLPtoLPTfile**(*c*, *filename*, *title=None*, *Dir=None*)

Write out a GLPK model as an LP format file

cbmpy.CBGLPK.**setReducedCosts**(*fba*, *reduced_costs*)

For each reaction/flux, sets the attribute "reduced_cost" from a dictionary of reduced costs

- *fba* an fba object
- *reduced_costs* a dictionary of {reaction : value} pairs

3.6 CBMPy: CBGUI module

You should have received a copy of the GNU General Public License along with this program. If not, see <http://www.gnu.org/licenses/>

Author: Brett G. Olivier Contact email: bgoli@users.sourceforge.net Last edit: $Author: bgoli $ ($Id: CBGUI.py 575 2017-04-13 12:18:44Z bgoli $)

cbmpy.CBGUI.**loadCBGUI**(*mod*, *version=2*)

Load an FBA model instance into the quick editor to view or change basic model properties

- *mod* a PySCeS CBMPy model instance

3.7 CBMPy: CBModel module

PySCeS Constraint Based Modelling (http://cbmpy.sourceforge.net) Copyright (C) 2009-2017 Brett G. Olivier, VU University Amsterdam, Amsterdam, The Netherlands

This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program. If not, see <http://www.gnu.org/licenses/>

Author: Brett G. Olivier Contact email: bgoli@users.sourceforge.net Last edit: $Author: bgoli $ ($Id: CBModel.py 575 2017-04-13 12:18:44Z bgoli $)

class cbmpy.CBModel.**Compartment**(*cid*, *name=None*, *size=1*, *dimensions=3*, *volume=None*)

A compartment

containsReactions()

Lists the species contained in this compartment

containsSpecies()

Lists the species contained in this compartment

getDimensions()

Get the compartment dimensions

getSize()

Get the compartment size

setDimensions(*dimensions*)

Get the compartment dimensions

- *dimensions* set the new compartment dimensions

setSize(*size*)

Set the compartment size

- *size* the new compartment size

class cbmpy.CBModel.**Fbase**

Base class for CB Model objects

addMIRIAMannotation(*qual*, *entity*, *mid*)
Add a qualified MIRIAM annotation or entity:

- *qual* a Biomodels biological qualifier e.g. "is" "isEncodedBy"
- *entity* a MIRIAM resource entity e.g. "ChEBI"
- *mid* the entity id e.g. CHEBI:17158 or fully qualifies url (if only_qual_uri)

addMIRIAMuri(*qual*, *uri*)
Add a qualified MIRIAM annotation or entity:

- *qual* a Biomodels biological qualifier e.g. "is" "isEncodedBy"
- *uri* the fully qualified entity id e.g. http://identifiers.org/chebi/CHEBI:12345 (no validity checking is done)

clone()
Return a clone of this object. Cloning performs a deepcop on the object which will also clone any objects that exist as attributes of this object, in other words an independent copy of the original. If this is not the desired behaviour override this method when subclassing or implement your own.

deleteAnnotation(*key*)
Unsets (deltes) an objects annotation with key

- *key* the annotation key

deleteMIRIAMannotation(*qual*, *entity*, *mid*)
Deletes a qualified MIRIAM annotation or entity:

- *qual* a Biomodels biological qualifier e.g. "is" "isEncodedBy"
- *entity* a MIRIAM resource entity e.g. "ChEBI"
- *mid* the entity id e.g. CHEBI:17158

getAnnotation(*key*)
Return the object annotation associated with:

- *key* the annotation key

getAnnotations()
Return the object annotation dictionary

getCompartmentId()
Return the compartment id where this element is located

getId()
Return the object ID.

getMIRIAMannotations()
Returns a dictionary of all MIRIAM annotations associated with this object or None of there are none defined.

getMetaId()
Return the object metaId.

getName()
Return the object name.

getNotes()
Return the object's notes

getPid()
Return the object ID.

getSBOterm()
Return the SBO term for this object.

hasAnnotation(*key*)
Returns a boolean representing the presence/absence of the key in the objext annotation

- *key* the annotation key

serialize(*protocol=0*)
Serialize object, returns a string by default

- ***protocol* [default=0] serialize to a string or binary if required,** see pickle module documentation for details

Reimplemented in Model

serializeToDisk(*filename*, *protocol=2*)
Serialize to disk using pickle protocol:

- *filename* the name of the output file
- ***protocol* [default=2] serialize to a string or binary if required,** see pickle module documentation for details

Reimplemented in Model

setAnnotation(*key*, *value*)
Set an objects annotation as a key : value pair.

- *key* the annotation key
- *value* the annotation value

setCompartmentId(*compartment*)
Set the compartment id where this element is located

setId(*fid*)
Sets the object Id

- *fid* a valid c variable style id string

Reimplemented by @Reaction, @Species

setMetaId(*mid=None*)
Sets the object Id

- *mid* [default=None] a valid c variable style metaid string, if None it will be set as meta+id

setName(*name*)
Set the object name:

- *name* the name string

setNotes(*notes*)
Sets the object's notes:

•*notes* the note string, should preferably be (X)HTML for SBML

setPid(*fid*)
Sets the object Id

•*fid* a valid c variable style id string

setSBOterm(*sbo*)
Set the SBO term for this object.

•*sbo* the SBOterm with format: SBO:nnnnnnn"

class cbmpy.CBModel.**FluxBound**(*fid*, *reaction*, *operation*, *value*)
A reaction fluxbound

getType()
Returns the *type* of FluxBound: 'lower', 'upper', 'equality' or None

getValue()
Returns the current value of the attribute (input/solution)

setReactionId(*react*)
Sets the reaction attribute of the FluxBound

setValue(*value*)
Sets the attribute ''value''

class cbmpy.CBModel.**FluxObjective**(*pid*, *reaction*, *coefficient=1*)
A weighted flux that appears in an objective function

NOTE: reaction is a string containing a reaction id

class cbmpy.CBModel.**Gene**(*gid*, *label=None*, *active=True*)
Contains all the information about a gene (or gene+protein construct depending on your philosophy)

TODO: I will change the whole Gene/GPR structure to a dictionary data structure on the model which should simplify this all significantly.

getLabel()
Returns the gene label

isActive()
Returns whether the gene is active or not

resetActivity()
Reset the gene to its default activity state

setActive()
Set the gene to be active

setInactive()
Set the gene to be inactive

setLabel(*label*)
Sets the gene label

class cbmpy.CBModel.**GeneProteinAssociation**(*gpid*, *protein*)
This class associates genes to proteins. TODO: I will change the whole Gene/GPR structure to a dictionary data structure on the model which should simplify this all significantly.

addAssociation(*assoc*)
Add a gene/protein association expression

addGeneref(*geneid*)
Add a gene reference to the list of gene references

- *geneid* a valid model Gene id

buildEvalFunc()
Builds a function which evaluates the gene expressions and evaluates to an integer using the following rules:

- True –> 1
- False –> 0
- and –> *
- or –> +

createAssociationAndGeneRefs(*assoc*, *altlabels=None*)
Evaluate the gene/protein association and add the genes necessary to evaluate it Note that this GPR should be added to a model with cmod.addGPRAssociation() before calling this method

- *assoc* the COBRA style gene protein association
- *altlabels* [default=None] a dictionary containing a label<–>id mapping

deleteGeneref(*gid*)
Deletes a gene reference

- *geneid* a valid model Gene id

evalAssociation()
Returns an integer value representing the logical associations or None.

getActiveGenes()
Return a list of active gene objects

getAssociationStr(*use_labels=False*)
Return the gene association string, alternatively return string with labels

- *use_lablels* [default=False] return the gene association string with labels rather than geneId's (FBCv2 issue)

getGene(*gid*)
Return a gene object with id

getGeneIds()
Return a list of gene id's

getGeneLabels()
Return a list of gene labels associated with this GPRass

getGenes()
Return a list of gene objects associated with this GPRass

getProtein()
Return the protein associated with this set of genes

isProteinActive()
This returns a boolean which indicates the result of evaluating the gene association. If the result is positive then the protein is expressed and *True* is returned, otherwise if the expression evaluates to a value of 0 then the protein is not expressed and *False* is returned.

setAllGenesActive()
Activate all genes in association

setAllGenesInactive()
Deactivates all genes in association

setGeneActive(*gid*)
Set a gene to be inactive

setGeneInactive(*gid*)
Set a gene to be inactive

setProtein(*protein*)
Sets the protein associated with this set of genes

class cbmpy.CBModel.**Group**(*pid*)
Container for SBML groups

addMember(*obj*)
Add member CBMPy object(s) to the group

- *obj* either a single, tuple or list of CBMPy objects

addSharedMIRIAMannotation(*qual*, *entity*, *mid*)
Add a qualified MIRIAM annotation or entity to the list of members (all) rather than the group itself:

- *qual* a Biomodels biological qualifier e.g. "is" "isEncodedBy"
- *entity* a MIRIAM resource entity e.g. "ChEBI"
- *mid* the entity id e.g. CHEBI:17158 or fully qualifies url (if only_qual_uri)

assignAllSharedPropertiesToMembers(*overwrite=False*)
Assigns all group shared properties (notes, annotations, MIRIAM annotations, SBO) to the group members.

- *overwrite* [default=False] overwrite the target notes if they are defined

assignSharedAnnotationToMembers()
This function merges or updates the group member objects annotations with the group shared annotation.

assignSharedMIRIAMannotationToMembers()
This function merges or updates the group member objects MIRIAM annotations with the group shared MIRIAM annotation.

assignSharedNotesToMembers(*overwrite=False*)
Assigns the group shared notes to the group members.

- *overwrite* [default=False] overwrite the target notes if they are defined

assignSharedSBOtermsToMembers(*overwrite=False*)
Assigns the group shared member SBO term to the group members.

- *overwrite* [default=False] overwrite the target SBO term if it is defined

clone()
Return a clone of this object. Note the for Groups this is a shallow copy, in that the reference objects themselves are not cloned only the group (and attributes)

deleteMember(*oid*)
Deletes a group member with group id.

- *oid* group member id

getKind()
Return the group kind

getMemberIDs(*as_set=False*)
Return the ids of the member objects.

- *as_set* return id's as a set rather than a list

getMembers(*as_set=False*)
Return the member objects of the group.

- *as_set* return objects as a set rather than a list

getSharedAnnotations()
Return a dictionary of the shared member annotations (rather than the group attribute).

getSharedMIRIAMannotations()
Return a dictionary of the shared member MIRIAM annotations (rather than the group attribute).

getSharedNotes()
Return the shared member notes (rather than the group attribute).

getSharedSBOterm()
Return the shared member SBO term (rather than the group attribute).

setKind(*kind*)
Sets the kind or type of the group, this must be one of: 'collection', 'partonomy', 'classification'.

- *kind* the kind

setSharedAnnotation(*key*, *value*)
Sets the list of members (all) annotation as a key : value pair.

- *key* the annotation key
- *value* the annotation value

setSharedNotes(*notes*)
Sets the group of objects notes attribute (all):

- *notes* the note string, should preferably be (X)HTML for SBML

setSharedSBOterm(*sbo*)
Set the SBO term for the the members of the group (all).

- *sbo* the SBOterm with format: "SBO:<7 digit integer>"

class cbmpy.CBModel.**GroupMemberAttributes**
Contains the shared attributes of the group members (equivalent to SBML annotation on ListOfMembers)

class cbmpy.CBModel.**Model**(*pid*)

Container for constraint based model, adds methods for manipulating:

- objectives
- constraints
- reactions
- species
- compartments
- groups
- parameters
- N a structmatrix object

addCompartment(*comp*)

Add an instantiated Compartment object to the CBM model

- *comp* an instance of the Compartment class

addFluxBound(*fluxbound*, *fbexists=None*)

Add an instantiated FluxBound object to the FBA model

- *fluxbound* an instance of the FluxBound class

addGPRAssociation(*gpr*, *update_idx=True*)

Add a GeneProteinAssociation instance to the model

- *gpr* an instantiated GeneProteinAssociation object

addGene(*gene*)

Add an instantiated Gene object to the FBA model

- *gene* an instance of the G class

addGroup(*obj*)

Add an instantiated group object to the model

- *obj* the Group instance

addMIRIAMannotation(*qual*, *entity*, *mid*)

Add a qualified MIRIAM annotation or entity:

- *qual* a Biomodels biological qualifier e.g. "is" "isEncodedBy"
- *entity* a MIRIAM resource entity e.g. "ChEBI"
- *mid* the entity id e.g. CHEBI:17158

addModelCreator(*firstname*, *lastname*, *organisation=None*, *email=None*)

Add a model creator to the list of model creators, only the first and fmaily names are mandatory:

- *firstname*
- *lastname*
- *organisation* [default=None]
- *email* [default=None]

addObjective(*obj*, *active=False*)
Add an instantiated Objective object to the FBA model

- *obj* an instance of the Objective class
- *active* [default=False] flag this objective as the active objective (fba.activeObjIdx)

addParameter(*par*)
Add an instantiated Parameter object to the model

- *par* an instance of the Parameter class

addReaction(*reaction*, *create_default_bounds=False*, *silent=False*)
Adds a reaction object to the model

- *reaction* an instance of the Reaction class
- *create_default_bounds* create default reaction bounds, irreversible 0 <= J <= INF, reversable -INF <= J <= INF
- *silent* [default=False] if enabled this disables the printing of information messages

addSpecies(*species*)
Add an instantiated Species object to the FBA model

- *species* an instance of the Species class

addUserConstraint(*pid*, *fluxes=None*, *operator='='*, *rhs=0.0*)
Add a user defined constraint to FBA model, this is additional to the automatically determined Stoichiometric constraints.

- *pid* user constraint name/id, use *None* for auto-assign
- *fluxes* a list of (coefficient, reaction id) pairs where coefficient is a float
- *operator* is one of = > < >= <=
- *rhs* a float

buildStoichMatrix(*matrix_type='numpy'*, *only_return=False*)
Build the stoichiometric matrix N and additional constraint matrix CN (if required)

- *matrix_type* [default='numpy'] the type of matrix to use to generate constraints
  - *numpy* a NumPy matrix default
  - *sympy* a SymPy symbolic matrix, if available note the denominator limit can be set in `CBModel.__CBCONFIG__['SYMPY_DENOM_LIMIT'] = 10**12`
  - *scipy_csr* create using NumPy but store as SciPy csr_sparse
- *only_return* [default=False] **IMPORTANT** only returns the stoichiometric matrix and constraint matrix (if required), does not update the model

changeAllFluxBoundsWithValue(*old*, *new*)
Replaces all flux bounds with value "old" with a new value "new":

- *old* value
- *new* value

clone()
Return a clone of this object.

createCompartment(*cid*, *name=None*, *size=1*, *dimensions=3*, *volume=None*)
Create a new compartment and add it to the model if the id does not exist

- *cid* compartment id
- *name* [None] compartment name
- *size* [1] compartment size
- *dimensions* [3] compartment size dimensions
- *volume* [None] compartment volume

createGeneAssociationsFromAnnotations(*annotation_key='GENE ASSOCIATION'*, *replace_existing=True*)
Add genes to the model using the definitions stored in the annotation key. If this fails it tries some standard annotation keys: GENE ASSOCIATION, GENE_ASSOCIATION, gene_association, gene association.

- *annotation_key* the annotation dictionary key that holds the gene association for the protein/enzyme
- *replace_existing* [default=True] replace existing annotations, otherwise only new ones are added

createGeneProteinAssociation(*protein*, *assoc*, *gid=None*, *name=None*, *gene_pattern=None*, *update_idx=True*, *altlabels=None*)
Create and add a gene protein relationship to the model, note genes are mapped on protein objects which may or may not be reactions

- *protein* in this case the reaction
- *assoc* the COBRA style gene protein association
- *gid* the unique id
- *name* the optional name
- *gene_pattern* deprecated, not needed anymore
- *update_idx* update the model gene index, not used
- *altlabels* [default=None] alternative labels for genes, default uses geneIds

createGroup(*gid*)
Create an empty group with

- *gid* the unique group id

createObjectiveFunction(*rid*, *coefficient=1*, *osense='maximize'*, *active=True*, *delete_current_obj=True*)
Create a single variable objective function:

- **rid** The
- **coefficient** [default=1]
- **osense** [default='maximize']
- **active** [default=True]
- **delete_current_obj** [default=True]

createReaction(*rid*, *name=None*, *reversible=True*, *create_default_bounds=True*, *silent=False*)
Create a new blank reaction and add it to the model:

- *id* the unique reaction ID
- *name* the reaction name
- *reversible* [default=True] the reaction reversibility. True is reversible, False is irreversible
- *create_default_bounds* create default reaction bounds, irreversible 0 <= J <= INF, reversable -INF <= J <= INF
- *silent* [default=False] if enabled this disables the printing of information messages

createReactionBounds(*reaction*, *lb_value*, *ub_value*)
Create a new lower bound for a reaction: value <= reaction

- **reaction** the reaction id
- **lb_value** the value of the lower bound
- **ub_value** the value of the upper bound

createReactionLowerBound(*reaction*, *value*)
Create a new lower bound for a reaction: value <= reaction

- **reaction** the reaction id
- **value** the value of the bound

createReactionReagent(*reaction*, *metabolite*, *coefficient*, *silent=False*)
Add a reagent to an existing reaction, both reaction and metabolites must exist

- *reaction* a reaction id
- *metabolite* a species/metabolite id
- *coefficient* the reagent coefficient

createReactionUpperBound(*reaction*, *value*)
Create a new upper bound for a reaction: reaction <= value

- **reaction** the reaction id
- **value** the value of the bound

createSingleGeneEffectMap()
This takes a model and analyses the logical gene expression patterns. This only needs to be done once, the result is a dictionary that has boolean effect patterns as keys and the (list of) genes that give rise to those patterns as values. This map is used by the single gene deletion method for further analysis.

Note this dictionary can also be stored and retrieved separately as long as the model structure is not changed i.e. the gene associations themselves or order of reactions (stored as the special entry 'keyJ').

Stored as self.__single_gene_effect_map__

createSpecies(*sid*, *boundary=False*, *name=''*, *value=nan*, *compartment=None*, *charge=None*, *chemFormula=None*)
Create a new species and add it to the model:

•**id** the unique species id

•**boundary** [default=False] whether the species is a variable (False) or is a boundary parameter (fixed)

•**name** [default='‘] the species name

•**value** [default=nan] the value *not currently used*

•**compartment** [default=None] the compartment the species is located in

•**charge** [default=None] the species charge

•**chemFormula** [default=None] the chemical formula

deleteAllFluxBoundsWithValue(*value*)
Delete all flux bounds which have a specified value:

•*value* the value of the flux bound(s) to delete

deleteBoundsForReactionId(*rid*, *lower=True*, *upper=True*)
Delete bounds connected to reaction, rid

•*rid* a valid reaction id

•*upper* [default=True] delete the upper bound

•*lower* [default=True] delete the lower bound

deleteGroup(*gid*)
Delete a group with

•*gid* the unique group id

deleteNonReactingSpecies(*simulate=True*)
Deletes all species that are not reagents (do not to take part in a reaction). *Warning* this deletion is permanent and greedy (not selective). Returns a list of (would be) deleted species

•*simulate* [default=True] only return a list of the speciesId's that would have been deleted if False

deleteObjective(*objective_id*)
Delete objective function:

objective_id the id of the objective function. If objective_id is given as 'active' then the active objective is deleted.

deleteReactionAndBounds(*rid*)
Delete all reaction and bounds connected to reaction

•*rid* a valid reaction id

deleteSpecies(*sid*, *also_delete=None*)
Deletes a species object with id

•*sid* the species id

•*also_delete* [default=None] only delete the species

– 'reagents' delete the species from the reactions it participates in as a **reagent** – 'reactions' deletes the **reactions** that the species participates in

emptyUndelete()
Empties the undelete cache

exportFVAdata()
Export the fva data as an array and list of reaction id's

findFluxesForConnectedSpecies(*metab*)
Returns a list of (reaction, flux value) pairs that this metabolite appears as a reagent of

- *metab* the metabolite name

getActiveObjective()
Returns the active objective object.

getActiveObjectiveReactionIds()
Returns the active objective flux objective reaction id's

getActiveObjectiveStoichiometry()
Returns a list of (coefficient, flux_objective) tuples

getAllFluxBounds()
Returns a dictionary of all flux bounds [id:value]

getAllGeneActivities()
Returns a dictionary of genes (if defined) and whether they are active or not

getAllGeneProteinAssociations(*use_labels=False*)
Returns a dictionary of genes associated with each protein

- *use_labels* use V2 gene labels rather than ID's

getAllProteinActivities()
Returns a dictionary of reactions (if genes and GPR's are defined) and whether they are active or not

getAllProteinGeneAssociations(*use_labels=False*)
Returns a dictionary of the proteins associated with each gene

- *use_labels* use V2 gene labels rather than ID's

getBoundarySpeciesIds(*rid=None*)
Return all boundary species associated with reaction

- rid [default=None] by default return all boundary species in a model, alternatively a string containing a reaction id or list of reaction id's

getCompartment(*cid*)
Returns a compartment object with *cid*

- *cid* compartment ID

getCompartmentIds(*substring=None*)
Returns a list of compartment Ids, applies a substring search if substring is defined

- *substring* search for this pattern anywhere in the id

getDescription()
Returns the model description which was stored in the SBML <notes> field

getExchangeReactionIds()
Returns id's of reactions where the 'is_exchange' attribute set to True. This is by default reactions that contain a boundary species.

getExchangeReactions()
Returns reaction instances where the 'is_exchange' attribute set to True. This is by default reactions that contain a boundary species.

getFluxBoundByID(*fid*)
Returns a FluxBound with id

•*fid* the fluxBound ID

getFluxBoundByReactionID(*rid*, *bound*)
Returns a FluxBound instance

•*rid* the reaction ID

•*bound* the bound: 'upper', 'lower', 'equality'

getFluxBoundIds(*substring=None*)
Returns a list of fluxbound Ids, applies a substring search if substring is defined

•*substring* search for this pattern anywhere in the id

getFluxBoundsByReactionID(*rid*)
Returns all FluxBound instances connected to a reactionId as a tuple of valid (lower, upper, None) or (None, None, equality) or alternatively invalid (lower, upper, equality).

•*rid* the reaction ID

under evaluation

getFluxesAssociatedWithSpecies(*metab*)
Returns a list of (reaction, flux value) pairs that this metabolite appears as a reagent in

•*metab* the metabolite name

getGPRassociation(*gpr_id*)
Returns a gene protein association object that has the identifier:

•*gpr_id* the gene protein identifier

getGPRforReaction(*rid*)
Return the GPR associated with the reaction id:

•*rid* a reaction id

getGene(*g_id*)
Returns a gene object that has the identifier:

•*gid* the gene identifier

getGeneByLabel(*label*)
Given a gene label return the corresponding Gene object

•*label*

getGeneIdFromLabel(*label*)
Given a gene label it returns the corresponding Gene id or None

•*label*

getGeneIds(*substring=None*)
Returns a list of gene Ids, applies a substring search if substring is defined

•*substring* search for this pattern anywhere in the id

getGeneLabels(*substring=None*)
Returns a list of gene labels (locus tags), applies a substring search if substring is defined

- *substring* search for this pattern anywhere in the id

getGroup(*gid*)
Return a group with

- *gid* the unique group id

getGroupIds()
Delete all group ids

getIrreversibleReactionIds()
Return a list of irreversible reaction Id's

getModelCreators()
Return model creator information

getObjFuncValue()
Returns the objective function value

getObjectiveIds(*substring=None*)
Returns a list of objective function Ids, applies a substring search if substring is defined

- *substring* search for this pattern anywhere in the id

getParameter(*pid*)
Returns a parameter object with pid

getReaction(*rid*)
Returns a reaction object with *id*

- *rid* reaction ID

getReactionActivity(*rid*)
If there is a GPR and genes associated with the reaction ID then return either active=True or inactive=False Note if there is no gene associated information then this will return active.

- *rid* a reaction id

getReactionBounds(*rid*)
Get the bounds of a reaction, returns a tuple of rid, lowerbound value, upperbound value and equality value (None means bound does not exist).

- *rid* the reaction ID

getReactionIds(*substring=None*)
Returns a list of reaction Ids, applies a substring search if substring is defined

- *substring* search for this pattern anywhere in the id

getReactionIdsAssociatedWithSpecies(*metab*)
Returns a list of (reaction, flux value) pairs that this metabolite appears as a reagent in

- *metab* the metabolite name

getReactionLowerBound(*rid*)
Returns the lower bound of a reaction (it it exists) or None

- *rid* the reaction ID

getReactionNames(*substring=None*)
Returns a list of reaction names, applies a substring search if substring is defined

•*substring* search for this pattern anywhere in the name

getReactionUpperBound(*rid*)
Returns the upper bound of a reaction (it it exists) or None

•*rid* the reaction ID

getReactionValues(*only_exchange=False*)
Returns a dictionary of ReactionID : ReactionValue pairs:

•*only_exchange* [default=False] only return the reactions labelled as exchange

getReversibleReactionIds()
Return a list of reversible reaction Id's

getSolutionVector(*names=False*)
Return a vector of solution values

•*names* [default=False] if True return a solution vector and list of names

getSpecies(*sid*)
Returns a species object with *sid*

•*sid* a specied ID

getSpeciesIds(*substring=None*, *non_boundary=False*)
Returns a list of species Ids, applies a substring search if substring is defined

•*substring* search for this pattern anywhere in the id

•*non_boundary* [default=False] return only non-boundary species, i.e., variable metabolites that appear in the stoichiometric matrix. The default is to return all metabolites boundary and variable.

renameObjectIds(*prefix=None*, *suffix=None*, *target='all'*, *ignore=None*)
This method is designed for target="all" other use may result in incomplete models.

•*prefix* [None] if supplied add as a prefix

•*suffix* [None] if supplied add as a suffix

•*target* ['all'] specify what class of objects to rename

•'species'

•'reactions'

•'bounds'

•'objectives'

•'all'

•*ignore* [default=None] a list of id's to ignore

resetAllGenes(*update_reactions=False*)
Resets all genes to their default activity state (normally on)

•*update_reactions* [default=False] update the associated reactions fluxbounds from the gene deletion bounds if they exist

resetAllInactiveGPRBounds()
Resets all reaction bounds modified by the `cmod.setAllInactiveGeneReactionBounds()` method to their previous values

serialize(*protocol=0*)
Serialize object, returns a string by default

•***protocol* [default=0] serialize to a string or binary if required,** see pickle module documentation for details

overloaded in CBModel

serializeToDisk(*filename*, *protocol=2*)
Serialize to disk using pickle protocol:

•*filename* the name of the output file

•***protocol* [default=2] serialize to a string or binary if required,** see pickle module documentation for details

overloaded in CBModel

setAllFluxBounds(*bounds*)
DEPRECATED! use setFluxBoundsFromDict()

Sets all the fluxbounds present in bounds

•*bounds* a dictionary of [fluxbound_id : value] pairs (not per reaction!!!)

setAllInactiveGPRBounds(*lower=0.0*, *upper=0.0*)
Set all reactions that are inactive (as determined by gene and gpr evaluation) to bounds:

•*lower* [default=0.0] the new lower bound

•*upper* [default=0.0] the new upper bound

setAllProteinActivities(*activites*, *lower=0.0*, *upper=0.0*)
Given a dictionary of activities [rid : boolean] pairs set all the corresponding reactions:

•*activities* a dictionary of [rid : boolean] pairs

•*lower* [default=0.0] the lower bound of the deactivated flux

•*upper* [default=0.0] the upper bound of the deactivated flux

setBoundValueByName(*rid*, *value*, *bound*)
Deprecated use setReactionBound

Set a reaction bound

- *rid* the reactions id
- *value* the new value
- *bound* this is either 'lower' or 'upper'

setCreatedDate(*date=None*)
Set the model created date tuple(year, month, day, hour, minute, second)

•*date* [default=None] default is now (automatic) otherwise (year, month, day, hour, minute, second) e.g. (2012, 09, 24, 13, 34, 00)

setDescription(*html*)
Sets the model description which translates into the SBML <notes> field.

- *html* any valid html or the empty string to clear ‘’

setFluxBoundsFromDict(*bounds*)
Sets all the fluxbounds present in bounds

- *bounds* a dictionary of [fluxbound_id : value] pairs (not per reaction!!!)

setGeneActive(*g_id*, *update_reactions=False*)
Effectively restores a gene by setting it’s active flag

- *g_id* a gene ID
- *update_reactions* [default=False] update the associated reactions fluxbounds from the gene deletion bounds if they exist

setGeneInactive(*g_id*, *update_reactions=False*, *lower=0.0*, *upper=0.0*)
Effectively deletes a gene by setting it’s inactive flag while optionally updating the GPR associated reactions

- *g_id* a gene ID
- *update_reactions* [default=False] update the associated reactions fluxbounds
- *lower* [default=0.0] the deactivated reaction lower bound
- *upper* [default=0.0] the deactivated reaction upper bound

setModifiedDate(*date=None*)
Set the model modification date: tuple(year, month, day, hour, minute, second)

- *date* [default=None] default is now (automatic) otherwise (year, month, day, hour, minute, second) e.g. (2012, 09, 24, 13, 34, 00)

setObjectiveFlux(*rid*, *coefficient=1*, *osense='maximize'*, *delete_objflx=True*)
Set single target reaction flux for the current active objective function.

- *rid* a string containing a reaction id
- *coefficient* [default=1] an objective flux coefficient
- *osense* the optimization sense must be **maximize** or **minimize**
- *delete_objflx* [default=True] delete all existing fluxObjectives in the active objective function

setPrefix(*prefix*, *target*)
This is alpha stuff, target can be:

- ‘species’
- ‘reactions’
- ‘constraints’
- ‘objectives’
- ‘all’

setReactionBound(*rid*, *value*, *bound*)
Set a reaction bound

- *rid* the reactions id
- *value* the new value
- *bound* this is either 'lower' or 'upper', or 'equal'

setReactionBounds(*rid*, *lower*, *upper*)
Set both the upper and lower bound of a reaction:

- *rid* the good old reaction id
- *lower* the lower bound value
- *upper* the upper bound value

setReactionLowerBound(*rid*, *value*)
Set a reactions lower bound (if it exists)

- *rid* the reactions id
- *value* the new value

setReactionUpperBound(*rid*, *value*)
Set a reactions upper bound (if it exists)

- *rid* the reaction id
- *value* the new value

setSuffix(*suffix*, *target*)
This is alpha stuff, target can be:

- 'species'
- 'reactions'
- 'constraints'
- 'objectives'
- 'all'

sortReactionsById()
Sorts the reactions by Reaction.id uses the python string sort

sortSpeciesById()
Sorts the reaction list by Reaction.id uses the python string sort

splitEqualityFluxBounds()
Splits any equalit flux bounds into lower and upper bounds.

testGeneProteinAssociations()
This method will test the GeneProtein associations and return a list of protein, association pairs

updateNetwork(*lower=0.0*, *upper=0.0*)
Update the reaction network based on gene activity. If reaction is deactivated then lower and upper bounds are used

- *lower* [default=0.0] deactivated lower bound
- *upper* [default=0.0] deactivated upper bound

class cbmpy.CBModel.**Objective**(*pid*, *operation*)
An objective function

addFluxObjective(*fobj*, *override=False*)
Adds a FluxObjective instance to the Objective

•*fobj* the FluxObjective object

•*override* [default=False] override pushing the global id map, this should never be used

createFluxObjectives(*fluxlist*)
Create and add flux objective objects to this objective function.

•*fluxlist* a list of one or more ('coefficient', 'rid') pairs

deleteAllFluxObjectives()
Delete all flux objectives

getFluxObjective(*foid*)
Return the flux objective with id.

•*foid* the flux objective id returns either an object or a list if there are multiply defined flux objectives

getFluxObjectiveData()
Returns a list of ObjectiveFunction components as (coefficient, flux) pairs

getFluxObjectiveForReaction(*rid*)
Returns the FluxObjective associated with the suplied rid. If there is more than fluxObjective associated with a reaction (illegal) then a list of fluxObjectives is returned.

rid a reaction id

getFluxObjectiveIDs()
Returns a list of ObjectiveFlux ids, for the reaction id's use *getFluxObjectiveReactions()* or for coefficient, fluxobjective pairs use *getFluxObjectiveData()*

getFluxObjectiveReactions()
Returns a list of reactions that are used as FluxObjectives

getFluxObjectives()
Returns the list of FluxObjective objects.

getOperation()
Returns the operation or sense of the objective

getValue()
Returns the current value of the attribute (input/solution)

setOperation(*operation*)
Sets the objective operation (sense)

•*operation* [default='maximize'] one of 'maximize', 'maximise', 'max', 'minimize', 'minimise', 'min'

setValue(*value*)
Sets the attribute ''value''

class cbmpy.CBModel.**Parameter**(*pid*, *value*, *name=None*, *constant=True*)
Holds parameter information

addAssociation(*assoc*)
Add an object ID to associate with this object

deleteAssociation(*assoc*)
Delete the object id associated with this object

getAssociations()
Return the Object ID's associated with this parameter

getValue()
Returns the current value of the attribute (input/solution)

setValue(*value*)
Sets the attribute ''value''

class cbmpy.CBModel.**Reaction**(*pid*, *name=None*, *reversible=True*)
Holds reaction information

addReagent(*reag*)
Adds an instantiated Reagent object to the reaction

changeId(*pid*)
Changes the Id of the reaction and updates associated FluxBounds

createReagent(*metabolite*, *coefficient*)
Create a new reagent and add it to the reaction:

•**metabolite** the metabolite name

•**coefficient** the

—negative coefficient is a substrate – positive coefficient is a product

Will fail if a species reference already exists

deactivateReaction(*lower=0.0*, *upper=0.0*)
Deactivates a reaction by setting its bounds to lower and upper. Restore with reactivateReaction()

•*lower* [default=0.0] bound

•*upper* [default=0.0] bound

deleteReagentWithSpeciesRef(*sid*)
Delete a reagent (or reagents) that refers to the species id:

•*sid* a species/metabolite id

getEquation(*reverse_symb='='*, *irreverse_symb='>'*, *use_names=False*)
Return a pretty printed string containing the reaction equation

•*reverse_symb* [default = '='] the symbol to use for reversible reactions

•*irreverse_symb* [default = '>'] the symbol to use for irreversible reactions

•*use_names* [defualt = False] use species names rather than id's

getFVAdata(*roundnum=None*, *silent=True*)
Returns the data generated by CBSolver.FluxVariabilityAnalysis() for this reaction as a tuple of (Flux, FVAmin, FVAmax, span) where span is abs(FVAmax - FVAmin). FVAmin or FVAmax is None this indicates no solution to that particular optimization (infeasible).

- *roundnum* [default=None] the integer number of roundoff decimals the default is no rounding
- *silent* [default=True] supress output to stdout

getLowerBound()
Get the value of the reactions lower bound

getProductIds(*use_names=False*)
Returns a list of the reaction products, species identifiers

- *use_names* [default = False] use species names rather than id's

getReagent(*rid*)
Return the one or more reagent objects which have *rid*:

- *rid* a reagent *rid*

getReagentObjIds()
Returns a list of the reagent id's. For the name of the reagents/metabolites use *<reaction>.getSpeciesIds()*

getReagentRefs()
Returns a list of the reagents/metabolites

getReagentWithSpeciesRef(*sid*)
Return the reagent object which refers to the *species* id. If there are multiple reagents that refer to the same species a list is returned.

- *sid* the species/metabolite id

getSpeciesIds()
Returns a list of the reagents/metabolites

getSpeciesObj()
Returns a list of the species objects that are reagents

getStoichiometry(*use_names=False*, *altout=False*)
Returns a list of (coefficient, species) pairs for this reaction

- *use_names* [default = False] use species names rather than id's
- *altout* [default = False] returns a dictionary [DEPRECATED]

getSubstrateIds(*use_names=False*)
Returns a list of the reaction substrates, species identifiers

- *use_names* [defualt = False] use species names rather than id's

getUpperBound()
Get the value of the reactions upper bound

getValue()
Returns the current value of the flux.

reactivateReaction()
Activates a reaction deactivated with deactivateReaction

setId(*fid*)
Sets the object Id

•*fid* a valid c variable style id string

Reimplements @FBase.setId()

setLowerBound(*value*)
Set the value of the reactions lower bound

•*value* a floating point value

setStoichCoefficient(*sid*, *value*)
Sets the stoichiometric coefficient of a reagent that refers to a metabolite. Note *negative coefficients* are *substrates* while *positive* ones are *products*. At this point zero coefficients are not allowed

•*sid* the species/metabolite id

•*value* a floating point value != 0

setUpperBound(*value*)
Set the value of the reactions upper bound

•*value* a floating point value

setValue(*value*)
Sets the attribute *value* in this case the flux.

class cbmpy.CBModel.**Reagent**(*reid*, *species_ref*, *coef*)

Has a reactive species id and stoichiometric coefficient:

- negative = substrate
- positive = product
- species_ref a reference to a species obj

getCoefficient()
Returns the reagent coefficient

getRole()
Returns the reagents role, "substrate", "product" or None

getSpecies()
Returns the metabolite/species that the reagent reference refers to

setCoefficient(*coef*)
Sets the reagent coefficient and role, negative coefficients are substrates and positive ones are products

•*coeff* the new coefficient

setSpecies(*spe*)
Sets the metabolite/species that the reagent reference refers to

class cbmpy.CBModel.**Species**(*pid*, *boundary=False*, *name=None*, *value=nan*, *compartment=None*, *charge=None*, *chemFormula=None*)
Holds species/metabolite information

getCharge()
Returns the species charge

getChemFormula()
Returns the species chemical formula

getReagentOf()
Returns a list of reaction id's that this metabolite occurs in

getValue()
Returns the current value of the attribute (input/solution)

isReagentOf()
Returns a dynamically generated list of reactions that this species occurs as a reagent

rename(*newid*, *overwrite=True*)
Changes the species id and updates all reagents in the model reactions. Note that existing species with id == newid will be overwritten/deleted.

- *newid* the new species id.
- *overwrite* [default=True] overwrite species objects (highly recommended)

setBoundary()
Sets the species so it is a boundary metabolite or fixed which does not occur in the stoichiometric matrix N

setCharge(*charge*)
Sets the species charge:

- *charge* a signed double but generally a signed int is used

setChemFormula(*cf*)
Sets the species chemical formula

- *cf* a chemical formula e.g. CH3NO2

setId(*fid*)
Sets the object Id

- *fid* a valid c variable style id string

Reimplements @FBase.setId()

setReagentOf(*rid*)
Adds the supplied reaction id to the reagent_of list (if it isn't one already)

- *rid* a valid reaction id

setValue(*value*)
Sets the attribute ''value''

unsetBoundary()
Unsets the species boundary attribute so that the metabolite is free and therefore occurs in the stoichiometric matrix N

3.8 CBMPy: CBModelTools module

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program. If not, see <http://www.gnu.org/licenses/>

Author: Brett G. Olivier Contact email: bgoli@users.sourceforge.net Last edit: $Author: bgoli $ ($Id: CBModelTools.py 575 2017-04-13 12:18:44Z bgoli $)

3.9 CBMPy: CBMultiCore module

PySCeS Constraint Based Modelling (http://cbmpy.sourceforge.net) Copyright (C) 2009-2017 Brett G. Olivier, VU University Amsterdam, Amsterdam, The Netherlands

This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program. If not, see <http://www.gnu.org/licenses/>

Author: Brett G. Olivier Contact email: bgoli@users.sourceforge.net Last edit: $Author: bgoli $ ($Id: CBMultiCore.py 575 2017-04-13 12:18:44Z bgoli $)

cbmpy.CBMultiCore.**grouper**(*3*, *'abcdefg'*, *'x'*) –> *('a', 'b', 'c'), ('d', 'e', 'f'), ('g', 'x', 'x')*

cbmpy.CBMultiCore.**runMultiCoreFVA**(*fba*, *selected_reactions=None*, *pre_opt=True*, *tol=None*, *objF2constr=True*, *rhs_sense='lower'*, *optPercentage=100.0*, *work_dir=None*, *quiet=True*, *debug=False*, *oldlpgen=False*, *markupmodel=True*, *procs=2*)

Run a multicore FVA where:

- *fba* is an fba model instance
- *procs* [default=2] number of processing threads (optimum seems to be about the number of physical cores)

3.10 CBMPy: CBMultiEnv module

PySCeS Constraint Based Modelling (http://cbmpy.sourceforge.net) Copyright (C) 2009-2017 Brett G. Olivier, VU University Amsterdam, Amsterdam, The Netherlands

This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program. If not, see <http://www.gnu.org/licenses/>

Author: Brett G. Olivier Contact email: bgoli@users.sourceforge.net Last edit: $Author: bgoli $ ($Id: CBMultiEnv.py 575 2017-04-13 12:18:44Z bgoli $)

3.11 CBMPy: CBNetDB module

PySCeS Constraint Based Modelling (http://cbmpy.sourceforge.net) Copyright (C) 2009-2017 Brett G. Olivier, VU University Amsterdam, Amsterdam, The Netherlands

This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program. If not, see <http://www.gnu.org/licenses/>

Author: Brett G. Olivier Contact email: bgoli@users.sourceforge.net Last edit: $Author: bgoli $ ($Id: CBNetDB.py 575 2017-04-13 12:18:44Z bgoli $)

class cbmpy.CBNetDB.**DBTools**

Tools to work with SQLite DB's (optimized, no SQL required).

checkEntryInColumn(*table*, *col*, *rid*)

Check if an entry exists in a table

- *table* the table name
- *col* the column name
- *rid* the row to search for

closeDB()

Close the DB connection and reset the DBTools instance (can be reconnected)

connectSQLiteDB(*db_name*, *work_dir=None*)

Connect to a sqlite database.

- *db_name* the name of the sqlite database
- *work_dir* the optional database path

createDBTable(*table*, *sqlcols*)

Create a database table if it does not exist:

- *table* the table name
- *sqlcols* a list containing the SQL definitions of the table columns: <id> <type> for example *['gene TEXT PRIMARY KEY', 'aa_seq TEXT', 'nuc_seq TEXT', 'aa_len INT', 'nuc_len INT']*

Effectively writes CREATE TABLE "table" (<id> <type>, gene TEXT PRIMARY KEY, aa_seq TEXT, nuc_seq TEXT, aa_len INT, nuc_len INT) % table

dumpTableToCSV (*table*, *filename*)
Save a table as tab separated txt file

- *table* the table to export
- *filename* the filename of the table dump

dumpTableToTxt (*table*, *filename*)
Save a table as tab separated txt file

- *table* the table to export
- *filename* the filename of the table dump

executeSQL (*sql*)
Execute a SQL command:

- *sql* a string containing a SQL command

fetchAll (*sql*)
Raw SQL query e.g. 'SELECT id FROM gene WHERE gene="G"'

getCell (*table*, *col*, *rid*, *cell*)
Get the table cell which correspond to rid in column. Returns the value or None

- *table* the database table
- *col* the column id
- *rid* the row index id
- *cell* the column of the cell you want tp extract

getColumns (*table*, *cols*)
Fetch the contents of one or more columns of data in a table

- *table* the database table
- *cols* a list of one or more column id's

getRow (*table*, *col*, *rid*)
Get the table row(s) which correspond to rid in column. Returns the row(s) as a list, if the column is the primary key this is always a single entry.

- *table* the database table
- *col* the column id
- *rid* the row index id

getTable (*table*, *colOut=False*)
Returns an entire database table

- *table* the table name
- *colOut* optionally return a tuple of (data,ColNames)

insertData (*table*, *data*, *commit=True*)

Insert data into a table: "INSERT INTO %s (?, ?, ?, ?, ?) VALUES (?, ?, ?, ?, ?)" % table,

(?, ?, ?, ?, ?)))

- *table* the DB table name
- *data* a dictionary of {id:value} pairs
- *commit* whether to commit the data insertions

updateData(*table*, *col*, *rid*, *data*, *commit=True*)
Update already defined data

•*table* the table name

•*col* the column name

•*rid* the row id to update

•*data* a dictionary of {id:value} pairs

•*commit* whether to commit the data updates

UPDATE COMPANY SET ADDRESS = 'Texas' WHERE ID = 6;

class cbmpy.CBNetDB.**KeGGSequenceTools**(*url*, *db_name*, *work_dir*)
Using the KeGG connector this class provides tools to construct an organims specific sequence database

class cbmpy.CBNetDB.**KeGGTools**(*url*)
Class that holds useful methods for querying KeGG via a SUDS provided soap client

fetchSeqfromKeGG(*k_gene*)
Given a gene name try and retrieve the gene and amino acid sequence

class cbmpy.CBNetDB.**MIRIAMTools**
Tools dealing with MIRIAM annotations

class cbmpy.CBNetDB.**RESTClient**
Class that provides the basis for application specific connectors to REST web services

Close()
Close the currently active connection

Connect(*root*)
Establish HTTP connection to

•*root* the site root "www.google.com"

Get(*query*)
Perform an http GET using:

•*query* e.g.

•*reply_mode* [default='] this is the reply mode

For example "/semanticSBML/annotate/search.xml?q=ATP"

GetLog()
Return the logged history

Log(*txt*)
Add txt to logfile history

•*txt* a string

class cbmpy.CBNetDB.**SemanticSBML**
REST client for connecting to SemanticSBML services

parseXMLtoText(*xml*)
Parse the xml output by quickLookup() into a list of URL

•*xml* XML returns from SemanticSBML

quickLookup(*txt*)
Do a quick lookpup for txt using SemanticSBML (connectic if required) and return results. Returns a list of identifiers.org id's in descending priority (as return)

•*txt* the string to lookup

viewDataInWebrowser(*maxres=10*)
Attempt to view #maxres results returned by SemanticSBML in the default browser

•*maxres* default maximum number of results to display.

3.12 CBMPy: CBPlot module

PySCeS Constraint Based Modelling (http://cbmpy.sourceforge.net) Copyright (C) 2009-2017 Brett G. Olivier, VU University Amsterdam, Amsterdam, The Netherlands

This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program. If not, see <http://www.gnu.org/licenses/>

Author: Brett G. Olivier Contact email: bgoli@users.sourceforge.net Last edit: $Author: bgoli $ ($Id: CBPlot.py 575 2017-04-13 12:18:44Z bgoli $)

cbmpy.CBPlot.**plotFluxVariability**(*fva_data*, *fva_names*, *fname*, *work_dir=None*, *title=None*, *ySlice=None*, *minHeight=None*, *maxHeight=None*, *roundec=None*, *autoclose=True*, *fluxval=True*, *type='png'*)
Plots and saves as an image the flux variability results as generated by CBSolver.FluxVariabilityAnalysis.

•*fva_data* FluxVariabilityAnalysis() FVA OUTPUT_ARRAY

•*fva_names* FluxVariabilityAnalysis() FVA OUTPUT_NAMES

•*fname* filename_base for the CSV output

•*work_dir* [default=None] if set the output directory for the csv files

•*title* [default=None] the user defined title for the graph

•*ySlice* [default=None] this sets an absolute (fixed) limit on the Y-axis (+- ySlice)

•*minHeight* [default=None] the minimum length that defined a span

- *maxHeight* [default=None] the maximum length a span can obtain, bar will be limited to maxHeight and coloured yellow
- *roundec* [default=None] an integer indicating at which decimal to round off output. Default is no rounding.
- *autoclose* [default=True] autoclose plot after save
- *fluxval* [default=True] plot the flux value
- *type* [default='png'] the output format, depends on matplotlib backend e.g. 'png', 'pdf', 'eps'

3.13 CBMPy: CBQt4 module

Constraint Based Modelling in Python (http://pysces.sourceforge.net/getNewReaction) Copyright (C) 2009-2017 Brett G. Olivier, VU University Amsterdam, Amsterdam, The Netherlands

This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program. If not, see <http://www.gnu.org/licenses/>

Author: Brett G. Olivier Contact email: bgoli@users.sourceforge.net Last edit: $Author: bgoli $ ($Id: CBQt4.py 575 2017-04-13 12:18:44Z bgoli $)

`cbmpy.CBQt4.`**`createReaction`**(*mod*)

Create a reaction using the graphical Reaction Creator

- *mod* a CBMPy model object

3.14 CBMPy: CBRead module

PySCeS Constraint Based Modelling (http://cbmpy.sourceforge.net) Copyright (C) 2009-2017 Brett G. Olivier, VU University Amsterdam, Amsterdam, The Netherlands

This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program. If not, see <http://www.gnu.org/licenses/>

Author: Brett G. Olivier Contact email: bgoli@users.sourceforge.net Last edit: $Author: bgoli $ ($Id: CBRead.py 575 2017-04-13 12:18:44Z bgoli $)

cbmpy.CBRead.**readCOBRASBML**(*fname*, *work_dir=None*, *return_sbml_model=False*, *delete_intermediate=False*, *fake_boundary_species_search=False*, *output_dir=None*, *skip_genes=False*, *scan_notes_gpr=True*)

Read in a COBRA format SBML Level 2 file with FBA annotation where and return either a CBM model object or a (cbm_mod, sbml_mod) pair if return_sbml_model=True

- *fname* is the filename
- *work_dir* is the working directory
- *delete_intermediate* [default=False] delete the intermediate SBML Level 3 FBC file
- *fake_boundary_species_search* [default=False] after looking for the boundary_condition of a species search for overloaded id's <id>_b
- *output_dir* [default=None] the directory to output the intermediate SBML L3 files (if generated) default to input directory
- *skip_genes* [default=False] do not load GPR data
- *scan_notes_gpr* [default=True] if the model is loaded and no genes are detected the scan the <notes> field for GPR associationa

cbmpy.CBRead.**readExcel97Model**(*xlname*, *write_sbml=True*, *sbml_level=3*, *return_dictionaries=False*)

Reads a model encoded as an Excel97 workbook and returns it as a CBMPy model object and SBML file. Note the workbook must be formatted exactly like those produced by cbm.writeModelToExcel97(). Note that reactions have to be defined in **both** the *reaction* and *network_react* sheets to be included in the model.

- *xlpath* the filename of the Excel workbook
- *return_model* [default=True] construct and return the CBMPy model
- *write_sbml* [default=True] write the SBML file to fname
- *return_dictionaries* [default=False] return the dictionaries constructed when reading the Excel file (in place of the model)
- *sbml_level* [default=3] write the SBML file as either SBML L2 FBA or SBML L3 FBC file.

cbmpy.CBRead.**readSBML2FBA**(*fname*, *work_dir=None*, *return_sbml_model=False*, *fake_boundary_species_search=False*, *scan_notes_gpr=True*)

Read in an SBML Level 2 file with FBA annotation where:

- *fname* is the filename
- *work_dir* is the working directory if None then only fname is used
- *return_sbml_model* [default=False] return a a (cbm_mod, sbml_mod) pair
- *fake_boundary_species_search* [default=False] after looking for the boundary_condition of a species search for overloaded id's <id>_b
- *scan_notes_gpr* [default=True] if the model is loaded and no genes are detected the scan the <notes> field for GPR associationa

cbmpy.CBRead.**readSBML3FBC**(*fname*, *work_dir=None*, *return_sbml_model=False*, *xoptions={'validate': False}*, *scan_notes_gpr=True*)

Read in an SBML Level 3 file with FBC annotation where and return either a CBM model object

- *fname* is the filename
- *work_dir* is the working directory
- *return_sbml_model* deprecated and ignored please update code
- *xoptions* special load options enable with option = True
  - *nogenes* do not load/process genes
  - *noannot* do not load/process any annotations
  - *validate* validate model and display errors and warnings before loading
  - *readcobra* read the cobra annotation
  - *read_model_string* [default=False] read the model from a string (instead of a filename) containing an SBML document
- *scan_notes_gpr* [default=True] if the model is loaded and no genes are__example_models__can the <notes> field for GPR associationa

cbmpy.CBRead.**readSK_FVA**(*filename*)

Read Stevens FVA results (opt.fva) file and return a list of dictionaries

cbmpy.CBRead.**readSK_vertex**(*fname*, *bigfile=True*, *fast_rational=False*, *nformat='%.14f'*, *compression=None*, *hdf5file=None*)

Reads in Stevens vertex analysis file:

- *fname* the input filename (.all file that results from Stevens pipeline)
- *bigfile* [default=True] this option is now always true and is left in for backwards compatability
- *fast_rational* [default=False] by default off and uses SymPy for rational–>float conversion, when on uses float decomposition with a slight (2th decimal) decrease in accuracy
- *nformat* [default='%.14f'] the number format used in output files
- *compression* [default=None] compression to be used in hdf5 files can be one of [None, 'lzf', 'gz?', 'szip']
- *hdf5file* [default=None] if None then generic filename '_vtx_.tmp.hdf5' is uses otherwise <hdf5file>.hdf5

and returns an hdf5 *filename* of the results with a single group named **data** which countains datasets

- vertices
- rays
- lin

where all vectors are in terms of the column space of N.

cbmpy.CBRead.**readSK_vertexOld**(*fname*, *bigfile=False*, *fast_rational=False*, *nformat='%.14f'*, *compresslevel=3*)

Reads in Stevens vertex analysis file and returns, even more optimized for large datasets than the original.

- a list of vertex vectors
- a list of ray vectors
- the basis of the lineality space as a list of vectors

all vectors in terms of the column space of N

3.15 CBMPy: CBReadtxt module

PySCeS Constraint Based Modelling (http://cbmpy.sourceforge.net) Copyright (C) 2009-2017 Brett G. Olivier, VU University Amsterdam, Amsterdam, The Netherlands

This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program. If not, see <http://www.gnu.org/licenses/>

Author: Brett G. Olivier Contact email: bgoli@users.sourceforge.net Last edit: $Author: bgoli $ ($Id: CBReadtxt.py 575 2017-04-13 12:18:44Z bgoli $)

cbmpy.CBReadtxt.**readCSV**(*model_file*, *bounds_file=None*, *biomass_flux=None*, *model_id='FBAModel'*, *reaction_prefix='R_'*, *has_header=False*)

This function loads a CSV file and translates it into a Python object:

```
- *model_file* the name of the CSV file that contains the model
- *bounds_file* the name of the CSV file that contains the flux bounds
- *biomass_flux* the name of the reaction that is the objective␣
↪function
- *reaction_prefix* [default='R _'] the prefix to add to input␣
↪reaction ID's
- *has_header* [default=False] if there is a header row in the csv␣
↪file
```

3.16 CBMPy: CBSolver module

PySCeS Constraint Based Modelling (http://cbmpy.sourceforge.net) Copyright (C) 2009-2017 Brett G. Olivier, VU University Amsterdam, Amsterdam, The Netherlands

This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program. If not, see <http://www.gnu.org/licenses/>

Author: Brett G. Olivier Contact email: bgoli@users.sourceforge.net Last edit: $Author: bgoli $ ($Id: CBSolver.py 575 2017-04-13 12:18:44Z bgoli $)

3.17 CBMPy: CBTools module

PySCeS Constraint Based Modelling (http://cbmpy.sourceforge.net) Copyright (C) 2009-2017 Brett G. Olivier, VU University Amsterdam, Amsterdam, The Netherlands

This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program. If not, see <http://www.gnu.org/licenses/>

Author: Brett G. Olivier Contact email: bgoli@users.sourceforge.net Last edit: $Author: bgoli $ ($Id: CBTools.py 575 2017-04-13 12:18:44Z bgoli $)

cbmpy.CBTools.**addFluxAsActiveObjective**(*f*, *reaction_id*, *osense*, *coefficient=1*)

Adds a flux as an active objective function

- *reaction_id* a string containing a reaction id
- *osense* objective sense must be **maximize** or **minimize**
- *coefficient* the objective funtion coefficient [default=1]

cbmpy.CBTools.**addGenesFromAnnotations**(*fba*, *annotation_key='GENE ASSOCIATION'*, *gene_pattern=None*)

THIS METHOD IS DERPRECATED PLEASE USE cmod.createGeneAssociationsFromAnnotations()

Add genes to the model using the definitions stored in the annotation key

- *fba* and fba object
- *annotation_key* the annotation dictionary key that holds the gene association for the protein/enzyme
- *gene_pattern* deprecated, not needed anymore

cbmpy.CBTools.**addSinkReaction**(*fbam*, *species*, *lb=0.0*, *ub=1000.0*)

Adds a sink reactions that consumes a model *species* so that X –>

- *fbam* an fba model object
- *species* a valid species name
- *lb* lower flux bound [default = 0.0]

- *ub* upper flux bound [default = 1000.0]

cbmpy.CBTools.**addSourceReaction**(*fbam*, *species*, *lb=0.0*, *ub=1000.0*)
Adds a source reactions that produces a model *species* so that –> X

- *fbam* an fba model object
- *species* a valid species name
- *lb* lower flux bound [default = 0.0]
- *ub* upper flux bound [default = 1000.0]

Note reversiblity is determined by the lower bound, default 0 = irreversible. If negative then reversible.

cbmpy.CBTools.**addStoichToFBAModel**(*fm*)
Build stoichiometry: this method has been refactored into the model class - cmod.buildStoichMatrix()

cbmpy.CBTools.**checkExchangeReactions**(*fba*, *autocorrect=True*)
Scan all reactions for exchange reactions (reactions containing a boundary species), return a list of inconsistent reactions or correct automatically.

- *fba* a CBMPy model
- *autocorrect* [default=True] correctly set the "is_exchange" attribute on a reaction

cbmpy.CBTools.**checkFluxBoundConsistency**(*fba*)
Check flux bound consistency checks for multiply defined bounds, bounds without a reaction, inconsistent bounds with respect to each other and reaction reversbility. Returns a dictionary of bounds/reactions where errors occur.

cbmpy.CBTools.**checkIds**(*fba*, *items='all'*)
Checks the id's of the specified model attributes to see if the name is legal and if there are duplicates. Returns a list of items with errors.

- *fba* a CBMPy model instance
- *items* [default='all'] 'all' means 'species,reactions,flux_bounds,objectives' of which one or more can be specified

cbmpy.CBTools.**checkProducibility**(*mod*, *metabolites=None*, *reactions=None*, *retOnlyZeroEntr=False*, *zeroLimit=1e-11*)
Check for blocked metabolites by adding a sink reaction and maximizing its output. If no metabolites are defined all metabolites are used by default. Returns a dictionary of metabolite id and sink flux pairs:

- *mod* a CBMPy model
- *metabolites* [default=[]] if not specified by default uses all metabolites defined in model
- *reactions* [default=[]] if defined, the reagents of each reaction listed here will be tested
- *retOnlyZeroEntr* [default=False] default returns all results, if this is try only blocked metabolites are returned
- *zeroLimit* [default=1.0e-11] values smaller than this are considered to be zero

This function was contributed by Willi Gottstein, Amsterdam, 2015.

cbmpy.CBTools.**checkProducibilityMetabolites**(*mod*, *metabolites=None*, *retOnlyZeroEntr=False*, *zeroLimit=1e-11*)

Check for blocked metabolites by adding a sink reaction and maximizing its output. If no metabolites are defined all metabolites are used by default. Returns a dictionary of metabolite id and sink flux pairs:

- *mod* a CBMPy model
- *metabolites* [default=[]] if not specified by default uses all metabolites defined in model
- *reactions* [default=[]] if defined, the reagents of each reaction listed here will be tested
- *retOnlyZeroEntr* [default=False] default returns all results, if this is try only blocked metabolites are returned
- *zeroLimit* [default=1.0e-11] values smaller than this are considered to be zero

This function was contributed by Willi Gottstein, Amsterdam, 2015.

cbmpy.CBTools.**checkProducibilityReactions**(*mod*, *reactions=None*, *retOnlyZeroEntr=False*, *zeroLimit=1e-11*)

Check for blocked metabolites by adding a sink reaction to each reaction reagent and maximizing its output. Returns a dictionary of reagent/metabolite id and sink flux pairs:

- *mod* a CBMPy model
- *reactions* [default=[]] if defined, the reagents of each reaction listed here will be tested
- *retOnlyZeroEntr* [default=False] default returns all results, if this is try only blocked metabolites are returned
- *zeroLimit* [default=1.0e-11] values smaller than this are considered to be zero

This function was contributed by Willi Gottstein, Amsterdam, 2015.

cbmpy.CBTools.**checkReactionBalanceElemental**(*f*, *Rid=None*, *zero_tol=1e-12*)

Check if the reaction is balanced using the chemical formula

- *f* the FBA object
- *Rid* [default = None] the reaction to check, defaults to all
- *zero_tol* [default=1.0e-12] the floating point zero used for elemental balancing

This function is derived from the code found here: http://pyparsing.wikispaces.com/file/view/chemicalFormulas.py

cbmpy.CBTools.**checkSuffixes**(*aList*, *suf1*, *suf2*)

Check whether there are strings in aList with the suffixes suf1 and suf2, respectively used in the function getReaByMetSuf

cbmpy.CBTools.**createTempFileName**()

Return a temporary filename

cbmpy.CBTools.**createZipArchive**(*zipname*, *files*, *move=False*, *compression='normal'*)

Create a zip archive which contains one or more files

- *zipname* the name of the zip archive to create (fully qualified)
- *files* either a valid filename or a list of filenames (fully qualified)
- *move* [default=False] attempt to delete input files after zip-archive creation

- *compression* [default='normal'] normal zip compression, set as None for no compression only store files (zlib not required)

cbmpy.CBTools.**deSerialize**(*s*)
Deserializes a serialised object contained in a string

cbmpy.CBTools.**deSerializeFromDisk**(*filename*)
Loads a serialised Python pickle from *filename* returns the Python object(s)

cbmpy.CBTools.**exportArray2CSV**(*arr*, *fname*)
Export an array to fname.csv

- *arr* the an array like object
- *fname* the output filename
- *sep* [default=','] the column separator

cbmpy.CBTools.**exportArray2TXT**(*arr*, *fname*)
Export an array to fname.txt

- *arr* the an array like object
- *fname* the output filename
- *sep* [default=','] the column separator

cbmpy.CBTools.**exportLabelledArray**(*arr*, *fname*, *names=None*, *sep=', '*, *fmt='%f'*)
Write a 2D array type object to file

- *arr* the an array like object
- *names* [default=None] the list of row names
- *fname* the output filename
- *sep* [default=','] the column separator
- *fmt* [default='%s'] the output number format

cbmpy.CBTools.**exportLabelledArray2CSV**(*arr*, *fname*, *names=None*)
Export an array with row names to fname.csv

- *arr* the an array like object
- *fname* the output filename
- *names* [default=None] the list of row names

cbmpy.CBTools.**exportLabelledArray2TXT**(*arr*, *fname*, *names=None*)
Export an array with row names to fname.txt

- *arr* the an array like object
- *names* [default=None] the list of row names
- *fname* the output filename

cbmpy.CBTools.**exportLabelledArrayWithHeader**(*arr*, *fname*, *names=None*, *header=None*, *sep=', '*, *fmt='%f'*)
Export an array with row names and header

- *arr* the an array like object

- *names* [default=None] the list of row names
- *header* [default=None] the list of column names
- *fname* the output filename
- *sep* [default=','] the column separator
- *fmt* [default='%s'] the output number format
- *appendlist* [default=False] if True append the array to *fname* otherwise create a new file

cbmpy.CBTools.**exportLabelledArrayWithHeader2CSV**(*arr*, *fname*, *names=None*, *header=None*)

Export an array with row names and header to fname.csv

- *arr* the an array like object
- *fname* the output filename
- *names* [default=None] the list of row names
- *header* [default=None] the list of column names

cbmpy.CBTools.**exportLabelledArrayWithHeader2TXT**(*arr*, *fname*, *names=None*, *header=None*)

Export an array with row names and header to fname.txt

- *arr* the an array like object
- *names* the list of row names
- *header* the list of column names
- *fname* the output filename

cbmpy.CBTools.**exportLabelledLinkedList**(*arr*, *fname*, *names=None*, *sep=','*, *fmt='%s'*, *appendlist=False*)

Write a 2D linked list [[...],[...],[...],[...]] and optionally a list of row labels to file:

- *arr* the linked list
- *fname* the output filename
- *names* [default=None] the list of row names
- *sep* [default=','] the column separator
- *fmt* [default='%s'] the output number format
- *appendlist* [default=False] if True append the array to *fname* otherwise create a new file

cbmpy.CBTools.**findDeadEndMetabolites**(*fbam*)

Finds dead-end (single reaction) metabolites rows in N with a single entry), returns a list of (metabolite, reaction) ids

cbmpy.CBTools.**findDeadEndReactions**(*fbam*)

Finds dead-end (single substrate/product) reactions (cols in N with a single entry), returns a list of (metabolite, reaction) ids

cbmpy.CBTools.**fixReversibility**(*fbam*, *auto_correct=False*)

Set fluxbound lower bound from reactions reversibility information.

- *fbam* and FBAModel instance
- *auto_correct* (default=False) if True automatically sets lower bound to zero if required, otherwise prints a warning if false.

cbmpy.CBTools.**getBoundsDict** (*fbamod*, *substring=None*)
Return a dictionary of reactions&bounds

cbmpy.CBTools.**getExchBoundsDict** (*fbamod*)
Return a dictionary of all exchange reactions (as determined by the is_exchange attribute of Reaction)

- *fbamod* a CBMPy model

cbmpy.CBTools.**getModelGenesPerReaction** (*fba*, *gene_pattern=None*, *gene_annotation_key='GENE ASSOCIATION'*)
Parse a BiGG style gene annotation string using default gene_pattern='((W*w*W*))' or (<any non-alphanum><any alphanum><any non-alphanum>)

Old eColi specific pattern '(bw*W)'

It is advisable to use the model methods directly rather than this function

cbmpy.CBTools.**getReaByMetSuf** (*fba_mod*, *suf1*, *suf2*, *retSpec=False*)

- can be used to determine all reactions in which at least two species with different suffixes are involved
- e.g. getReaByMetSuf(fba_mod, '_e', '_c') returns all reactions IDs between the extracellular compartment (suffix

'_e') and the cytosol (suffix '_c)'.

INPUT: fba_mod: a model instance suf1: suffix one (string) suf2: suffix two (string)

OUTPUT: if retSpec=True, a dictionary of reaction IDs and their associated species are returned if retSpec=False, a list with reaction IDs is returned

cbmpy.CBTools.**loadObj** (*filename*)
Loads a serialised Python pickle from *filename*.dat returns the Python object(s)

cbmpy.CBTools.**merge2Models** (*m1*, *m2*, *ignore=None*, *ignore_duplicate_ids=False*)
Merge 2 models, this method does a raw merge of model 2 into model 1 without any model checking. Component id's in ignore are ignored in both models and the first objective of model 1 is arbitrarily set as active. Compartments are also merged and a new "OuterMerge" compartment is also created.

In all cases duplicate id's are tracked and ignored, essentially using the object id encountered first - usually that of model 1. Duplicate checking can be disabled by setting the *ignore_duplicate_ids* flag.

- *m1* model 1
- *m2* model 2
- *ignore* [[]] do not merge these id's
- *ignore_duplicate_ids* [False] default behaviour that can be enabled

In development: merging genes and gpr's.

cbmpy.CBTools.**processBiGGannotationNote**(*fba*, *annotation_key='note'*)
Parse the HTML formatted reaction information stored in the BiGG notes field. This function is being deprecated and replaced by *CBTools.processSBMLAnnotationNotes()*

•requires an *annotation_key* which contains a BiGG HTML fragment

cbmpy.CBTools.**processBiGGchemFormula**(*fba*)
Disambiguates the overloaded BiGG name NAME_CHEMFORMULA into

•*species.name* NAME

•*species.chemFormula* CHEMFORMULA

cbmpy.CBTools.**processExchangeReactions**(*fba*, *key*)
Extract exchange reactions from model using *key* and return:

•a dictionary of all exchange reactions without *medium* reactions

•a dictionary of *medium* exchange reactions (negative lower bound)

cbmpy.CBTools.**processSBMLAnnotationNotes**(*fba*, *annotation_key='note'*)
Parse the HTML formatted reaction information stored in the SBML notes field currently processes BiGG and PySCeSCBM style annotations it looks for the the annotation indexed with the *annotation_key*

•*annotation_key* [default='note'] which contains a HTML/XHTML fragment in BiGG/PySCeSCBM format

cbmpy.CBTools.**removeFixedSpeciesReactions**(*f*)
This function is a hack that removes reactions which only have boundary species as reactants and products. These are typically gene associations encoded in the Manchester style and there is probably a better way of working around this problem ...

•*f* an instantiated fba model object

cbmpy.CBTools.**roundOffWithSense**(*val*, *osense='max'*, *tol=1e-08*)
Round of a value in a way that takes into consideration the sense of the operation that generated it

•*val* the value

•*osense* [default='max'] the sense

•*tol* [default=1e-8] the tolerance of the roundoff factor

cbmpy.CBTools.**scanForReactionDuplicates**(*f*, *ignore_coefficients=False*)
This method uses uses a brute force apprach to finding reactions with matching stoichiometry

cbmpy.CBTools.**scanForUnbalancedReactions**(*f*, *output='all'*)
Scan a model for unbalanced reactions, returns a tuple of dictionaries balanced and unbalanced:

•*f* an FBA model instance

•*output* [default='all'] can be one of ['all','charge','element']

•*charge* return all charge **un** balanced reactions

•*element* return all element **un** balanced reactions

cbmpy.CBTools.**setSpeciesPropertiesFromAnnotations**(*fbam*, *overwriteCharge=False*, *overwriteChemFormula=False*)

This will attempt to set the model Species properties from the annotation. With the default options it will only replace missing data. With ChemicalFormula this is easy to detect however charge may have an "unknown value" of 0. Setting the optional values to true will replace any existing value with any valid annotation.

- *overwriteChemFormula* [default=False]
- *overwriteCharge* [default=False]

cbmpy.CBTools.**splitReversibleReactions**(*fba*, *selected_reactions=None*)

Split a (set of) reactions into reversible reactions returns a copy of the original model

R1: A = B R1f: A -> B R1r: B -> A

- *fba* an instantiated CBMPy model object
- *selected_reactions* if a reversible reaction id is in here split it

cbmpy.CBTools.**splitSingleReversibleReaction**(*fba*, *rid*, *fwd_id=None*, *rev_id=None*)

Split a single reversible reaction into two irreversible reactions, returns the original reversible reaction and bounds while deleting them from model.

R1: A = B R1_fwd: A -> B R1_rev: B -> A

- *fba* an instantiated CBMPy model object
- *rid* a valid reaction id
- *fwd_id* [default=None] the new forward reaction id, defaults to rid_fwd
- *rev_id* [default=None] the new forward reaction id, defaults to rid_rev

cbmpy.CBTools.**storeObj**(*obj*, *filename*, *compress=False*)

Stores a Python *obj* as a serialised binary object in *filename*.dat

- *obj* a python object
- *filename* the base filename
- *compress* [False] use gzip compression not *implemented*

cbmpy.CBTools.**stringReplace**(*fbamod*, *old*, *new*, *target*)

This is alpha stuff, target can be:

- 'species'
- 'reactions'
- 'constraints'
- 'objectives'
- 'all'

3.18 CBMPy: CBWrite module

PySCeS Constraint Based Modelling (http://cbmpy.sourceforge.net) Copyright (C) 2009-2017 Brett G. Olivier, VU University Amsterdam, Amsterdam, The Netherlands

This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program. If not, see <http://www.gnu.org/licenses/>

Author: Brett G. Olivier Contact email: bgoli@users.sourceforge.net Last edit: $Author: bgoli $ ($Id: CBWrite.py 575 2017-04-13 12:18:44Z bgoli $)

cbmpy.CBWrite.**BuildHformatFluxBounds**(*fba*, *infinity_replace=None*, *use_rational=False*)

Build and return a csio that contains the flux bounds in H format

- *fba* a PySCeS-CBM FBA object
- *infinity_replace* [default=None] if defined this is the abs(value) of +-<infinity>

cbmpy.CBWrite.**BuildLPConstraints**(*fba*, *use_rational=False*)

Build and return a csio that contains constraint constructed from the StoichiometeryLP object

- *fba* an fba model object which has a stoichiometry
- *use_rational* write rational number output [default=False]

cbmpy.CBWrite.**BuildLPConstraintsMath**(*fba*, *use_rational=False*)

Build and return a csio that contains the constaints in LP format Strict refers to dS/dt => 0 and dS/dt <= 0

cbmpy.CBWrite.**BuildLPConstraintsRelaxed**(*fba*)

Build and return a csio that contains the constaints in LP format Relaxed refers to dS/dt >= 0

cbmpy.CBWrite.**BuildLPConstraintsStrict**(*fba*, *use_rational=False*)

Build and return a csio that contains the constaints in LP format Strict refers to dS/dt = 0

cbmpy.CBWrite.**BuildLPFluxBounds**(*fba*, *use_rational=False*)

Build and return a csio that contains the flux bounds in LP format

cbmpy.CBWrite.**BuildLPUserConstraints**(*fba*, *use_rational=False*)

Build and return a csio that contains constraint constructed from the StoichiometeryLP object

- *fba* an fba model object which has a stoichiometry
- *use_rational* write rational number output [default=False]

cbmpy.CBWrite.**WriteFVAdata**(*fva*, *names*, *fname*, *work_dir=None*, *roundec=None*, *scale_min=False*, *appendfile=False*, *info=None*)

INFO: this method will be deprecated please update your scripts to use "writeFVAdata()"

cbmpy.CBWrite.**WriteFVAtoCSV**(*id*, *fva*, *names*, *Dir=None*, *fbaObj=None*)

INFO: this method will be deprecated please update your scripts to use "writeFVAtoCSV()"

cbmpy.CBWrite.**WriteModelHFormatFBA**(*fba*, *work_dir=None*, *use_rational=False*, *fullLP=True*, *format='%s'*, *infinity_replace=None*)

INFO: this method will be deprecated please update your scripts to use "writeModelHFormatFBA2()"

cbmpy.CBWrite.**WriteModelHFormatFBA2**(*fba*, *fname=None*, *work_dir=None*, *use_rational=False*, *fullLP=True*, *format='%s'*, *infinity_replace=None*)

INFO: this method will be deprecated please update your scripts to use "writeModelHFormatFBA2()"

cbmpy.CBWrite.**WriteModelLP**(*fba*, *work_dir=None*, *fname=None*, *multisymb=' '*, *format='%s'*, *use_rational=False*, *constraint_mode=None*, *quiet=False*)

INFO: this method will be deprecated please update your scripts to use "writeModelLP()"

cbmpy.CBWrite.**WriteModelLPOld**(*fba*, *work_dir=None*, *multisymb=' '*, *lpt=True*, *constraint_mode='strict'*, *use_rational=False*, *format='%s'*)

INFO: this method will be deprecated please update your scripts to use "writeModelLPOld()"

cbmpy.CBWrite.**WriteModelRaw**(*fba*, *work_dir=None*)

INFO: this method will be deprecated please update your scripts to use "writeModelRaw()"

cbmpy.CBWrite.**convertExcelToFloat**(*num*)

Converts an Excel "number" to a float

- *num* a number

cbmpy.CBWrite.**convertFloatToExcel**(*num*, *roundoff*)

Converts a float to Excel compatible "number"

- *num* a number
- *roundoff* the number of roundoff digits for round()

cbmpy.CBWrite.**exportModel**(*fba*, *fname=None*, *fmt='lp'*, *work_dir=None*, *use_rational='both'*)

Export the FBA model in different formats:

- *fba* the FBA model
- *fname* [default=None] the exported filename if None then *fba.getId()* is used
- *fmt* [default='lp'] the export format can be one of: 'lp' (CPLEX), 'hformat' (Polyhedra), 'all' (both)
- *use_rational* [default='both'] if *all* or *hformat* is specified should hformat files be written using rational math or not. The default *both* is the legacy behaviour and writes both.

Note that 'hformat' ignores 'fname' and only uses fba.getId() this is a legacy behaviour

cbmpy.CBWrite.**generateBGID**(*num*, *prefix*)

Create a BGID generator, which is <prefix><num> where perfix is two letters num is padded to 6 figures

- *num* the starting number
- *prefix* the two letter prefix

cbmpy.CBWrite.**printFBASolution**(*fba*, *include_all=False*)

Prints the FBA optimal solution to the screen.

- *fba* an FBA model object
- *include_all* include all variables

cbmpy.CBWrite.**writeCOBRASBML**(*fba*, *fname*, *directory=None*)

Takes an FBA model object and writes it to file as a COBRA compatible :

- *fba* an fba model object
- *fname* the model will be written as XML to *fname*
- *directory* [default=None] if defined it is prepended to fname

cbmpy.CBWrite.**writeFVAdata**(*fvadata*, *names*, *fname*, *work_dir=None*, *roundec=None*, *scale_min=False*, *appendfile=False*, *info=None*)

Takes the resuls of a FluxVariabilityAnalysis method and writes it to a nice csv file. Note this method replaces the glpk/cplx_WriteFVAtoCSV methods. Data is output as a csv file with columns: FluxName, FVA_MIN, FVA_MAX, OPT_VAL, SPAN

- *fvadata* FluxVariabilityAnalysis() FVA OUTPUT_ARRAY
- *names* FluxVariabilityAnalysis() FVA OUTPUT_NAMES
- *fname* filename_base for the CSV output
- *work_dir* [default=None] if set the output directory for the csv files
- *roundec* [default=None] an integer indicating at which decimal to round off output. Default is no rounding.
- *scale_min* [default=False] normalise each flux such that that FVA_MIN = 0.0
- *appendfile* [default=False] instead of opening a new file try and append the data
- *info* [default=None] a string added to the results as an extra column, useful with *appendfile*

cbmpy.CBWrite.**writeFVAtoCSV**(*fvadata*, *names*, *fname*, *Dir=None*, *fbaObj=None*)

Takes the resuls of a FluxVariabilityAnalysis method and writes it to a nice csv file. Note this method replaces the glpk/cplx_WriteFVAtoCSV methods.

- *fvadata* FluxVariabilityAnalysis() OUTPUT_ARRAY
- *names* FluxVariabilityAnalysis() OUTPUT_NAMES
- *fname* filename_base for the CSV output
- *Dir* [default=None] if set the output directory for the csv files
- *fbaObj* [default=None] if supplied adds extra model information into the output tables

cbmpy.CBWrite.**writeMinDistanceLPwithCost**(*fname*, *fbas*, *work_dir=None*, *ignoreDistance=[]*, *constraint_mode='strict'*)

For backwards compatability only

cbmpy.CBWrite.**writeModelHFormatFBA**(*fba*, *work_dir=None*, *use_rational=False*, *fullLP=True*, *format='%s'*, *infinity_replace=None*)

Write an FBA-LP in polynomial H-Format file. This version has been replaced by *writeModelHFormatFBA2()* but is kept for backwards compatability.

- *fba* a PySCeS-CBM FBA object
- *Work_dir* [default=None] the output directory
- *use_rational* [default=false] use rational numbers in output (requires sympy)
- *fullLP* [default=True] include the default objective function as a maximization target

•*format* [default='%s'] the number format string

•*infinity_replace* [default=None] if defined this is the abs(value) of +-<infinity>

cbmpy.CBWrite.**writeModelHFormatFBA2**(*fba*, *fname=None*, *work_dir=None*, *use_rational=False*, *fullLP=True*, *format='%s'*, *infinity_replace=None*)

Write an FBA-LP in polynomial H-Format file. This is an improved version of *WriteModelHFormatFBA()* which it replaces. Note that if a SymPy matrix is used as input then use_rational is automatically enabled.

•*fba* a PySCeS-CBM FBA object

•*fname* [default=None] the output filename, fba.getId() if not defined

•*Work_dir* [default=None] the output directory

•*use_rational* [default=false] use rational numbers in output (requires sympy)

•*fullLP* [default=True] include the default objective function as a maximization target

•*format* [default='%s'] the number format string

•*infinity_replace* [default=None] if defined this is the abs(value) of +-<infinity>

cbmpy.CBWrite.**writeModelInfoToFile**(*fba*, *fname*, *Dir=None*, *separator=','*, *only_exchange=False*, *met_type='all'*)

This function writes a CBModel to file

•*fba* an instance of an PySCeSCBM model

•*fname* the output filename

•*Dir* [default=None] use directory if not None

•*separator* [default=','] the column separator

•*only_exchange* [default=False] only output fluxes labelled as exchange reactions

•*type* [default='all'] only output certain type of species: 'all','boundary' or 'variable'

cbmpy.CBWrite.**writeModelLP**(*fba*, *work_dir=None*, *fname=None*, *multisymb=' '*, *format='%s'*, *use_rational=False*, *constraint_mode=None*, *quiet=False*)

Writes an FBA object as an LP in CPLEX LP format

•*fba* an instantiated FBAmodel instance

•*work_dir* directory designated for output

•*fname* the file name [default=fba.getId()]

•*multisymb* the multiplication symbol (default: <space>)

•*format* the number format of the output

•*use_rational* output rational numbers [default=False]

•*quiet* [default=False] supress information messages

cbmpy.CBWrite.**writeModelLPOld**(*fba*, *work_dir=None*, *multisymb=' '*, *lpt=True*, *constraint_mode='strict'*, *use_rational=False*, *format='%s'*)

Writes a fba as an LP/LPT

- *fba* an instantiated FBAmodel instance
- *work_dir* directory designated for output
- *multisymb* the multiplication symbol (default: <space>)
- *lpt* the file format (default: True for lpt) or False for lp

cbmpy.CBWrite.**writeModelRaw**(*fba*, *work_dir=None*)

Writes a fba (actually just dumps it) to a text file.

- *fba* an instantiated FBAmodel instance
- *work_dir* directory designated for output

cbmpy.CBWrite.**writeModelToCOMBINEarchive**(*mod*, *fname=None*, *directory=None*, *sbmlname=None*, *withExcel=True*, *vc_given='CBMPy'*, *vc_family='Software'*, *vc_email='None'*, *vc_org='cbmpy.sourceforge.net'*, *add_cbmpy_annot=True*, *add_cobra_annot=True*)

Write a model in SBML and Excel format to a COMBINE archive using the following information:

- *mod* a model object
- *fname* the output base filename, archive will be <fname>.zip
- *directory* [default=None] created the combine archive 'directory'
- *sbmlname* [default='None'] If *sbmlname* is defined then SBML file is <sbmlname>.xml otherwise sbml will be <fname>.xml.
- *withExcel* [default=True] include a human readable Excel spreadsheet version of the model
- *vc_given* [default='CBMPy'] first name
- *vc_family* [default='Software'] family name
- *vc_email* [default='None'] email
- *vc_org* [default='None'] organisation
- *add_cbmpy_annot* [default=True] add CBMPy KeyValueData annotation. Replaces <notes>
- *add_cobra_annot* [default=True] add COBRA <notes> annotation

cbmpy.CBWrite.**writeModelToExcel97**(*fba*, *filename*, *roundoff=6*)

Exports the model as an Excel 97 spreadsheet

- *fba* a CBMPy model instance
- *filename* the filename of the workbook
- *roundoff* [default=6] the number of digits to round off to

cbmpy.CBWrite.**writeOptimalSolution**(*fba*, *fname*, *Dir=None*, *separator=','*, *only_exchange=False*)

This function writes the optimal solution to file

- *fba* an instance of an PySCeSCBM model

- *fname* the output filename
- *Dir* [default=None] use current directory if not None
- *separator* [default=','] the column separator
- *only_exchange* [default=False] only output fluxes labelled as exchange reactions

cbmpy.CBWrite.**writeProteinCostToCSV**(*fba*, *fname*)

Writes the protein costs 'CBM_PEPTIDE_COST' annotation toa csv file.

- *fba* an instantiated FBA object
- *fname* the exported file name

cbmpy.CBWrite.**writeReactionInfoToFile**(*fba*, *fname*, *Dir=None*, *separator=', '*, *only_exchange=False*)

This function writes a CBModel to file

- *fba* an instance of an PySCeSCBM model
- *fname* the output filename
- *Dir* [default=None] use directory if not None
- *separator* [default=','] the column separator
- *only_exchange* [default=False] only output fluxes labelled as exchange reactions

cbmpy.CBWrite.**writeSBML2FBA**(*fba*, *fname*, *directory=None*, *sbml_level_version=None*)

Takes an FBA model object and writes it to file as SBML L2 with FBA annotations. Note if you want to write BiGG/FAME style annotations then you must use *sbml_level_version=(2,1)*

- *fba* an fba model object
- *fname* the model will be written as XML to *fname*
- *sbml_level_version* [default=None] a tuple containing the SBML level and version e.g. (2,1)

This is a utility wrapper for the function *CBXML.sbml_writeSBML2FBA*

cbmpy.CBWrite.**writeSBML3FBC**(*fba*, *fname*, *directory=None*, *gpr_from_annot=False*, *add_groups=True*, *add_cbmpy_annot=True*, *add_cobra_annot=False*, *xoptions={'fbc_version': 1, 'validate': False, 'compress_bounds': True}*)

Takes an FBA model object and writes it to file as SBML L3 FBC:

- *fba* an fba model object
- *fname* the model will be written as XML to *fname*
- *directory* [default=None] if defined it is prepended to fname
- *gpr_from_annot* [default=True] if enabled will attempt to add the gene protein associations from the annotations if no gene protein association objects exist
- *add_groups* [default=True] add SBML3 groups (if supported by libSBML)
- *add_cbmpy_annot* [default=True] add CBMPy KeyValueData annotation. Replaces <notes>
- *add_cobra_annot* [default=True] add COBRA <notes> annotation
- *xoptions* extended options

–*fbc_version* [default=1] write SBML3FBC using version 1 (2013) or version 2 (2015)

–*validate* [default=False] validate the output SBML file

–*compress_bounds* [default=False] try compress output flux bound parameters

–*zip_model* [default=False] compress the model using PKZIP encoding

–*return_model_string* [default=False] return the SBML XML file as a string

cbmpy.CBWrite.**writeSBML3FBCV2**(*fba*, *fname*, *directory=None*, *gpr_from_annot=False*, *add_groups=True*, *add_cbmpy_annot=True*, *add_cobra_annot=False*, *validate=False*, *compress_bounds=True*, *zip_model=False*, *return_model_string=False*)

Takes an FBA model object and writes it to file as SBML L3 FBCv2 :

•*fba* an fba model object

•*fname* the model will be written as XML to *fname*

•*directory* [default=None] if defined it is prepended to fname

•*gpr_from_annot* [default=False] if enabled will attempt to add the gene protein associations from the annotations

•*add_groups* [default=True] add SBML3 groups (if supported by libSBML)

•*add_cbmpy_annot* [default=True] add CBMPy KeyValueData annotation. Replaces <notes>

•*add_cobra_annot* [default=False] add COBRA <notes> annotation

•*validate* [default=False] validate the output SBML file

•*compress_bounds* [default=True] try compress output flux bound parameters

•*zip_model* [default=False] compress the model using ZIP encoding

•*return_model_string* [default=False] return the SBML XML file as a string

cbmpy.CBWrite.**writeSensitivitiesToCSV**(*sensitivities*, *fname*)

Write out a sensitivity report using the objective sensitivities and bound sensitivity dictionaries created by e.g. cplx_getSensitivities().

•*sensitivity* tuple containing

•*obj_sens* dictionary of objective coefficient sensitivities (per flux)

•*rhs_sens* dictionary of constraint rhs sensitivities (per constraint)

•*bound_sens* dictionary of bound sensitivities (per flux)

•*fname* output filename e.g. fname.csv

cbmpy.CBWrite.**writeSolutions**(*fname*, *sols=[]*, *sep=','*, *extra_output=None*, *fba=None*)

Write 2 or more solutions where a solution is a dictionary of flux:value pairs:

•*fname* the export filename

•*sols* a list of dictionaries containing flux:value pairs (e.g. output by cmod.getReactionValues())

- *sep* [default=’,’] the column separator
- *extra_output* [default=None] add detailed information to output e.g. reaction names by giving a CBModel object as an argument to *extra_output*.
- *fba* an fba model that canbe used for extra_output

cbmpy.CBWrite.**writeSpeciesInfoToFile**(*fba*, *fname*, *Dir=None*, *separator=’, ‘*, *met_type=’all’*)

This function writes a CBModel to file

- *fba* an instance of an PySCeSCBM model
- *fname* the output filename
- *Dir* [default=None] use directory if not None
- *separator* [default=’,’] the column separator
- *met_type* [default=’all’] only output certain type of species: ‘all’,’boundary’ or ‘variable’

cbmpy.CBWrite.**writeStoichiometricMatrix**(*fba*, *fname=None*, *work_dir=None*, *use_rational=False*, *fullLP=True*, *format=’%s’*, *infinity_replace=None*)

Write an FBA-LP in polynomial H-Format file. This is an improved version of *WriteModelHFormatFBA()* which it replaces but is kept for backwards compatability.

- *fba* a PySCeS-CBM FBA object
- *fname* [default=None] the output filename, fba.getId() if not defined
- *Work_dir* [default=None] the output directory
- *use_rational* [default=false] use rational numbers in output (requires sympy)
- *fullLP* [default=True] include the default objective function as a maximization target
- *format* [default=’%s’] the number format string
- *infinity_replace* [default=None] if defined this is the abs(value) of +-<infinity>

3.19 CBMPy: CBWx module

PySCeS Constraint Based Modelling (http://cbmpy.sourceforge.net) Copyright (C) 2009-2017 Brett G. Olivier, VU University Amsterdam, Amsterdam, The Netherlands

This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program. If not, see <http://www.gnu.org/licenses/>

Author: Brett G. Olivier Contact email: bgoli@users.sourceforge.net Last edit: $Author: bgoli $ ($Id: CBWx.py 575 2017-04-13 12:18:44Z bgoli $)

class cbmpy.CBWx.**HtmlWindowMod**(**args*, ***kwargs*)
Overrides 'OnLinkClicked' to open links in external browser

cbmpy.CBWx.**circlePoints**(*totalPoints=4*, *startAngle=0*, *arc=360*, *circleradius=1*, *centerxy=(0, 0)*, *direction='forward'*, *evenDistribution=True*)
Returns a list of points evenly spread around a circle:

- *totalPoints* how many points
- *startAngle* where to start
- *arc* how far to go
- *circleradius* radius
- *centerxy* origin
- *direction* 'forward' or 'backward'
- *evenDistribution* True/False

This code has been adapted from the Flash example that can be found here: http://www.lextalkington.com/blog/2009/12/generate-points-around-a-circles-circumference/

3.20 CBMPy: CBXML module

PySCeS Constraint Based Modelling (http://cbmpy.sourceforge.net) Copyright (C) 2009-2017 Brett G. Olivier, VU University Amsterdam, Amsterdam, The Netherlands

This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program. If not, see <http://www.gnu.org/licenses/>

Author: Brett G. Olivier Contact email: bgoli@users.sourceforge.net Last edit: $Author: bgoli $ ($Id: CBXML.py 575 2017-04-13 12:18:44Z bgoli $)

class cbmpy.CBXML.**MLStripper**
Class for stripping a string of HTML/XML used from: http://stackoverflow.com/questions/753052/strip-html-from-strings-in-python

cbmpy.CBXML.**SBML_NS = [('http://www.sbml.org/sbml/level3/version1/fbc/version2', 'L3V1FBC2'), ('http://w**
print libsbml.BQB_ENCODES , 8 # "encodes", print libsbml.BQB_HAS_PART , 1 # "hasPart", print libsbml.BQB_HAS_PROPERTY , 10 # "hasProperty", print libsbml.BQB_HAS_VERSION , 4 # "hasVersion", print libsbml.BQB_IS , 0 # "isA", print libsbml.BQB_IS_DESCRIBED_BY , 6 # "isDescribedBy", print libsbml.BQB_IS_ENCODED_BY , 7 # "isEncodedBy", print libsbml.BQB_IS_HOMOLOG_TO , 5 # "isHomologTo", print libsbml.BQB_IS_PART_OF , 2 # "isPartOf", print libsbml.BQB_IS_PROPERTY_OF , 11 # "isPropertyOf", print libsbml.BQB_IS_VERSION_OF , 3 # "isVersionOf", print libsbml.BQB_OCCURS_IN , 9 # "occursIn", print libsbml.BQB_UNKNOWN , 12 # None

print libsbml.BQM_IS , 0 # None print libsbml.BQM_IS_DERIVED_FROM , 2 # None print libsbml.BQM_IS_DESCRIBED_BY , 1 # None print libsbml.BQM_UNKNOWN , 3 # None

cbmpy.CBXML.**sbml_convertCOBRASBMLtoFBC**(*fname*, *outname=None*, *work_dir=None*, *output_dir=None*)

Read in a COBRA SBML Level 2 file and return the name of the created SBML Level 3 with FBC file that is created in the output directory

- *fname* is the filename
- *outname* the name of the output file. If not specified then <filename>.l3fbc.xml is used as default
- *work_dir* [default=None] is the working directory
- *output_dir* [default=None] is the output directory (default is work_dir)

This method is based on code from libSBML (http://sbml.org) in the file "convertCobra.py" written by Frank T. Bergmann.

cbmpy.CBXML.**sbml_convertSBML3FBCToCOBRA**(*fname*, *outname=None*, *work_dir=None*, *output_dir=None*)

Read in a SBML Level 3 file and return the name of the created COBRA file that is created in the output directory

- *fname* is the filename
- *outname* the name of the output file. If not specified then <filename>.cobra.xml is used as default
- *work_dir* [default=None] is the working directory
- *output_dir* [default=None] is the output directory (default is work_dir)

This method is based on code from libSBML (http://sbml.org) in the file "convertFbcToCobra.py" written by Frank T. Bergmann.

cbmpy.CBXML.**sbml_createAssociationFromAST**(*node*, *out*)

Converts a GPR string '((g1 and g2) or g3)' to an association via a Python AST. In future I will get rid of all the string elements and work only with associations and AST's.

- *node* a Python AST note (e.g. body)
- *out* a new shiny FBC V2 GeneProductAssociation

cbmpy.CBXML.**sbml_createModelL2**(*fba*, *level=2*, *version=1*)

Create an SBML model and document:

- *fba* a PySCeSCBM model instance
- *level* always 2
- *version* always 1

and returns:

- *model* an SBML model

cbmpy.CBXML.**sbml_exportSBML2FBAModel**(*document*, *filename*, *directory=None*, *return_doc=False*, *remove_note_body=False*)

Writes an SBML model object to file. Note this is an internal SBML method use *sbml_writeSBML2FBA()* to write an FBA model:

•*model* a libSBML model instance

•*filename* the output filename

•*directory* [default=None] by default use filename otherwise join, <dir><filename>

•*return_doc* [default=False] return the SBML document used to write the XML

cbmpy.CBXML.**sbml_fileFindVersion**(*f*)
Try and find the SBML version and FBC support

•*f* the SBML file

cbmpy.CBXML.**sbml_fileValidate**(*f*, *level='normal'*)
Validate an SBML file and model

•*f* the SBML file

•*level* [default='normal'] the level of validation "normal" or "full"

cbmpy.CBXML.**sbml_getCVterms**(*sb*, *model=False*)
Get the MIRIAM compliant CV terms and return a MIRIAMAnnotation or None

•*sb* a libSBML SBase derived object

•*model* is this a BQmodel term

cbmpy.CBXML.**sbml_getGeneRefs**(*association*, *out*)
Walk through a gene association and extract GeneRefs inspired by Frank

cbmpy.CBXML.**sbml_getNotes**(*obj*)
Returns the SBML objects notes

•*obj* an SBML object

cbmpy.CBXML.**sbml_readCOBRANote**(*s*)
Parses a COBRA style note from a XML string

•*s* an XML string

cbmpy.CBXML.**sbml_readCOBRASBML**(*fname*, *work_dir=None*, *return_sbml_model=False*, *delete_intermediate=False*, *fake_boundary_species_search=False*, *output_dir=None*, *speciesAnnotationFix=True*, *skip_genes=False*)
Read in a COBRA format SBML Level 2 file with FBA annotation where and return either a CBM model object or a (cbm_mod, sbml_mod) pair if return_sbml_model=True

•*fname* is the filename

•*work_dir* is the working directory

•*return_sbml_model* [default=False] return a a (cbm_mod, sbml_mod) pair

•*delete_intermediate* [default=False] delete the intermediate SBML Level 3 FBC file

•*fake_boundary_species_search* [default=False] after looking for the boundary_condition of a species search for overloaded id's <id>_b

•*output_dir* [default=None] the directory to output the intermediate SBML L3 files (if generated) default to input directory

- *speciesAnnotationFix* [default=True]
- *skip_genes* [default=False] convert GPR associations

cbmpy.CBXML.**sbml_readKeyValueDataAnnotation**(*annotations*)
Reads KeyValueData annotation (http://pysces.sourceforge.net/KeyValueData) and returns a dictionary of key:value pairs

cbmpy.CBXML.**sbml_readSBML2FBA**(*fname*, *work_dir=None*, *return_sbml_model=False*, *fake_boundary_species_search=False*)
Read in an SBML Level 2 file with FBA annotation where and return either a CBM model object or a (cbm_mod, sbml_mod) pair if return_sbml_model=True

- *fname* is the filename
- *work_dir* is the working directory (only used if not None)
- *return_sbml_model* [default=False] return a a (cbm_mod, sbml_mod) pair
- *fake_boundary_species_search* [default=False] after looking for the boundary_condition of a species search for overloaded id's <id>_b

cbmpy.CBXML.**sbml_readSBML3FBC**(*fname*, *work_dir=None*, *return_sbml_model=False*, *xoptions={}*)
Read in an SBML Level 3 file with FBC annotation where and return either a CBM model object or a (cbm_mod, sbml_mod) pair if return_sbml_model=True

- *fname* is the filename
- *work_dir* is the working directory
- *return_sbml_model* [default=False] return a a (cbm_mod, sbml_mod) pair
- *xoptions* special load options enable with option = True - *nogenes* do not load/process genes - *noannot* do not load/process any annotations - *validate* validate model and display errors and warnings before loading - *readcobra* read the cobra annotation - *read_model_string* [default=False] read the model from a string (instead of a filename) containing an SBML document

cbmpy.CBXML.**sbml_setAnnotationsL3Fbc**(*cbmo*, *sbmlo*)
Add CBMPy Fbase annotations to an SBML object, MIRIAM, SBO, Notes. Should be called last when creating SBML objects.

- *cbmo* the CBMPy object
- *sbmlo* SBML object

Note: this function should be used for new code, old code still needs to be refactored.

cbmpy.CBXML.**sbml_setCVterms**(*sb*, *uridict*, *model=False*)
Add MIRIAM compliant CV terms to a sbml object from a CBM object

- *sb* a libSBML SBase derived object
- *uridict* a dictionary of uri's as produced by getAllMIRIAMUris()
- *model* is this a BQmodel term [deprecated attribute, ignored and autodetected]

cbmpy.CBXML.**sbml_setCompartmentsL3**(*model*, *fba*)
Sets the model compartments.

- *model* a libSBML model instance

- *fba* a PySCeSCBM model instance

cbmpy.CBXML.**sbml_setDescription**(*model*, *fba*)
Sets the model description as a <note> containing *txt* in an HTML paragraph on the model object.

- *model* a libSBML model instance
- *fba* a PySCeSCBM model instance

cbmpy.CBXML.**sbml_setGroupsL3**(*cs*, *fba*)
add groups to the SBML model

- *cs* a CBMLtoSBML instance
- *fba* a CBMPy model instance

cbmpy.CBXML.**sbml_setNotes3**(*obj*, *s*)
Formats the CBMPy notes as an SBML note and adds it to the SBMl object

- *obj* an SBML object
- *s* a string that should be added as a note

cbmpy.CBXML.**sbml_setParametersL3Fbc**(*fbcmod*, *add_cbmpy_anno=True*)
Add non fluxbound related parameters to the model

- *fbcmod* a CBM2SBML instance
- *add_cbmpy_anno* [default=True] add CBMPy KeyValueData annotation.

cbmpy.CBXML.**sbml_setReactionsL2**(*model*, *fba*, *return_dict=False*)
Add the FBA instance reactions to the SBML model

- *model* an SBML model instance
- *fba* a PySCeSCBM model instance
- *return_dict* [default=False] if True do not add reactions to SBML document instead return a dictionary description of the reactions

cbmpy.CBXML.**sbml_setReactionsL3Fbc**(*fbcmod*, *return_dict=False*, *add_cobra_anno=False*, *add_cbmpy_anno=True*, *fbc_version=1*)
Add the FBA instance reactions to the SBML model

- *fbcmod* a CBM2SBML instance
- *return_dict* [default=False] if True do not add reactions to SBML document instead return a dictionary description of the reactions
- *add_cbmpy_anno* [default=True] add CBMPy KeyValueData annotation. Replaces <notes>
- *add_cobra_anno* [default=False] add COBRA <notes> annotation
- *fbc_version* [default=1] writes either FBC v1 (2013) or v2 (2015)

cbmpy.CBXML.**sbml_setSpeciesL2**(*model*, *fba*, *return_dicts=False*)
Add the species definitions to the SBML object:

- *model* [default='‘] a libSBML model instance or can be None if *return_dicts* == True
- *fba* a PySCeSCBM model instance

- *return_dicts* [default=False] only returns the compartment and species dictionaries without updated the SBML

returns:

- *compartments* a dictionary of compartments (except when give *return_dicts* argument)

cbmpy.CBXML.**sbml_setSpeciesL3**(*model*, *fba*, *return_dicts=False*, *add_cobra_anno=False*, *add_cbmpy_anno=True*, *substance_units=True*)

Add the species definitions to the SBML object:

- *model* and SBML model instance or can be None if *return_dicts* == True
- *fba* a PySCeSCBM model instance
- *return_dicts* [default=False] only returns the compartment and species dictionaries without updating the SBML
- *add_cbmpy_anno* [default=True] add CBMPy KeyValueData annotation. Replaces <notes>
- *add_cobra_anno* [default=False] add COBRA <notes> annotation
- *substance_units* [default=True] defines the species in amounts rather than concentrations (necessary for default mmol/gdw.h)

returns:

- *compartments* a dictionary of compartments (except when given *return_dicts* argument)

cbmpy.CBXML.**sbml_setUnits**(*model*, *units=None*, *give_default=False*, *L3=True*)

Adds units to the model:

- *model* a libSBML model instance
- *units* [default=None] a dictionary of units, if None default units are used
- *give_default* [default=False] if true method returns the default unit dictionary
- *L3* [default=True] use the L3 defaults

cbmpy.CBXML.**sbml_setValidationOptions**(*D*, *level*)

set the validation level of an SBML document

- *D* an SBML document
- *level* the level of consistency check can be either one of:
- 'normal' basic id checking only
- 'full' all checks enabled

cbmpy.CBXML.**sbml_validateDocument**(*D*, *fullmsg=False*)

Validates and SBML document returns three dictionaries, errors, warnings, other and a boolean indicating an invalid document:

- *D* and SBML document
- *fullmsg* [default=False] optionally display the full error message

cbmpy.CBXML.**sbml_writeAnnotationsAsCOBRANote**(*annotations*)

Writes the annotations dictionary as a COBRA compatible SBML <note>

cbmpy.CBXML.**sbml_writeCOBRASBML**(*fba*, *fname*, *directory=None*)
Takes an FBA model object and writes it to file as a COBRA compatible :

- *fba* an fba model object
- *fname* the model will be written as XML to *fname*
- *directory* [default=None] if defined it is prepended to fname

cbmpy.CBXML.**sbml_writeKeyValueDataAnnotation**(*annotations*)
Writes the key:value annotations as a KeyValueData annotation (http://pysces.sourceforge.net/KeyValueData)

cbmpy.CBXML.**sbml_writeSBML2FBA**(*fba*, *fname*, *directory=None*, *sbml_level_version=None*)
Takes an FBA model object and writes it to file as SBML L3 FBA:

- *fba* an fba model object
- *fname* the model will be written as XML to *fname*
- *directory* [default=None] if defined it is prepended to fname
- *sbml_level_version* [default=None] a tuple containing the SBML level and version e.g. (2,4) (ignored)

cbmpy.CBXML.**sbml_writeSBML3FBC**(*fba*, *fname*, *directory=None*, *sbml_level_version=(3, 1)*, *autofix=True*, *return_fbc=False*, *gpr_from_annot=False*, *add_groups=False*, *add_cbmpy_annot=True*, *add_cobra_annot=False*, *xoptions={}*)
Takes an FBA model object and writes it to file as SBML L3 FBC:

- *fba* an fba model object
- *fname* the model will be written as XML to *fname*
- *directory* [default=None] if defined it is prepended to fname
- *sbml_level_version* [default=(3,1)] a tuple containing the SBML level and version e.g. (3,1)
- *autofix* convert <> to <=>=
- *return_fbc* return the FBC converter instance
- *gpr_from_annot* [default=False] if enabled will attempt to add the gene protein associations from the annotations if no gene protein association objects exist
- *add_cbmpy_annot* [default=True] add CBMPy KeyValueData annotation. Replaces <notes>
- *add_cobra_annot* [default=True] add COBRA <notes> annotation
- *xoptions* extended options
  - –*fbc_version* [default=1] write SBML3FBC using version 1 (2013) or version 2 (2015)
  - –*validate* [default=False] validate the output SBML file
  - –*compress_bounds* [default=False] try compress output flux bound parameters
  - –*zip_model* [default=False] compress the model using ZIP encoding
  - –*return_model_string* [default=False] return the SBML XML file as a string

cbmpy.CBXML.**setCBSBOterm**(*sbo*, *obj*)
Given an SBOterm from libSBML, add it to a CBMPy object

- *sbo* the sbo term string
- *obj* the CBMPy Fbase derived object

cbmpy.CBXML.**xml_addSBML2FBAFluxBound**(*document*, *rid*, *operator*, *value*, *fbid=None*)
Adds an SBML3FBA flux bound to the document:

- *document* a minidom XML document created by xml_createSBML2FBADoc
- *rid* the reaction id
- *operator* one of ['greater','greaterEqual','less','lessEqual','equal','>','>=','<','<=','=']
- *value* a float which will be cast to a string using str(value)
- *fbid* the flux bound id, autogenerated by default

cbmpy.CBXML.**xml_addSBML2FBAObjective**(*document*, *objective*, *active=True*)
Adds an objective element to the documents listOfObjectives and sets the active attribute:

- *document* a minidom XML document created by *xml_createSBML2FBADoc*
- *objective* a minidom XML objective element created with *xml_createSBML2FBAObjective*
- *active* [default=True] a boolean flag specifiying whether this objective is active

cbmpy.CBXML.**xml_createListOfFluxObjectives**(*document*, *fluxObjectives*)
Create a list of fluxObjectives to add to an Objective:

- *document* a minidom XML document created by xml_createSBML2FBADoc
- *fluxobjs* a list of (rid, coefficient) tuples

cbmpy.CBXML.**xml_createSBML2FBADoc**()
Create a 'document' to store the SBML2FBA annotation, returns:

- *DOC* a minidom document

cbmpy.CBXML.**xml_createSBML2FBAObjective**(*document*, *oid*, *sense*, *fluxObjectives*)
Create a list of fluxObjectives to add to an Objective:

- *document* a minidom XML document created by xml_createSBML2FBADoc
- *oid* the objective id
- *sense* a string containing the objective sense either: **maximize** or **minimize**
- *fluxObjectives* a list of (rid, coefficient) tuples

cbmpy.CBXML.**xml_getSBML2FBAannotation**(*fba*, *fname=None*)
Takes an FBA model object and returns the SBML3FBA annotation as an XML string:

- *fba* an fba model object
- *fname* [default=None] if supplied the XML will be written to file *fname*

cbmpy.CBXML.**xml_stripTags**(*html*)
Strip a string of HTML/XML, returns a string

- *html* the string containing html

cbmpy.CBXML.**xml_viewSBML2FBAXML**(*document*, *fname=None*)
Print a minidom XML document to screen or file, arguments:

- *document* a minidom XML document
- *fname* [default=None] by default print to screen or write to file fname

3.21 PyscesStoich

PySCeS stoichiometric analysis classes.

class cbmpy.PyscesStoich.**MathArrayFunc**
PySCeS array functions - used by Stoich

MatrixFloatFix(*mat*, *val=1.e-15*)
Clean an array removing any floating point artifacts defined as being smaller than a specified value. Processes an array inplace

Arguments:

mat: the input 2D array val [default=1.e-15]: the threshold value (effective zero)

MatrixValueCompare(*matrix*)
Finds the largest/smallest abs(value) > 0.0 in a matrix. Returns a tuple containing (smallest,largest) values

Arguments:

matrix: the input 2D array

SwapCol(*res_a*, *r1*, *r2*)
Swap two columns using BLAS swap, arrays can be (or are upcast to) type double (d) or double complex (D). Returns the colswapped array

Arguments:

res_a: the input array r1: the first column to be swapped r2: the second column to be swapped

SwapCold(*res_a*, *c1*, *c2*)
Swaps two double (d) columns in an array using BLAS DSWAP. Returns the colswapped array.

Arguments:

res_a: input array c1: column index 1 c2: column index 2

SwapColz(*res_a*, *c1*, *c2*)
Swaps two double complex (D) columns in an array using BLAS ZSWAP. Returns the colswapped array.

Arguments:

res_a: input array c1: column index 1 c2: column index 2

SwapElem(*res_a*, *r1*, *r2*)
Swaps two elements in a 1D vector

Arguments:

res_a: the input vector r1: index 1 r2: index 2

SwapRow(*res_a*, *r1*, *r2*)
Swaps two rows using BLAS swap, arrays can be (or are upcast to) type double (d) or double complex (D). Returns the rowswapped array.

Arguments:

res_a: the input array r1: the first row index to be swapped r2: the second row index to be swapped

SwapRowd(*res_a*, *c1*, *c2*)
Swaps two double (d) rows in an array using BLAS DSWAP. Returns the rowswapped array.

Arguments:

res_a: input array c1: row index 1 c2: row index 2

SwapRowz(*res_a*, *c1*, *c2*)
Swaps two double complex (D) rows in an array using BLAS ZSWAP. Returns the rowswapped array.

Arguments:

res_a: input array c1: row index 1 c2: row index 2

assertRank2(**arrays*)
Check that we are using a 2D array

Arguments:

*arrays: input array(s)

castCopyAndTranspose(*type*, **arrays*)
Cast numeric arrays to required type and transpose

Arguments:

type: the required type to cast to *arrays: the arrays to be processed

commonType(**arrays*)
Numeric detect and set array precision (will be replaced with new scipy.core compatible code when ready)

Arguments:

*arrays: input arrays

class cbmpy.PyscesStoich.**Stoich**(*input*)
PySCeS stoichiometric analysis class: initialized with a stoichiometric matrix N (input)

AnalyseK()
Evaluate the stoichiometric matrix and calculate the nullspace using LU decomposition and backsubstitution . Generates the MCA K and Ko arrays and associated row and column vectors

Arguments: None

AnalyseL()
Evaluate the stoichiometric matrix and calculate the left nullspace using LU factorization and backsubstitution. Generates the MCA L, Lo, Nr and Conservation matrix and associated row and column vectors

Arguments: None

BackSubstitution(*res_a*, *row_vector*, *column_vector*)
Jordan reduction of a scaled upper triangular matrix. The returned array is now in the form [I R] and can be used for nullspace determination. Modified row and column tracking vetors are also returned.

Arguments:

res_a: unitary pivot upper triangular matrix row_vector: row tracking vector column_vector: column tracking vector

GetUpperMatrix(*a*)
Core analysis algorithm; an input is preconditioned using PivotSort_initial and then cycles of PLUfactorize and PivotSort are run until the factorization is completed. During this process the matrix is reordered by column swaps which emulates a full pivoting LU factorization. Returns the pivot matrix P, upper factorization U as well as the row/col tracking vectors.

Arguments:

a: a stoichiometric matrix

GetUpperMatrixUsingQR(*a*)
GetUpperMatrix(a)

Core analysis algorithm; an input is preconditioned using PivotSort_initial and then cycles of PLUfactorize and PivotSort are run until the factorization is completed. During this process the matrix is reordered by column swaps which emulates a full pivoting LU factorization. Returns the pivot matrix P, upper factorization U as well as the row/col tracking vectors.

Arguments:

a: a stoichiometric matrix

K_split_R(*R_a*, *row_vector*, *column_vector*)
Using the R factorized form of the stoichiometric matrix we now form the K and Ko matrices. Returns the r_ipart,Komatrix,Krow,Kcolumn,Kmatrix,Korow,info

Arguments:

R_a: the Gauss-Jordan reduced stoichiometric matrix row_vector: row tracking vector column_vector: column tracking vector

L_split_R(*Nfull*, *R_a*, *row_vector*, *column_vector*)
Takes the Gauss-Jordan factorized N^T and extract the L, Lo, conservation (I -Lo) and reduced stoichiometric matrices. Returns: lmatrix_col_vector, lomatrix, lomatrix_row, lomatrix_co, nrmatrix, Nred_vector_row, Nred_vector_col, info

Arguments:

Nfull: the original stoichiometric matrix N R_a: gauss-jordan factorized form of N^T row_vector: row tracking vector column_vector: column tracking vector

PLUfactorize(*a_in*)
Performs an LU factorization using LAPACK D/ZGetrf. Now optimized for FLAPACK interface. Returns LU - combined factorization, IP - rowswap information and info - Getrf error control.

Arguments:

a_in: the matrix to be factorized

PivotSort(*a*, *row_vector*, *column_vector*)
This is a sorting routine that accepts a matrix and row/colum vectors and then sorts them so that: there are no zero rows (by swapping with first non-zero row) The abs(largest) pivots are moved onto the diagonal to maintain numerical stability. Row and column swaps are recorded in the tracking vectors.

Arguments:

a: the input array row_vector: row tracking vector column_vector: column tracking vector

PivotSort_initial(*a*, *row_vector*, *column_vector*)
This is a sorting routine that accepts a matrix and row/colum vectors and then sorts them so that: the abs(largest) pivots are moved onto the diagonal to maintain numerical stability i.e. the matrix diagonal is in descending max(abs(value)). Row and column swaps are recorded in the tracking vectors.

Arguments:

a: the input array row_vector: row tracking vector column_vector: column tracking vector

SVD_Rank_Check(*matrix=None*, *factor=1.0e4*, *resultback=0*)
Calculates the dimensions of L/L0/K/K) by way of SVD and compares them to the Guass-Jordan results. Please note that for LARGE ill conditioned matrices the SVD can become numerically unstable when used for nullspace determinations

Arguments:

matrix [default=None]: the stoichiometric matrix default is self.Nmatrix factor [default=1.0e4]: factor used to calculate the 'zero pivot' mask = mach_eps*factor resultback [default=0]: return the SVD results, U, S, vh

ScalePivots(*a_one*)
Given an upper triangular matrix U, this method scales the diagonal (pivot values) to one.

Arguments:

a_one: an upper triangular matrix U

SplitLU(*plu*, *row*, *col*, *t*)
PLU takes the combined LU factorization computed by PLUfactorize and extracts the upper matrix. Returns U.

Arguments:

plu: LU factorization row: row tracking vector col: column tracking vector t [default=None)]: typecode argument (currently not used)

class cbmpy.PyscesStoich.**StructMatrix**(*array*, *ridx*, *cidx*, *row=None*, *col=None*)
This class is specifically designed to store structural matrix information give it an array and row/col index permutations it can generate its own row/col labels given the label src.

getColsByIdx(**args*)
Return the columns referenced by index (1,3,5)

getColsByName(**args*)
Return the columns referenced by label ('s','x','d')

getIndexes(*axis='all'*)
Return the matrix indexes ([rows],[cols]) where axis='row'/'col'/'all'

getLabels(*axis='all'*)
Return the matrix labels ([rows],[cols]) where axis='row'/'col'/'all'

getRowsByIdx(**args*)
Return the rows referenced by index (1,3,5)

getRowsByName(**args*)
Return the rows referenced by label ('s','x','d')

setCol(*src*)
Assuming that the col index array is a permutation (full/subset) of a source label array by supplying that src to setCol maps the row labels to cidx and creates self.col (col label list)

setRow(*src*)
Assuming that the row index array is a permutation (full/subset) of a source label array by supplying that source to setRow it maps the row labels to ridx and creates self.row (row label list)

3.22 CBMPy: MultiCoreFVA module

PySCeS Constraint Based Modelling (http://cbmpy.sourceforge.net) Copyright (C) 2009-2017 Brett G. Olivier, VU University Amsterdam, Amsterdam, The Netherlands

This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program. If not, see <http://www.gnu.org/licenses/>

Author: Brett G. Olivier Contact email: bgoli@users.sourceforge.net Last edit: $Author: bgoli $ ($Id: _multicorefva.py 575 2017-04-13 12:18:44Z bgoli $)

3.23 CBMPy: MultiCoreEnvFVA module

PySCeS Constraint Based Modelling (http://cbmpy.sourceforge.net) Copyright (C) 2009-2017 Brett G. Olivier, VU University Amsterdam, Amsterdam, The Netherlands

This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program. If not, see <http://www.gnu.org/licenses/>

Author: Brett G. Olivier Contact email: bgoli@users.sourceforge.net Last edit: $Author: bgoli $ ($Id: _multicoreenvfva.py 575 2017-04-13 12:18:44Z bgoli $)

CHAPTER

FOUR

INDICES AND TABLES

- genindex
- modindex
- search

C