# PSAMM Documentation

*Release 0.29*

**Jon Lund Steffensen**

**Apr 18, 2017**

# Contents


**1 Overview** **3**
- 1.1 Citing PSAMM . . . 3
- 1.2 Software license . . . 3

**2 PSAMM Tutorials** **5**
- 2.1 Installation and Materials . . . 5
- 2.2 Importing, Exporting, and working with Models with PSAMM . . . 9
- 2.3 Model Curation . . . 23
- 2.4 Constraint Based Analysis with PSAMM . . . 31

**3 Install** **39**
- 3.1 Dependencies . . . 39
- 3.2 Cplex . . . 40
- 3.3 Gurobi . . . 40
- 3.4 GLPK . . . 40
- 3.5 QSopt_ex . . . 40

**4 Model file format** **43**
- 4.1 Biomass . . . 44
- 4.2 Extracellular Compartment . . . 44
- 4.3 Default Compartment . . . 44
- 4.4 Compartments . . . 44
- 4.5 Compounds . . . 44
- 4.6 Reactions . . . 45
- 4.7 Exchange compounds . . . 46
- 4.8 Reaction flux limits . . . 47
- 4.9 Model Definition . . . 47

**5 Command line interface** **49**
- 5.1 Linear programming solver . . . 49
- 5.2 Flux balance analysis (`fba`) . . . 50
- 5.3 Flux variability analysis (`fva`) . . . 50
- 5.4 Robustness (`robustness`) . . . 51
- 5.5 Random sparse network (`randomsparse`) . . . 51
- 5.6 Gene Deletion (`genedelete`) . . . 51
- 5.7 Flux coupling analysis (`fluxcoupling`) . . . 52
- 5.8 Stoichiometric consistency check (`masscheck`) . . . 52

5.9 Formula consistency check (formulacheck) . . . 53
5.10 Charge consistency check (chargecheck) . . . 53
5.11 Flux consistency check (fluxcheck) . . . 54
5.12 Reaction duplicates check (duplicatescheck) . . . 54
5.13 Gap check (gapcheck) . . . 54
5.14 GapFill (gapfill) . . . 55
5.15 FastGapFill (fastgapfill) . . . 55
5.16 SBML Export (sbmlexport) . . . 55
5.17 Excel Export (excelexport) . . . 56
5.18 Table Export (tableexport) . . . 56
5.19 Search (search) . . . 56
5.20 Console (console) . . . 56

**6 Development** **57**
6.1 Test suite . . . 57
6.2 Adding new tests . . . 58
6.3 Documentation tests . . . 58

**7 FAQ** **59**

**8 PSAMM API** **61**
8.1 psamm.balancecheck – check balance of charge and formula . . . 61
8.2 psamm.command – Command line interface . . . 62
8.3 psamm.database – Reaction database . . . 63
8.4 psamm.datasource.context – File system contexts . . . 64
8.5 psamm.datasource.entry – Model entry representations . . . 64
8.6 psamm.datasource.kegg – KEGG data parser . . . 65
8.7 psamm.datasource.modelseed – ModelSEED data parser . . . 66
8.8 psamm.datasource.native – Native data format parser . . . 66
8.9 psamm.datasource.reaction – Parser for reactions . . . 72
8.10 psamm.datasource.sbml – SBML model parser . . . 72
8.11 psamm.expression.affine – Affine expressions . . . 75
8.12 psamm.expression.boolean – Boolean expressions . . . 76
8.13 psamm.fastcore – Fastcore (approximate consistent subset) . . . 76
8.14 psamm.fastgapfill – FastGapFill algorithm . . . 77
8.15 psamm.fluxanalysis – Constraint-based reaction flux analysis . . . 77
8.16 psamm.fluxcoupling – Flux coupling analysis . . . 80
8.17 psamm.formula – Chemical compound formula . . . 81
8.18 psamm.gapfill – GapFind/GapFill . . . 82
8.19 psamm.gapfilling – Gap-filling functions . . . 83
8.20 psamm.lpsolver.cplex – CPLEX LP solver . . . 84
8.21 psamm.lpsolver.generic – Generic linear programming solver . . . 85
8.22 psamm.lpsolver.glpk – GLPK LP solver . . . 86
8.23 psamm.lpsolver.gurobi – Gurobi LP solver . . . 87
8.24 psamm.lpsolver.lp – Linear programming problems . . . 88
8.25 psamm.lpsolver.qsoptex – QSopt_ex LP solver . . . 92
8.26 psamm.massconsistency – Mass consistency check . . . 93
8.27 psamm.metabolicmodel – Metabolic model representation . . . 94
8.28 psamm.randomsparse – Find a random minimal network of model reactions . . . 96
8.29 psamm.reaction – Reaction equations and compounds . . . 96
8.30 psamm.util – Internal utilities . . . 98

**9 References** **101**

**10 Indices and tables** **103**

Bibliography 105

Python Module Index 107

Contents:

CHAPTER 1

# Overview

PSAMM is an open source software that is designed for the curation and analysis of metabolic models. It supports model version tracking, model annotation, data integration, data parsing and formatting, consistency checking, automatic gap filling, and model simulations.

PSAMM is developed as an open source project, coordinated through Github. The PSAMM software is being developed in the Zhang Laboratory at the University of Rhode Island.

## Citing PSAMM

If you use PSAMM in a publication, please cite:

Steffensen JL, Dufault-Thompson K, Zhang Y. PSAMM: A Portable System for the Analysis of Metabolic Models. PLOS Comput Biol. Public Library of Science; 2016;12: e1004732. doi:10.1371/journal.pcbi.1004732.

## Software license

PSAMM 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/.

CHAPTER 2

# PSAMM Tutorials

## Installation and Materials

This tutorial will show you how to get PSAMM up and running on your computer, how to work with the PSAMM YAML format, how to import published models into PSAMM, and how to apply the main tools included with PSAMM to your models.


- *Downloading the PSAMM Tutorial Data*
- *PSAMM Installation*
- *PSAMM Model Collection*


### Downloading the PSAMM Tutorial Data

The PSAMM tutorial materials are available in the psamm-tutorial GitHub repository

These files can be downloaded using the following command:

```
$ git clone https://github.com/zhanglab/psamm-tutorial.git
```

This will create a directory named `psamm-tutorial` in your current working folder. You can then navigate to this directory using the following command:

```
$ cd psamm-tutorial
```

Now you should be in the `psamm-tutorial` folder and should see the following folders:

```
additional_files/
E_coli_sbml/
E_coli_excel/
E_coli_json/
```

These directories include all of the files that will be needed to run the tutorial.

## PSAMM Installation

PSAMM can be installed using the Python package installer pip. We recommend that all installations be performed under a virtual Python environment. Major programs and dependencies include: `psamm-model`, which supports model checking, model simulation, and model exports; Linear programming (LP) solvers (e.g. CPLEX, Gurobi, QSopt_ex), which provide the solution of linear programming problems; `psamm-import`, which supports the import of models from SBML, JSON, and Excel formats.

### Setting up a Virtual Python Environment

It is recommended that the PSAMM software and dependencies should be installed under a virtual Python environment. This can be done by using the Virtualenv software. Virtualenv will set up a Python environment that permits you to install Python packages in a local directory that will not interfere with other programs in the global Python. The virtual environment can be set up at any local directory that you have write permission to. For example, here we will set up the virtual environment under the main directory of this PSAMM tutorial. First, run the following command if you are not in the `psamm-tutorial` folder:

```
$ cd <PATH>/psamm-tutorial
```

In this command, `<PATH>` should be substituted by the directory path to where you created the `psamm-tutorial`. This will change your current directory to the `psamm-tutorial` directory. Then, you can create a virtual environment in the `psamm-tutorial` directory:

```
$ virtualenv psamm-env
```

That will set up the virtual environment in a folder called `psamm-env/`. The next step is to activate the virtual environment so that the Python that is being used will be the one that is in the virtualenv. To do this use the following command:

```
$ source psamm-env/bin/activate
```

This will change your command prompt to the following:

```
(psamm-env) $
```

This indicates that the virtual environment is activated, and any installation of Python packages will now be installed in the virtual environment. It is important to note that when you leave the environment and return at a later time, you will have to reactivate the environment (use the `source` command above) to be able to use any packages installed in it.

**Note:** For Windows users, the virtual environment is installed in a different file structure. The `activate` script on these systems will reside in a `Scripts` folder. To activate the environment on these systems use the command:

```
> psamm-env\Scripts\activate
```

**Note:** After activating the environment, the command `pip list` can be used to quickly get an overview of the packages installed in the environment and the version of each package.

### Installation of `psamm-model` and `psamm-import`

The next step will be to install `psamm-model` and `psamm-import` as well as their requirements. To do this, you can use the Python Package Installer, *pip*. To install both `psamm-import` and `psamm-model` you can use the following command:

```
(psamm-env) $ pip install git+https://github.com/zhanglab/psamm-import.git
```

This will install `psamm-import` from its Git repository and also install its Python dependencies automatically. One of these dependencies is `psamm-model`, so when `psamm-import` is installed you will also be installing `psamm-model`.

If you only want to install `psamm-model` in the environment you can run the following command:

```
(psamm-env) $ pip install psamm
```

It is important to note that if only `psamm-model` is installed you will be able to apply PSAMM only on models that are represented in the YAML format which will be described later on in the tutorial.

### Installation of LP Solvers

The LP (linear programming) solvers are necessary for analysis of metabolic fluxes using the constraint-based modeling approaches.

CPLEX is the recommended solver for PSAMM and is available with an academic license from IBM. Make sure that you use at least **CPLEX version 12.6**. Instructions on how to install CPLEX can be found here.

Once CPLEX is installed, you need to install the Python bindings under the psamm-env virtual environment using the following command:

```
(psamm-env) $ pip install <PATH>/IBM/ILOG/CPLEX_Studio<XXX>/cplex/python/<python_
↪version>/<platform>
```

The directory path in the above command should be replaced with the path to the IBM CPLEX installation in your computer. This will install the Python bindings for CPLEX into the virtual environment.

---

**Note:** While the CPLEX software will be installed globally, the Python bindings should be installed specifically under the virtual environment with the PSAMM installation.

---

PSAMM also supports the use of two other linear programming solvers, Gurobi and QSopt_ex. To install the Gurobi solver, Gurobi will first need to be installed on your computer. Gurobi can be obtained with an academic license from here: Gurobi

Once Gurobi is installed the Python bindings will need to be installed in the virtual environment by using pip to install them from the package directory. An example of how this could be done on a macOS is (on other platforms the path will be different):

```
(psamm-env) $ pip install /Library/gurobi604/mac64/
```

The QSopt_ex solver can also be used with PSAMM. To install this solver you will first need to install Qsopt_ex on your computer and afterwards the Python bindings (*python-qsoptex*) can be installed in the virtual environment:

```
(psamm-env) $ pip install python-qsoptex
```

Please see the python-qsoptex documentation for more information on installing both the library and the Python bindings.

**Note:** The QSopt_ex solver does not support Integer LP problems and as a result cannot be used to perform flux analysis with thermodynamic constraints. If this solver is used thermodynamic constraints cannot be used during simulation. By default `psamm-model` will not use these constraints.

Once a solver is installed you should now be able to fully use all of the `psamm-model` flux analysis functions. To see a list of the installed solvers the use the `psamm-list-lpsolvers` command:

```
(psamm-env) $ psamm-list-lpsolvers
```

You will see the details on what solvers are installed currently and avaliable to PSAMM. For example if the Gurobi and CPLEX solvers were both installed you would see the following output from `psamm-list-lpsolvers`:

```
Prioritized solvers:
Name: cplex
Priority: 10
MILP (integer) problem support: True
Rational solution: False
Class: <class 'psamm.lpsolver.cplex.Solver'>

Name: gurobi
Priority: 9
MILP (integer) problem support: True
Rational solution: False
Class: <class 'psamm.lpsolver.gurobi.Solver'>

Unavailable solvers:
qsoptex: Error loading solver: No module named qsoptex
```

By default the solver with the highest priority (highest priority number) is used in constraint based simulations. If you want to use a solver with a lower priority you will need to specify it by using the `--solver` option. For example to run FBA on a model while using the Gurobi solver the following command would be used:

```
(psamm-env) $ psamm-model fba --solver name=gurobi
```

## PSAMM Model Collection

Converted versions of 57 published SBML metabolic models, 9 published Excel models and one MATLAB model are available in the PSAMM Model Collection on GitHub. These models were converted to the YAML format and then manually edited when needed to produce models that can generate non-zero biomass fluxes. The changes to the models are tracked in the Git history of the repository so you can see exactly what changes needed to be made to the models. To download and use these models with *PSAMM* you can clone the Git repository using the following command:

```
$ git clone https://github.com/zhanglab/psamm-model-collection.git
```

This will create the directory `psamm-model-collection` in your current folder that contains one directory named `excel` with the converted Excel models, one directory named `sbml` with the converted SBML models and one named `matlab` with the converted MATLAB model. These models can then be used for simulations with *PSAMM* using the commands detailed in this tutorial.

# Importing, Exporting, and working with Models with PSAMM

This part of the tutorial will focus on how to use PSAMM to convert files between the YAML format and other popular formats. An additional description of the YAML model format and its features is also provided here.


- *Import Functions in PSAMM*
- *Importing Existing Models (psamm-import)*
- *YAML Format and Model Organization*
- *Version Control with the YAML Format*
- *Using PSAMM to export the model to other Software*


## Import Functions in PSAMM

For information on how to install *PSAMM* and the associated requirements, as well how to download the materials required for this tutorial you can reference the Installation and Materials section of the tutorial.

## Importing Existing Models (psamm-import)

In order to work with a metabolic model in PSAMM the model must be in the PSAMM-specific YAML format. This format allows for a human readable representation of the model components and allows for enhanced customization with respect to the organization of the metabolic model. This enhanced organization will allow for a more direct interaction with the metabolic model and make the model more accessible to both the modeler and experimental biologists.

### Import Formats

The `psamm-import` program supports the import of models in various formats. For the SBML format, it supports the COBRA-compliant SBML specifications, the FBC specifications, and the basic SBML specifications in levels 1, 2, and 3; for the JSON format, it supports the import of JSON files directly from the BiGG database or from locally downloaded versions; the support for importing from Excel file is model specific and are available for 17 published models. There is also a generic Excel import for models produced by the ModelSEED pipeline. To see a list of these models or model formats that are supported, use the command:

```
(psamm-env) $ psamm-import list
```

In the output, you will see a list of specific Excel models that are supported by `psamm-import` as well as the different SBML parsers that are available in PSAMM:

```
Generic importers:
json            COBRApy JSON
modelseed       ModelSEED model (Excel format)
sbml            SBML model (non-strict)
sbml-strict     SBML model (strict)

Model-specific importers:
icce806         Cyanothece sp. ATCC 51142 iCce806 (Excel format), Vu et al., 2012
ecoli_textbook  Escerichia coli Textbook (core) model (Excel format), Orth et al.,␣
↪2010
```

```
ijo1366       Escherichia coli iJO1366 (Excel format), Orth et al., 2011
gsmn-tb       Mycobacterium tuberculosis GSMN-TB (Excel format), Beste et al., 2007
inj661        Mycobacterium tuberculosis iNJ661 (Excel format), Jamshidi et al., 2007
inj661m       Mycobacterium tuberculosis iNJ661m (Excel format), Fang et al., 2010
inj661v       Mycobacterium tuberculosis iNJ661v (Excel format), Fang et al., 2010
ijn746        Pseudomonas putida iJN746 (Excel format), Nogales et al., 2011
ijp815        Pseudomonas putida iJP815 (Excel format), Puchalka et al., 2008
stm_v1.0      Salmonella enterica STM_v1.0 (Excel format), Thiele et al., 2011
ima945        Salmonella enterica iMA945 (Excel format), AbuOun et al., 2009
irr1083       Salmonella enterica iRR1083 (Excel format), Raghunathan et al., 2009
ios217_672    Shewanella denitrificans OS217 iOS217_672 (Excel format), Ong et al.,␣
↪2014
imr1_799      Shewanella oneidensis MR-1 iMR1_799 (Excel format), Ong et al., 2014
imr4_812      Shewanella sp. MR-4 iMR4_812 (Excel format), Ong et al., 2014
iw3181_789    Shewanella sp. W3-18-1 iW3181_789 (Excel format), Ong et al., 2014
isyn731       Synechocystis sp. PCC 6803 iSyn731 (Excel format), Saha et al., 2012
```

Now the model can be imported using the `psamm-import` functions. Return to the `psamm-tutorial` folder if you have left it using the following command:

```
(psamm-env) $ cd <PATH>/tutorial-part-1
```

### Importing an SBML Model

In this tutorial, we will use the *E. coli* textbook core model *[Orth13]* as an example to demonstrate these functions in PSAMM. First, we will convert the model from the SBML model. To import the `E_coli_core.xml` model to YAML format run the following command:

```
(psamm-env) $ psamm-import sbml --source E_coli_sbml/ecoli_core_model.xml --dest E_
↪coli_yaml
```

This will convert the SBML file in the `E_coli_sbml` directory into the YAML format that will be stored in the `E_coli_yaml/` directory. The output will give the basic statistics of the model and should look like this:

```
...
WARNING: Species M_pyr_b was converted to boundary condition because of "_b" suffix
WARNING: Species M_succ_b was converted to boundary condition because of "_b" suffix
INFO: Detected biomass reaction: R_Biomass_Ecoli_core_w_GAM
INFO: Removing compound prefix 'M_'
INFO: Removing reaction prefix 'R_'
INFO: Removing compartment prefix 'C_'
Model: Ecoli_core_model
- Biomass reaction: Biomass_Ecoli_core_w_GAM
- Compartments: 2
- Compounds: 72
- Reactions: 95
- Genes: 137
INFO: e is extracellular compartment
INFO: Using default flux limit of 1000.0
INFO: Converting exchange reactions to exchange file
```

`psamm-import` will produce some warnings if there are any aspects of the model that are going to be changed during import. In this case the warnings are notifying you that the metabolites with a _b suffix have been converted to the boundary conditions of the model. There will also be information on what prefixes were removed from the metabolite IDs and if the importer was able to identify the Biomass Reaction in the model. This information is important to check

to make sure that the model was imported correctly. After the import the model will be available and ready to use for any other PSAMM functions.

### Importing an Excel Model

The process of importing an Excel model is the same as importing an SBML model except that you will need to specify the specific model name in the command. The list of supported models can be seen using the list function above. An example of an Excel model import is below:

```
(psamm-env) $ psamm-import ecoli_textbook --source E_coli_excel/ecoli_core_model.xls -
↪-dest converted_excel_model
```

This will produce a YAML version of the Excel model in the `converted_excel_model/` directory.

Since the Excel models are not in a standardized format these parsers need to be developed on a model-by-model basis in order to parse all of the relevant information out of the model. This means that the parser can only be used for the listed models and not for a general import.

### Importing a JSON Model

`psamm-import` also supports the conversion of JSON format models that follows the conventions in COBRApy. If the JSON model is stored locally, it can be converted with the following command:

```
(psamm-env) $ psamm-import json --source E_coli_json/e_coli_core.json --dest␣
↪converted_json_model/
```

Alternatively, an extension of the JSON importer has been provided, `psamm-import-bigg`, which can be applied to convert JSON models from BiGG database. To see the list of available models on the BiGG database the following command can be used:

```
(psamm-env) $ psamm-import-bigg list
```

This will show the available models as well as their names. You can then import any of these models to YAML format. For example, using the following command to import the *E. coli* iJO1366 *[Orth11]* model from the BiGG database:

```
(psamm-env) $ psamm-import-bigg iJO1366 --dest converted_json_model_bigg/
```

**Note:** To use `psamm-import-bigg` you must have internet access to download the models remotely.

## YAML Format and Model Organization

Now that we have imported the models into the YAML format we can take a look at what the different files are and what information they contain. The PSAMM YAML format stores individual models under a designated directory, in which there will be a number of files that stores the information of the model and specifies the simulation conditions. The entry point of the YAML model is a file named `model.yaml`, which points to additional files that store the information of the model components, including compounds, reactions, flux limits, exchange conditions, etc. While we recommend that you use the name `model.yaml` for the central reference file, the file names for the included files are flexible and can be customized as you prefer. In this tutorial, we simply used the names: `compounds.yaml`, `reactions.yaml`, `limits.yaml`, and `exchange.yaml` for the included files.

First change directory into `E_coli_yaml`:

```
(psamm-env) $ cd E_coli_yaml/
```

The directory contains the main `model.yaml` file as well as the other files that contain the model data:

```
(psamm-env) $ ls
compounds.yaml
exchange.yaml
limits.yaml
model.yaml
reactions.yaml
```

These files can be opened using any standard text editor. We highly recommend using an editor that includes syntax highlighting for the YAML language (one such editor is the Atom editor which includes built-in support for YAML and is available for macOS, Linux and Windows). You can also use commands like `less` and editors like `vi` or `nano` to quickly inspect and edit the files from the command line:

```
(psamm-env) $ less <file_name>.yaml
```

The central file in this organization is the `model.yaml` file. The following is an example of the `model.yaml` file that is obtained from the import of the *E. coli* textbook model. The `model.yaml` file for this imported SBML model should look like the following:

```
name: Ecoli_core_model
biomass: Biomass_Ecoli_core_w_GAM
default_flux_limit: 1000.0
compartments:
- id: c
  adjacent_to: e
  name: Cytoplasm
- id: e
  adjacent_to: c
  name: Extracellular
compounds:
- include: compounds.yaml
reactions:
- include: reactions.yaml
exchange:
- include: exchange.yaml
limits:
- include: limits.yaml
```

The `model.yaml` file defines the basic components of a metabolic model, including the model name (*Ecoli_core_model*), the biomass function (*Biomass_Ecoli_core_w_GAM*), the compound files (`compounds.yaml`), the reaction files (`reactions.yaml`), the flux boundaries (`limits.yaml`), and the exchange conditions (`exchange.yaml`). The additional files are defined using include functions. This organization allows you to easily change aspects of the model like the exchange reactions by simply referencing a different exchange file in the central `model.yaml` definition. In addition to the information on the other components of the model there will also be details on the compartment information for the model. This will provide an overview of how compartments are related to each other and what their abbreviations and names are. For this small model there is only an `e` and a `c` compartment representing the cytoplasm and extracellular space but more complex cells with multiple compartments can also be represented.

This format can also be used to include multiple files in the list of reactions and compounds. This feature can be useful, for example, if you want to name different reaction files based on the subsystem designations or cellular compartments, or if you want to separate the temporary reactions that are used to fill reaction gaps from the main model. An example of how you could designate multiple reaction files is found below. This file can be found in the additional files folder in the file `complex_model.yaml`.

```
name: Ecoli_core_model
biomass: Biomass_Ecoli_core_w_GAM
default_flux_limit: 1000.0
compartments:
- id: c
  adjacent_to: e
  name: Cytoplasm
- id: e
  adjacent_to: c
  name: Extracellular
model:
- include: core_model_definition.tsv
compounds:
- include: compounds.yaml
reactions:
- include: reactions/cytoplasm.yaml
- include: reactions/periplasm.yaml
- include: reactions/transporters.yaml
- include: reactions/extracellular.yaml
exchange:
- include: exchange.yaml
limits:
- include: limits.yaml
```

As can be seen here the modeler chose to distribute their reaction database files into different files representing various cellular compartments and roles. This organization can be customized to suit your preferred workflow.

There are also situations where you may wish to designate only a subset of the reaction database in a metabolic simulation. In these situations it is possible to use a model definition file to identify which subset of reactions will be used from the larger database. The model definition file is simply a list of reaction IDs that will be included in the simulation.

An example of how to include a model definition file can be found below.

```
name: Ecoli_core_model
biomass: Biomass_Ecoli_core_w_GAM
default_flux_limit: 1000.0
compartments:
- id: c
  adjacent_to: e
  name: Cytoplasm
- id: e
  adjacent_to: c
  name: Extracellular
model:
- include: subset.tsv
compounds:
- include: compounds.yaml
reactions:
- include: reactions.yaml
exchange:
- include: exchange.yaml
limits:
- include: limits.yaml
```

**Note:** When the model definition file is not identified, PSAMM will include the entire reaction database in the model. However, when it is identified, PSAMM will only include the reactions that are listed in the model definition file in the

model. This design can be useful when you want to make targeted tests on a subset of the model or when you want to include a larger database for use with the gap filling functions.

### Reactions

The `reactions.yaml` file is where the reaction information is stored in the model. A sample from this file can be seen below:

```
- id: ACALD
  name: acetaldehyde dehydrogenase (acetylating)
  genes: b0351 or b1241
  equation: '|acald[c]| + |coa[c]| + |nad[c]| <=> |accoa[c]| + |h[c]| +
    |nadh[c]|'
  subsystem: Pyruvate Metabolism
- id: ACALDt
  name: acetaldehyde reversible transport
  genes: s0001
  equation: '|acald[e]| <=> |acald[c]|'
  subsystem: Transport, Extracellular
```

Each reaction entry is designated with the reaction ID first. Then the various properties of the reaction can be listed below it. The required properties for a reaction are ID and equation. Along with these required attributes others can be included as needed in a specific project. These can include but are not limited to EC numbers, subsystems, names, and genes associated with the reaction. For example, in a collaborative reconstruction you may want to include a field named `authors` to identify which authors have contributed to the curation of a particular reaction.

Reaction equations can be formatted in multiple ways to allow for more flexibility during the modeling process. The reactions can be formatted in a string format based on the ModelSEED reaction format. In this representation individual compounds in the reaction are represented as compound IDs followed by the cellular compartment in brackets, bordered on both sides by single pipes. For example if a hydrogen compound, `Hydr`, in a `cytosol` compartment was going to be in an equation it would be represented as follows:

```
|Hydr[cytosol]|
```

These individual compounds can be assigned stoichiometric coefficients by adding a number in parentheses before the compound. For example if a reaction contained two hydrogens it could appear as follows:

```
(2) |Hydr[cytosol]|
```

These individual components are separated by + signs in the reaction string. The separation of the reactants and products is through the use of an equal sign with greater than or less than signs designating directionality. These could include => or <= for reactions that can only progress in one direction or <=> for reactions that can progress in both directions. An example of a correctly formatted reaction could be as follows:

```
'|ac[c]| + |atp[c]| <=> |actp[c]| + |adp[c]|'
```

For longer reactions the YAML format provides a way to list each reaction component on a single line. For example a reaction could be represented as follows:

```
- id: ACKr
  name: acetate kinase
  equation:
    compartment: c
    reversible: yes
    left:
```

```
      - id: ac_c
        value: 1
      - id: atp_c
        value: 1
    right:
      - id: actp_c
        value: 1
      - id: adp_c
        value: 1
  subsystem: Pyruvate Metabolism
```

This line based format can be especially helpful when dealing with larger equations like biomass reactions where there can be dozens of components in a single reaction.

Gene associations for the reactions in a model can also be included in the reaction definitions so that gene essentiality experiments can be performed with the model. These genes associations are included by adding the `genes` property to the reaction like follows:

```
- id: ACALDt
  name: acetaldehyde reversible transport
  equation: '|acald[e]| <=> |acald[c]|'
  subsystem: Transport, Extracellular
  genes: gene_0001
```

More complex gene associations can also be included by using logical and/or statements in the genes property. When performing gene essentiality simulations this logic will be taken into account. Some examples of using this logic with the genes property can be seen below:

```
genes: gene_0001 or gene_0002

genes: gene_0003 and gene_0004

genes: gene_0003 and gene_0004 or gene_0005 and gene_0006

genes: gene_0001 and (gene_0002 or gene_0003)
```

### Compounds

The `compounds.yaml` file is organized in a similar way as the `reactions.yaml`. An example can be seen below.

```
- id: 13dpg_c
  name: 3-Phospho-D-glyceroyl-phosphate
  formula: C3H4O10P2
- id: 2pg_c
  name: D-Glycerate-2-phosphate
  formula: C3H4O7P
- id: 3pg_c
  name: 3-Phospho-D-glycerate
  formula: C3H4O7P
```

The compound entries begin with a compound ID which is then followed by the compound properties. These properties can include a name, chemical formula, and charge of the compound.

### Limits

The limits file is used to designate reaction flux limits when it is different from the defaults in PSAMM. By default, PSAMM would assign the lower and upper bounds to reactions based on their reversibility, i.e. the boundary of reversible reactions are $-1000 \leq v_j \leq 1000$, and the boundary for irreversible reactions are $0 \leq v_j \leq 1000$. Therefore, the `limits.yaml` file will consist of only the reaction boundaries that are different from these default values. For example, if you want to force flux through an artificial reaction like the ATP maintenance reaction *ATPM* you can add in a lower limit for the reaction in the limits file like this:

```
- reaction: ATPM
  lower: 8.39
```

Each entry in the limits file includes a reaction ID followed by upper and lower limits. Note that when a model is imported only the non-default flux limits are explicitly stated, so some of the imported models will not contain a predefined limits file. In the *E. coli* core model, only one reaction has a non-default limit. This reaction is an ATP maintenance reaction and the modelers chose to force a certain level of flux through it to simulate the general energy cost of cellular maintenance processes.

### Exchange

The exchange file is where you can designate the boundary conditions for the model. The compartment of the exchange compounds can be designated using the `compartment` tag, and if omitted, the extracellular compartment (*e*) will be assumed. An example of the exchange file can be seen below.

```
compounds:
- id: ac_e
  reaction: EX_ac_e
  lower: 0.0
- id: acald_e
  reaction: EX_acald_e
  lower: 0.0
- id: akg_e
  reaction: EX_akg_e
  lower: 0.0
- id: co2_e
  reaction: EX_co2_e
```

Each entry starts with the ID of the boundary compound and followed by lines that defines the lower and upper limits of the compound flux. Internally, PSAMM will translate these boundary compounds into exchange reactions in metabolic models. Additional properties can be designated for the exchange reactions including an ID for the reaction, the compartment for the reaction, and lower and upper flux bounds for the reaction. In the same way that only non-standard limits need to be specified in the limits file, only non-standard exchange limits need to be specified in the exchange file. This can be seen with the example above where the upper limits are not set since they should just be the default limit of 1000.

### Model Format Customization

The YAML model format is highly customizable to suit your preferences. File names can be changed according to your own design. These customizations are all allowed by PSAMM as long as the central `model.yaml` file is also updated to reflect the different file names referred. While all the file names can be changed it is recommended that the central `model.yaml` file name does not change. PSAMM will automatically detect and read the information from the file if it is named `model.yaml`. If you *do* wish to also alter the name of this file you will need to specify the path of your model file using the `--model` option whenever any PSAMM commands are run. For example, to run FBA with a different central model file named `ecoli_model.yaml`, you could run the command like this:

```
(psamm-env) $ psamm-model --model ecoli_model.yaml fba
```

## Version Control with the YAML Format

The YAML format contains a logical division of the model information and allows for easier modification and interaction with the model. Moreover, the text-based representation of YAML files can enable the tracking of model modifications using version control systems. In this tutorial we will demonstrate the use of the Git version control system during model development to track the changes that have been added to an existing model. This feature will improve the documentation of the model development process and improve collaborative annotations during model curation.

A broad overview of how to use various Git features can be found here: Git

### Initiate a Git Repository for the YAML Model

Throughout this tutorial version tracking using Git will be highlighted in various sections. As you follow along with the tutorial you can try to run the Git commands to get a sense of how Git and PSAMM work together. We will also highlight how the features of Git help with model curation and development when using the YAML format.

To start using Git to track the changes in this git model the folder must first be initialized as a Git repository. To do this first enter the YAML model directory and use the following command:

```
(psamm-env) $ git init
Initialized empty Git repository in <...>/psamm-tutorial/E_coli_yaml/.git/
```

After the folder is initialized as a Git repository the files that were initially imported from the SBML version can be added to the repository using the following command:

```
(psamm-env) $ git add *.yaml
```

this will stage all of the files with the `yaml` extension to be committed. Then the addition of these files can be added to the repository to be tracked by using the following command:

```
(psamm-env) $ git commit -m 'Initial import of E. coli Core Model'
```

Now these files will be tracked by Git and any changes that are made will be easily viewable using various Git commands. PSAMM will also print out the Git commit ID when any commands are run. This makes it easier for you to track exactly what version of the model a past simulation was done on.

The next step in the tutorial will be to add in a new carbon utilization pathway to the *E. coli* core model and Git will be used to track these new additions and manage the curation in an easy to track manner. The tutorial will return to the version tracking at various points in order to show how this can be used during model development.

### FBA on Model Before Expansion

Now that the model is imported and being tracked by Git it will be helpful to do a quick simulation to confirm that the model is complete and able to generate flux. To do this you can run the FBA command in the model directory:

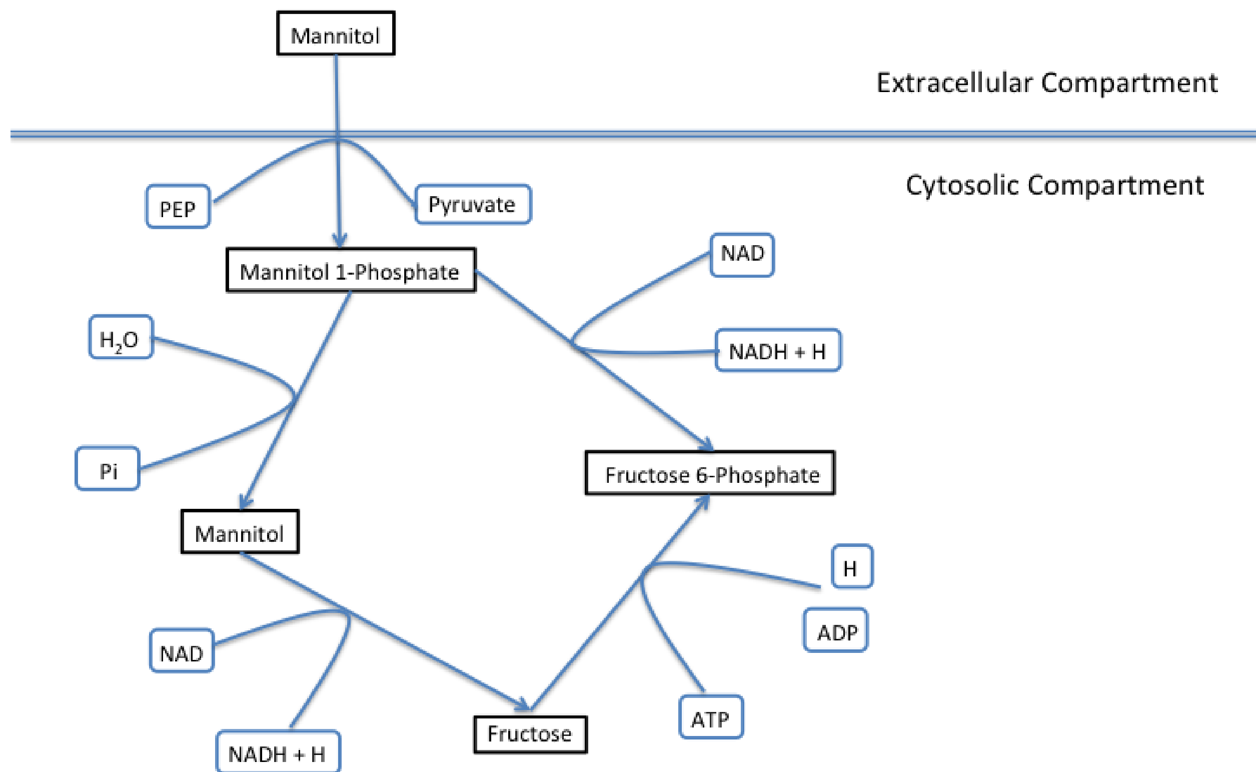
```
(psamm-env) $ psamm-model fba
```

The following is a sample of the output from this initial flux balance analysis. It can be seen that the model is generating flux through the objective function and seems to be a complete working model. Now that this is known any future changes that are made to the model can be made with the knowledge that the unchanged model was able to generate biomass flux.

```
ACONTa       6.00724957535   |Citrate[c]| <=> |cis-Aconitate[c]| + |H2O[c]|  b0118 or␣
↪b1276
ACONTb       6.00724957535   |cis-Aconitate[c]| + |H2O[c]| <=> |Isocitrate[c]|       ␣
↪b0118 or b1276
...
INFO: Objective flux: 0.873921506968
```

### Adding a new Pathway to the Model

The *E. coli* textbook model that was imported above is a small model representing the core metabolism of *E. coli*. This model is great for small tests and demonstrations due to its size and excellent curation. For the purposes of this tutorial this textbook model will be modified to include a new metabolic pathway for the utilization of D-Mannitol by *E. coli*. This is a simple pathway which involves the transport of D-Mannitol via the PTS system and then the conversion of D-Mannitol 1-Phosphate to D-Fructose 6-Phosphate. Theoretically the inclusion of this pathway should allow the model to utilize D-Mannitol as a sole carbon source. Along with this direct pathway another set of reactions will be added that remove the phosphate from the mannitol 1-phosphate to create cytoplasmic mannitol which can then be converted to fructose and then to fructose 6-phosphate.




To add these reactions, there will need to be three components added to the model. First the new reactions will be added to the model, then the relevant exchange reactions, and finally the compound information.

The new reactions in the database can be added directly to the already generated reactions file but for this case they will be added to a separate database file that can then be added to the model through the include function in the `model.yaml` file.

A reaction database file named `mannitol_path.yaml` is supplied in `additional_files` folder. This file can be added into the `model.yaml` file by copying it to your working folder using the following command:

```
(psamm-env) $ cp ../additional_files/mannitol_pathway.yaml .
```

And then specifying it in the `model.yaml` file by adding the following line in the reactions section:

```
reactions:
- include: reactions.yaml
- include: mannitol_pathway.yaml
```

Alternatively you can copy an already changed `model.yaml` file from the additional files folder using the following command:

```
(psamm-env) $ cp ../additional_files/model.yaml .
```

This line tells PSAMM that these reactions are also going to be included in the model simulations.

Now you can test the model again to see if there were any effects from these new reactions added in. To run an FBA simulation you can use the following command:

```
(psamm-env) $ psamm-model fba --all-reactions
```

The `--all-reactions` option makes the command write out all reactions in the model even if they have a flux of zero in the simulation result. It can be seen that the newly added reactions are being read into the model since they do appear in the output. For example the *MANNI1DEH* reaction can be seen in the FBA output and it can be seen that this reaction is not carrying any flux. This is because there is no exchange reaction added into the model that would provide mannitol.

```
FRUKIN      0.0     |fru[c]| + |ATP[c]| => |D-Fructose-6-phosphate[c]| + |ADP[c]| +␣
↪|H[c]|
...
MANNI1PDEH  0.0     |Nicotinamide-adenine-dinucleotide[c]| + |manni1p[c]| => |D-
↪Fructose-6-phosphate[c]| + |H[c]| + |Nicotinamide-adenine-dinucleotide-reduced[c]|
MANNI1PPHOS 0.0     |manni1p[c]| + |H2O[c]| => |manni[c]| + |Phosphate[c]|
MANNIDEH    0.0     |Nicotinamide-adenine-dinucleotide[c]| + |manni[c]| =>␣
↪|Nicotinamide-adenine-dinucleotide-reduced[c]| + |fru[c]|
MANNIPTS    0.0     |manni[e]| + |Phosphoenolpyruvate[c]| => |manni1p[c]| +␣
↪|Pyruvate[c]|
...
```

### Changing the Boundary Definitions Through the Exchange File

To add new exchange reactions to the model a modified `exchange.yaml` file has been included in the additional files. This new boundary condition could be added by creating a new entry in the existing `exchange.yaml` file but for this tutorial the exchange file can be changed by running the following command:

```
(psamm-env) $ cp ../additional_files/exchange.yaml .
```

This will simulate adding in the new mannitol compound into the exchange file as well as setting the uptake of glucose to be zero.

Now you can track changes to the exchange file using the Git command:

```
(psamm-env) $ git diff exchange.yaml
```

From the output, it can be seen that a new entry was added in the exchange file to add the mannitol exchange reaction and that the lower flux limit for glucose uptake was changed to zero. This will ensure that any future simulations done with the model in these conditions will only have mannitol available as a carbon source.

```
@@ -1,5 +1,7 @@
 name: Default medium
 compounds:
+- id: manni
+  lower: -10
 - id: ac_e
   reaction: EX_ac
   lower: 0.0
@@ -25,7 +27,7 @@
   lower: 0.0
 - id: glc_D_e
   reaction: EX_glc
-  lower: -10.0
+  lower: 0.0
 - id: gln_L_e
   reaction: EX_gln_L
   lower: 0.0
```

In this case the Git output indicates what lines were added or removed from the previous version. Added lines are indicated with a plus sign next to them. These are the new lines in the new version of the file. The lines with a minus sign next to them are the line versions from the old format of the file. This makes it easy to figure out exactly what changed between the new and old version of the file.

Now you can test out if the new reactions are functioning in the model. Since there is no other carbon source, if the model sustains flux through the biomass reaction it must be from the supplied mannitol. The following command can be used to run FBA on the model:

```
(psamm-env) $ psamm-model fba --all-reactions
```

From the output it can be seen that there is flux through the biomass reaction and that the mannitol utilization reactions are being used. In this situation it can also be seen that the pathway that converts mannitol to fructose first is not being used.

```
FRUKIN       0.0      |fru[c]| + |ATP[c]| => |D-Fructose-6-phosphate[c]| + |ADP[c]| +␣
↪|H[c]|
...
MANNI1PDEH   10.0     |Nicotinamide-adenine-dinucleotide[c]| + |manni1p[c]| => |D-
↪Fructose-6-phosphate[c]| + |H[c]| + |Nicotinamide-adenine-dinucleotide-reduced[c]|
MANNI1PPHOS  0.0      |manni1p[c]| + |H2O[c]| => |manni[c]| + |Phosphate[c]|
MANNIDEH     0.0      |Nicotinamide-adenine-dinucleotide[c]| + |manni[c]| =>␣
↪|Nicotinamide-adenine-dinucleotide-reduced[c]| + |fru[c]|
MANNIPTS     10.0     |manni[e]| + |Phosphoenolpyruvate[c]| => |manni1p[c]| +␣
↪|Pyruvate[c]|
```

You can also choose to maximize other reactions in the network. For example this could be used to analyze the network when production of a certain metabolite is maximized or to quickly change between different objective functions that are in the model. To do this you will just need to specify a reaction ID in the command and that will be used as the objective function for that simulation. For example if you wanted to analyze the network when the *FRUKIN* reaction is maximized the following command can be used:

```
(psamm-env) $ psamm-model fba --objective=FRUKIN --all-reactions
```

It can be seen from this simulation that the *FRUKIN* reaction is now being used and that the fluxes through the network have changed from when the biomass function was used as the objective function.

```
...
EX_lac_D_e  20.0    |D-Lactate[e]| <=>
EX_manni_e  -10.0   |manni[e]| <=>
EX_o2_e     -5.0    |O2[e]| <=>
EX_pi_e     0.0     |Phosphate[e]| <=>
EX_pyr_e    0.0     |Pyruvate[e]| <=>
EX_succ_e   0.0     |Succinate[e]| <=>
FBA 10.0    |D-Fructose-1-6-bisphosphate[c]| <=> |Dihydroxyacetone-phosphate[c]| +
↪|Glyceraldehyde-3-phosphate[c]|  b2097 or b1773 or b2925
FBP 0.0     |D-Fructose-1-6-bisphosphate[c]| + |H2O[c]| => |D-Fructose-6-
↪phosphate[c]| + |Phosphate[c]|     b3925 or b4232
FORt2       0.0     |Formate[e]| + |H[e]| => |Formate[c]| + |H[c]|  b0904 or b2492
FORti       0.0     |Formate[c]| => |Formate[e]|    b0904 or b2492
FRD7        0.0     |Fumarate[c]| + |Ubiquinol-8[c]| => |Ubiquinone-8[c]| +
↪|Succinate[c]|  b4151 and b4152 and b4153 and b4154
FRUKIN      10.0    |fru[c]| + |ATP[c]| => |D-Fructose-6-phosphate[c]| + |ADP[c]| +
↪|H[c]|
...
```

### Adding new Compounds to the Model

In the previous two steps the reactions and boundary conditions were added into the model. There was no information added in about what the compounds in these reactions actually are but PSAMM is still able to treat them as metabolites in the network and utilize them accordingly. It will be helpful if there is information on these compounds in the model. This will allow you to use the various curation tools and will allow PSAMM to use the new compound names in the output of these various simulations. To add the new compounds to the model a modified `compounds.yaml` file has been provided in the `additional_files` folder. These compounds can be entered into the existing `compounds.yaml` file but for this tutorial the new version can be copied over by running the following command.

```
(psamm-env) $ cp ../additional_files/compounds.yaml .
```

Using the diff command in Git, you will be able to identify changes in the new `compounds.yaml` file:

```
(psamm-env) $ git diff compounds.yaml
```

It can be seen that the new compound entries added to the model were the various new compounds involved in this new pathway.

```
@@ -1,3 +1,12 @@
+- id: fru_c
+  name: Fructose
+  formula: C6H12O6
+- id: manni
+  name: Mannitol
+  formula: C6H14O6
+- id: manni1p
+  name: Mannitol 1-phosphate
+  formula: C6H13O9P
 - id: 13dpg_c
   name: 3-Phospho-D-glyceroyl-phosphate
   formula: C3H4O10P2
```

This will simulate adding in the new compounds to the existing database. Now you can run another FBA simulation to check if these new compound properties are being incorporated into the model. To do this run the following command:

```
(psamm-env) $ psamm-model fba --all-reactions
```

It can be seen that the reactions are no longer represented with compound IDs but are now represented with the compound names. This is because the new compound features are now being added to the model.

```
EX_manni_e  -10.0   |Mannitol[e]| <=>
...
FRUKIN      0.0     |Fructose[c]| + |ATP[c]| => |D-Fructose-6-phosphate[c]| +␣
↪|ADP[c]| + |H[c]|
...
MANNI1PDEH  10.0    |Nicotinamide-adenine-dinucleotide[c]| + |Mannitol 1-
↪phosphate[c]| => |D-Fructose-6-phosphate[c]| + |H[c]| + |Nicotinamide-adenine-
↪dinucleotide-reduced[c]|
MANNI1PPHOS 0.0     |Mannitol 1-phosphate[c]| + |H2O[c]| => |Mannitol[c]| +␣
↪|Phosphate[c]|
MANNIDEH    0.0     |Nicotinamide-adenine-dinucleotide[c]| + |Mannitol[c]| =>␣
↪|Nicotinamide-adenine-dinucleotide-reduced[c]| + |Fructose[c]|
MANNIPTS    10.0    |Mannitol[e]| + |Phosphoenolpyruvate[c]| => |Mannitol 1-
↪phosphate[c]| + |Pyruvate[c]|
```

### Checking File Changes with Git

Now that the model has been updated it will be useful to track the changes that have been made.

First it will be helpful to get a summary of all the files have been modified in the model. Since the changes have been tracked with Git the files that have changed can be viewed by using the following Git command:

```
(psamm-env) $ git status
```

The output of this command should show that the exchange, compound, and `model.yaml` files have changed and that there is a new file that is not being tracked named `mannitol_pathway.yaml`. First the new mannitol pathway file can be added to the Git repository so that future changes can be tracked using the following commands:

```
(psmam-env) $ git add mannitol_pathway.yaml
```

Then specific changes in individual files can be viewed by using the `git diff` command followed by the file name. For example to view the changes in the `compounds.yaml` file the following command can be run.

```
(psamm-env) $ git diff model.yaml
```

The output should look like the following:

```
@@ -5,6 +5,7 @@ compounds:
   - include: compounds.yaml
   reactions:
   - include: reactions.yaml
+  - include: mannitol_pathway.yaml
   exchange:
   - include: exchange.yaml
   limits:
```

This can be done with any file that had changes to make sure that no accidental changes are added in along with whatever the desired changes are. In this example there should be one line added in the `model.yaml` file, three compounds added into the `compounds.yaml` file, and one exchange reaction added into the `exchange.yaml` file along with one change that removed glucose from the list of carbon sources in the exchange settings (by changing the lower bound of its exchange reaction to zero).

Once the changes are confirmed these files can be added with the Git add command.

```
(psamm-env) $ git add compounds.yaml
(psamm-env) $ git add exchange.yaml
(psamm-env) $ git add model.yaml
```

These changes can then be committed to the repository using the following command:

```
(psamm-env) $ git commit -m 'Addition of mannitol utilization pathway and associated␣
↪compounds'
```

Now the model has been updated and the changes have been committed. The Git log command can be used to view what commits have been made in the repository. This allows you to track the overall progress as well as examine what specific changes have been made. More detailed information between the commits can be viewed using the `git diff` command along with the commit ID that you want to compare the current version to. This will tell you specifically what changes occurred between that commit and the current version.

You can also view a log of the commits in the model by using the following command:

```
(psamm-env) $ git log
```

This can be helpful for getting an overall view of what changes have been made to a repository.

The Git version tracking can also be used with GitHub, BitBucket, GitLab or any other Git hosting provider to share repositories with other people. This can enable you to collaborate on different aspects of the modeling process while still tracking the changes made by different groups and maintaining a functional model.

### Using PSAMM to export the model to other Software

If you want to export the model in a format to use with other software, that is also possible using PSAMM. The YAML formatted model can be easily exported as an SBML file using the following command:

```
(psamm-env) $ psamm-model sbmlexport Modified_core_ecoli.xml
```

This will export the model in SBML level 3 version 1 format which can then be used in other software that support this format.

## Model Curation

This tutorial will go over how to utilize the curation functions in PSAMM to correct common errors and ensure that metabolic reconstructions are accurate representations of the metabolism of an organism.


- *Materials*
- *Common Errors in Metabolic Reconstructions*
- *PSAMM Warnings*
- *Reaction Consistency in PSAMM*
- *Gap Identification in PSAMM*
- *Search Functions in PSAMM*
- *Duplicate Reaction Checks*

## Materials

For information on how to install *PSAMM* and the associated requirements, as well how to download the materials required for this tutorial you can reference the Installation and Materials section of the tutorial.

For this part of the tutorial we will be using a modified version of the E. coli core metabolic models that has been used in the other sections of the tutorial. This model has been modified to add in a new pathways for the utilization of mannitol as a carbon source. To access this model and the other files needed you will need to go into the tutorial-part-2 folder located in the psamm-tutorial folder.

```
(psamm-env) $ cd <PATH>/tutorial-part-2/
```

Once in this folder you should see two directories. One is the E_coli_yaml folder which contains the version of the model we will use. The other is called additional_files, which contains some files we will use during the tutorial.

## Common Errors in Metabolic Reconstructions

Many types of errors can be introduced into metabolic models. Some errors can be introduced during manual editing of model files while others can result from inconsistent representations of the biology of the system. Various features in PSAMM are designed ot help identify and fix these problems to ensure that the reconstruction does not contain these kinds of errors.

Some errors cannot be easily identified without extensive manual inspection of the model data files. These PSAMM functions are designed to help identify these errors and make the correction process easier.

## PSAMM Warnings

The most basic way to identify possible errors in a model will be through reading the warning messages printed out by PSAMM when any functions are run on a model. These warning messages can be an easy way to identify if something in the reconstruction is not set up the way that was intended. The following are examples of the types of warnings that PSAMM will provide and what kinds of errors they might indicate.

The first type of warning that PSAMM can provide is a waning that there is a compound that is in a reaction but is not defined in the compound information of the model. While PSAMM doesn't necessarily know if this is an error, these warning can help identify compound ids in the reconstruction that may have typos in them or that need to be defined in the compounds data for the reconstruction. For example in the warning below it would appear that the compound id for ATP had been mistyped and included two extra t's in it. These types of errors can make reactions in a model inconsistent and may lead to incorrect conclusions from the model if they are not corrected.

```
WARNING: The compound cpd_atttp was not defined in the list of compounds
```

The second type of warning will similarly help identify if there was an error introduced in one of the reconstruction's reactions. This warning will indicate that there is a compound present in the reconstruction that has a compartment that is not defined elsewhere in the model. In the example below a compound was added in a reaction as being in the compartment 'X'. Since this compartment was not used in the model the reaction involving this instance of the compound would become flux inconsistent.

```
WARNING: The compartment X was not defined in the list of compartments.
```

The third and fourth types of warnings can be useful in identify that the exchange file is set up correctly for the reconstruction. These two kinds of errors will help identify if there are compounds that are present in the extracellular compartment but do not have a corresponding exchange reaction in the boundary conditions. This can be problematic for some models that require certain sinks for overproduced compounds in the boundary. The other kind of warning will indicate if there are compounds in the exchange reactions that cannot be utilized by any reactions in the model.

This could indicate that a transport reaction is missing from the model or that the compound could be removed from the exchange file.

```
WARNING: The compound cpd_chitob was in the extracellular compartment but not defined␣
↪in the medium
WARNING: The compound cpd_etoh was defined in the medium but is not in the␣
↪extracellular compartment
```

## Reaction Consistency in PSAMM

The previous examples of warning messages produced by PSAMM can be helpful as a first step in identifying possible errors in a model but there are various other types of errors that may be present in models that specific PSAMM functions can help identify. The first kind of errors are ones related to the balancing of reactions in model. It is important that metabolic models be balanced in terms of elements, charge, and stoichiometry. PSAMM has three functions available to identify reactions that are not balanced in these properties which can help correct them and lead to more accurate and true representations of metabolism.

### Stoichiometric Checking

PSAMM's masscheck tool can be used to check if the reactions in the model are stoichiometrically consistent and the compounds that are causing the imbalance. This can be useful when curating the model because it can assist in easily identify missing compounds in reactions. A common problem that can be identified using this tool is a loss of hydrogen atoms during a metabolic reaction. This can occur due to modeling choices or incomplete reaction equations but is generally easy to identify using masscheck.

To report on the compounds that are not balanced use the following masscheck command:

```
(psamm-env) $ psamm-model masscheck
```

This command will produce an output like the following:

```
...
accoa_c      1.0      Acetyl-CoA
acald_e      1.0      Acetaldehyde
acald_c      1.0      Acetaldehyde
h_e 0.0      H
h_c 0.0      H
INFO: Consistent compounds: 73/75
```

The `masscheck` command will first try to assign a positive mass to all of the compounds in the model while balancing the masses such that the left-hand side and right-hand side add up in every model reaction. All the compound masses are reported, and the compounds that have been assigned a zero value for the mass are the ones causing imbalances.

In certain cases a metabolic model can contain compounds that represent electrons, photons, or some other artificial compound. These compounds can cause problems with the stoichiometric balance of a reaction because of their unique functions. In order to deal with this an additional property can be added to the compound entry that will designate it as a compound with zero mass. This designation will tell PSAMM to consider these compounds to have no mass during the stoichiometric checking which will prevent them from causing imbalances in the reactions. An example of how to add that property to a compound entry can be seen below:

```
- id: phot
  name: Photon
  zeromass: yes
```

To report on the specific reactions that may be causing the imbalance, the following command can be used:

```
(psamm-env) $ psamm-model masscheck --type=reaction
...
FRUKIN      1.0     |Fructose[c]| + |ATP[c]| => |D-Fructose-6-phosphate[c]| +␣
↪|ADP[c]| + |H[c]|
INFO: Consistent reactions: 100/101
```

This check is performed similarly to the compound check. In addition, mass residual values are introduced for each metabolic reaction in the network. These mass residuals are then minimized and any reactions that result in a non-zero mass residual value after minimization are reported as being stoichiometrically inconsistent. A non-zero residual value after minimization tells you that the reaction in question may be unbalanced and missing some mass from it.

Sometimes the residue minimization problem may have multiple solutions. In these cases the residue value may be reallocated among a few connected reactions. In this example the unbalanced reaction is the MANNIDEH reaction:

```
MANNIDEH    |manni[c]| + |nad[c]| => |fru[c]| + |nadh[c]|
```

In this reaction equation the right hand side is missing a proton. However minimization problem can result in the residue being placed on either the *fru_c* or the *nadh_c* compounds in an attempt to balance the reaction. Because *nadh_c* occurs in thirteen other reactions in the network, the program has already determined that that compound is stoichiometrically consistent. On the other hand *fru_c* only occurs one other time. Since this compound is less connected the minimization problem will assign the non-zero residual to this compound. This process results in the FRUKIN reaction which contains this compound as being identified as being stoichiometrically inconsistent.

In these cases you will need to manually check the reaction and then use the `--checked` option for the `masscheck` command to force the non-zero residual to be placed on a different reaction. This will rerun the consistency check and force the residual to be placed on a different reaction. To do this we would run the following command.

```
(psamm-env) $ psamm-model masscheck --type=reaction --checked FRUKIN
```

Now, the output should report the *MANNIDEH* reaction and it can be seen that the reaction equation of *MANNIDEH* is specified incorrectly. It appears that a hydrogen compound was left out of the reaction for *MANNIDEH*. This would be an easy problem to correct by simply adding in a hydrogen compound to correct the lost atom in the equation.

The stoichiometric consistency checking allows for the easy identification of stoichiometrically inconsistent compounds while providing a more targeted subset of reactions to check to fix the problem. This allows you to quickly identify problematic reactions rather than having to manually go through the whole reaction database in an attempt to find the problem.

In some cases there are reactions that are going to be inherently unbalanced and might cause problems with using these methods. If you know that this is the case for a specific reaction they can specify that the reaction be excluded from the mass check so that the rest of the network can be analyzed. To do this the `--exclude` option can be used. For example if you wanted to exclude the reaction *FRUKIN* from the mass check they could use the following command:

```
(psamm-env) $ psamm-model masscheck --exclude FRUKIN
```

This exclude option can be helpful in removing inherently unbalanced reactions like macromolecule synthesis reations or incomplete reactions that would be identified as being stoichiometrically inconsistent. It is also possible to create a file that lists multiple reactions to exclude. Put each reaction identifier on a separate line in the file and refer to the file be prefixing the file name with a `@`:

```
(psamm-env) $ psamm-model masscheck --exclude @excluded_reactions.txt
```

Before we fix the model with the correction to the *MANNIDEH* reaction, let us first check the model for formula inconsistencies to show how this can also be used in conjunction with mass checking and other methods to correct model inconsistencies.

### Formula Consistency Checking

Formula checking will check that each reaction in the model is balanced with respect to the chemical formulas of each compound. To check the model for formula consistencies run the formula check command:

```
(psamm-env) $ psamm-model formulacheck
```

The output should appear as follows:

```
INFO: Model: Ecoli_core_model
INFO: Model Git version: 9812080
MANNIDEH    C27H40N7O20P2   C27H39N7O20P2           H
Biomass_Ecoli_core_w_GAM    C1088.0232H1471.1810N446.7617O1236.7018P240.5298S3.7478␣
↪C1045.4677H1395.2089N441.3089O1189.0281P236.8511S3.7478         C42.5555H75.9721N5.
↪4528O47.6737P3.6787
INFO: Unbalanced reactions: 2/80
INFO: Unchecked reactions due to missing formula: 0/80
```

In this case two reactions are identified in the model as being unbalanced. The biomass objective function, *Biomass_Ecoli_core_w_GAM*, and the reaction that was previously identified through masscheck as being unbalanced, *MANNIDEH*. In the case of the objective function this is imbalanced due to the formulation of the objective function. The reaction functions as a sink for the compounds required for growth and only outputs depleted energy compounds. This leads to it being inherently formula imbalanced but it is a necessary feature of the model. The other reaction is *MANNIDEH*. It can be seen that the total number of atoms on each side does not match up. PSAMM also outputs what atoms would be needed to balance the reaction on both sides. In this case there is a missing hydrogen atom on the right side of the equation. This can be easily rectified by adding in the missing hydrogen. To do this correction in this tutorial, you can copy a fixed version of the mannitol pathway from the additional files folder using the following command:

```
(psamm-env) $ cp ../additional_files/mannitol_pathway_v2.yaml mannitol_pathway.yaml
```

Once that problem with the new reaction is fixed the model will pass both the formula check and mass check.

### Charge Consistency Checking

The charge consistency function is similar to the formula consistency function but instead of using the chemical formulas for the compounds, PSAMM will use the assigned charges that are designated in the compounds file and check that these charges are balanced on both sides of the reaction.

To run a charge consistency check on the model use the chargecheck command:

```
(psamm-env) $ psamm-model chargecheck
```

This *E. coli* SBML model does not contain charge information for the compounds. A sample output is provided below to show what the results would look like for a charge imbalanced model. The output from the charge check will display any reactions that are charge imbalanced and show what the imbalance is and then show the reaction equation. This can be used to quickly check for any missed inconsistencies and identify reactions and compounds that should be looked at more closely to confirm their correctness.

```
...
rxn12510    1.0     |ATP[c]| + |Pantothenate[c]| => |4-phosphopantothenate[c]| +␣
↪|H+[c]| + |ADP[c]|
rxn12825    4.0     |hemeO[c]| + |H2O[c]| => |Heme[c]| + (4) |H+[c]|
rxn13643    1.0     |ADP-glucose[c]| => |Glycogen[c]| + |H+[c]| + |ADP[c]|
rxn13710    6.0     (5) |D-Glucose[c]| + (4) |ATP[c]| => |Glycogen[c]| + (4) |H+[c]|␣
↪+ (4) |Phosphate[c]| + (4) |H2O[c]| + |ADP[c]|
```

```
INFO: Unbalanced reactions: 94/1093
INFO: Unchecked reactions due to missing charge: 0/1093
```

### Flux Consistency Checking

The flux consistency checking function can be used to identify reactions that cannot carry flux in the model. This tool can be used as a curation tool as well as an analysis tool. In this tutorial it will be highlighted for the curation aspects and later its use in flux analysis will be demonstrated.

To run a flux consistency check on the model use the `fluxcheck` command:

```
(psamm-env) $ psamm-model fluxcheck --unrestricted
```

The unrestricted option with the command will tell PSAMM to remove any limits on the exchange reactions. This will tell you which reactions in the model can carry flux if the model is given all compounds in the media freely. This can be helpful for identifying which reactions may not be linked to other parts of the metabolism and can be helpful in identifying gaps in the model. In this case it can be seen that no reactions were identified as being inconsistent.

In some situations there are pathways that might be modeled but not necessarily connected to the other aspects of metabolism. A common occurrence of this is with vitamin biosynthesis pathways that are not incorporated into the biomass in the model. `fluxcheck` will identify these as being flux inconsistent but the modeler will need to identify if this is due to incomplete information on the pathways or if it is due to some error in the formulation of the reactions.

PSAMM will tell you how many exchange reactions cannot be used as well as how many internal model reactions cannot carry flux. PSAMM will also list the reactions and the equations for the reactions to make curation of these reactions easier.

Above the `fluxcheck` command was used with the –unrestricted option which allowed the exchange reactions to all be active. This command can also be used to see what reactions cannot carry flux when specific media are supplied. To run this command on the network with the media that is specified in the media file run the following command:

```
(psamm-env) $ psamm-model fluxcheck
INFO: Model: Ecoli_core_model
INFO: Model Git version: 9812080
INFO: Using flux bounds to determine consistency.
...
EX_fru_e     |D-Fructose[e]| <=>
EX_fum_e     |Fumarate[e]| <=>
EX_glc_e     |D-Glucose[e]| <=>
EX_gln_L_e   |L-Glutamine[e]| <=>
EX_mal_L_e   |L-Malate[e]| <=>
FRUpts2      |D-Fructose[e]| + |Phosphoenolpyruvate[c]| => |D-Fructose-6-phosphate[c]|␣
↪+ |Pyruvate[c]|
FUMt2_2      (2) |H[e]| + |Fumarate[e]| => (2) |H[c]| + |Fumarate[c]|
GLCpts       |Phosphoenolpyruvate[c]| + |D-Glucose[e]| => |Pyruvate[c]| + |D-Glucose-6-
↪phosphate[c]|
GLNabc       |ATP[c]| + |L-Glutamine[e]| + |H2O[c]| => |L-Glutamine[c]| + |ADP[c]| +␣
↪|H[c]| + |Phosphate[c]|
MALt2_2      |L-Malate[e]| + (2) |H[e]| => |L-Malate[c]| + (2) |H[c]|
INFO: Model has 5/80 inconsistent internal reactions (0 disabled by user)
INFO: Model has 5/21 inconsistent exchange reactions (0 disabled by user)
```

In this case it can be seen that there are various exchange reactions blocked as well as various internal reactions related to other carbon metabolic pathways. The current model should only be supplying mannitol as a carbon source and this would mean that these other carbon pathways would be blocked in this condition. In this way, you can use the `fluxcheck` command to see what reactions are specific to certain metabolic pathways and environmental conditions.

## Gap Identification in PSAMM

In addition to inconsistencies found within individual reactions there can also be global inconsistencies for the reactions within a metabolic network. These include metabolites that can be produced but not consumed, ones that can be consumed by reactions but are not produced, and reactions that cannot carry flux in a model. PSAMM includes various functions for the identification of these features in a network including the functions `gapcheck` and `fluxcheck`. Additionally the functions `gapfill` and `fastgapfill` can be used to help fill these gaps that are present through the introduction of additional reactions into the network.

### Gapcheck in PSAMM

The `gapcheck` function in *PSAMM* can be used to identify dead end metabolites in a metabolic network. These dead end metabolites are compounds in the metabolic model that can either be produced but not consumed or ones that can be consumed but not produced. Reactions that contain these compounds cannot carry flux within a model and are often the result of knowledge gaps in our understanding of metabolic networks.

The `gapcheck` function allows the use of three methods for the identification of these dead end metabolites within a metabolic network. These are the `prodcheck`, `sinkcheck`, and `gapfind` methods.

The `prodcheck` method is the most straightforward of these methods and can be used to identify any compounds that cannot be produced in the metabolic network. It will iterate through the reactions in a network and maximize each one. If the reaction can carry a flux then the metabolites involved in the reaction are not considered to be blocked.

To use this function the following command can be run:

```
(psamm-env) $ psamm-model gapcheck --method prodcheck
```

The function will produce output like the following that lists out any metabolites in the model that cannot be produced in this condition:

```
fru[e]      D-Fructose
fum[e]      Fumarate
glc_D[e]    D-Glucose
gln_L[e]    L-Glutamine
mal_L[e]    L-Malate
INFO: Blocked compounds: 5
```

This result indicates that the following metabolites currently cannot be produced in the model. This only tells part of the story though, as this function was run with the defined media that was set for the model. As a result there are gaps identified like, 'D-Glucose', that will not be considered gaps in other conditions. To do a global check using this function on the model without restrictions on the media the following command can be used:

```
(psamm-env) $ psamm-model gapcheck --method prodcheck --unrestricted-exchange
```

The unrestricted tag in this function will temporarily set all of the exchange reaction bounds to be -1000 to 1000 allowing all nutrients to be either taken up or produced. Gap-checking in this condition will allow for the identification of gaps that are not media dependent and may instead be the result of incomplete pathways and knowledge gaps.

The second method implemented in the `gapcheck` function is the `sinkcheck` method. This method is similar to `prodcheck` but is implemented in a way where the flux through each introduced sink for a compound is maximized. This ensures that the metabolite can be produced in excess from the network for it to not be considered a dead end metabolite.

```
(psamm-env) $ psamm-model gapcheck --method sinkcheck --unrestricted-exchange
```

The last method implemented in the `gapcheck` function is the `gapfind` method. This method is an implementation of a previously published method to identify gaps in metabolic networks *[Kumar07]*. This method will use a network based optimization to identify metabolites with no production pathways present.

```
(psamm-env) $ psamm-model gapcheck --method gapfind --unrestricted-exchange
```

These methods included in the `gapcheck` function can be used to identify various kinds of 'gaps' in a metabolic model network. *PSAMM* also includes two functions for filling these gaps through the addition of artificial reactions or reactions from a supplied database. The functions `gapfill` and `fastgapfill` can be used to perform these gapfilling procedures during the process of generating and curating a model.

## Search Functions in PSAMM

`psamm-model` includes a search function that can be used to search the model information for specific compounds or reactions. To do this the search function can be used. This can be used for various search methods. For example to search for the compound named fructose the following command can be used:

```
(psamm-env) $ psamm-model search compound --name 'Fructose'
INFO: Model: Ecoli_core_model
INFO: Model Git version: db22229
id: fru_c
formula: C6H12O6
name: Fructose
Defined in ./compounds.yaml:?:?
```

To do the same search but instead use the compound ID the following command can be used:

```
(psamm-env) $ psamm-model search compound --id 'fru_c'
```

These searches will result in a printout of the relevant information contained within the model about these compounds. In a similar way reactions can also be searched. For example to search for a reaction by a specific ID the following command can be used:

```
(psamm-env) $ psamm-model search reaction --id 'FRUKIN'
```

Or to search for all reactions that include a specific compound the following command can be used:

```
(psamm-env) $ psamm-model search reaction --compound 'manni[c]'
```

## Duplicate Reaction Checks

An additional searching function called `duplicatescheck` is also included in *PSAMM*. This function will search through a model and compare all of the reactions in the network to each other. Any reactions that have all of the same metabolites consumed and produced will then be reported. This can be a helpful function to use if there a multiple people working on the construction of a model as it allows for an automated checking that two individuals did not add the same reaction to the reconstruction. The `duplicatescheck` function can be run through the following command:

```
(psamm-env) $ psamm-model duplicatescheck
```

The additional tags `--compare-direction` and `--compare-stoichiometry` can be added to the command to take into account the reaction directionality and metabolite stoichiometry when comparing two different reactions.

# Constraint Based Analysis with PSAMM

This tutorial will go over how to use the constraint based analysis methods that are included in PSAMM. These methods can be used to perform various simulations of growth with metabolic models. These simulations can be used to explore growth phenotypes, nutrient utilization, and gene essentiality.


- *Tutorial Materials*
- *Constraint-based Flux Analysis with PSAMM*
- *FBA in PSAMM*


## Tutorial Materials

The materials used in the part of the tutorial can be found in the *tutorial-part-3* directory in the psamm-tutorial repository. This directory contains a copy of the E. coli core metabolic model that has been used in the other tutorials. This model can be used to run all of the simulations in this part of the tutorial. In addition to the model the virtual environment where PSAMM has been installed will need to be activated to run the *psamm-model* commands. For instructions on how to install or activate *PSAMM* in a virtual environment reference the Installation and Materials section of the tutorial.

To access the materials needed to run the following commands go to the E_coli_yaml folder in the tutorial-part-3 folder.

```
(psamm-env) $ cd <PATH>/tutorial-part-3/E_coli_yaml
```

## Constraint-based Flux Analysis with PSAMM

Along with the various curation tools that are included with PSAMM there are also various flux analysis tools that can be used to perform simulations on the model. This allows for a seamless integration of the model development, curation, and simulation processes.

There are various options that you can change in these different flux analysis commands. Before introducing the specific commands these options will be detailed here.

### Loop Minimization in PSAMM

First, you can choose the options for loop minimization when running constraint-based analyses. This can be done by using the `--loop-removal` option. There are three options for loop removal when performing constraint based analysis:

***none*** No removal of loops

***tfba*** Removes loops by applying thermodynamic constraints

***l1min*** Removes loops by minimizing the L1 norm (the sum of absolute flux values)

For example, you could run flux balance analysis with thermodynamic constraints:

```
(psamm-env) $ psamm-model fba --loop-removal=tfba
```

or without:

```
(psamm-env) $ psamm-model fba --loop-removal=none
```

### Choosing Linear Programming Solvers

You also have the option to set which solver you want to use for the linear programming problems. To view the solvers that are currently installed the following command can be used:

```
(psamm-env) $ psamm-list-lpsolvers
```

By default PSAMM will use CPLEX if it available but if you want to specify a different solver you can do so using the `--solver` option. For example to select the Gurobi solver during an FBA simulation you can use the following command:

```
(psamm-env) $ psamm-model fba --solver name=gurobi
```

If multiple solvers are installed and you do not want to use the default solver, you will need to set this option for every simulation run.

**Note:** The QSopt_ex solver does not support integer linear programming problems. This solver can be used with any commands but you will not be able to run the simulation with thermodynamic constraints.

### Other Global Options

Another option that can be used with the various flux analysis commands is the `--epsilon` option. This option can be used to set the minimum value that a flux needs to be above to be considered non-zero. By default PSAMM will consider any number above $10^{-5}$ to be non-zero. An example of changing the epsilon value with this option during an FBA simulation is:

```
(psamm-env) $ psamm-model fba --epsilon 0.0001
```

These various options can be used for any of the flux analysis functions in PSAMM by adding them to the command that is being run. A list of the functions available in PSAMM can be viewed by using the command:

```
(psamm-env) $ psamm-model --help
```

The options for a specific function can be viewed by using the command:

```
(psamm-env) $ psamm-model <command> --help
```

## FBA in PSAMM

PSAMM allows for the integration of the model development and curation process with the simulation process. In this way changes to a metabolic model can be immediately tested using the various flux analysis tools that are present in PSAMM. In this tutorial, aspects of the *E. coli* core model *[Orth11]* will be expanded to demonstrate the various functions available in PSAMM and throughout these changes the model will be analyzed with PSAMM's simulation functions to make sure that these changes are resulting in a functional model.

### Flux Balance Analysis

Flux Balance Analysis (FBA) is one of the basic methods that allows you to quickly examine if the model is viable (i.e. can produce biomass). PSAMM provides the `fba` function in the `psamm-model` command to perform FBA on metabolic models. For example, to run FBA on the *E. coli* core model first make sure that the current directory is the `E_coli_yaml/` directory using the following command:

```
(psamm-env) $ cd <PATH>/psamm-tutorial/E_coli_yaml/
```

Then run FBA on the model with the following command.

```
(psamm-env) $ psamm-model fba
```

Note that the command above should be executed within the folder that stores the `model.yaml` file. Alternatively, you could run the following command anywhere in your file system:

```
(psamm-env) $ psamm-model --model <PATH-TO-MODEL.YAML> fba
```

The following is a sample of some output from the FBA command:

```
INFO: Model: Ecoli_core_model
INFO: Model Git version: 9812080
INFO: Using Biomass_Ecoli_core_w_GAM as objective
INFO: Loop removal disabled; spurious loops are allowed
INFO: Setting feasibility tolerance to 1e-09
INFO: Setting optimality tolerance to 1e-09
INFO: Solving took 0.05 seconds
ACONTa      6.00724957535   |Citrate[c]| <=> |cis-Aconitate[c]| + |H2O[c]|  b0118 or␣
↪b1276
ACONTb      6.00724957535   |cis-Aconitate[c]| + |H2O[c]| <=> |Isocitrate[c]|      ␣
↪b0118 or b1276
AKGDH       5.06437566148   |2-Oxoglutarate[c]| + |Coenzyme-A[c]|...
...
INFO: Objective flux: 0.873921506968
INFO: Reactions at zero flux: 47/95
```

At the beginning of the output of `psamm-model` commands information about the model as well as information about simulation settings will be printed. At the end of the output PSAMM will print the maximized flux of the designated objective function. The rest of the output is a list of the reaction IDs in the model along with their fluxes, and the reaction equations represented with the compound names. This output is human readable because the reactions equations are represented with the full names of compound. It can be saved as a tab separated file that can be sorted and analyzed quickly allowing for easy analysis and comparison between FBA in different conditions.

By default, PSAMM fba will use the biomass function designated in the central model file as the objective function. If the biomass tag is not defined in a `model.yaml` file or if you want to use a different reaction as the objective function, you can simply specify it using the `--objective` option. For example to maximize the citrate synthase reactions, *CS*, the command would be as follows:

```
(psamm-env) $ psamm-model fba --objective=CS
```

Flux balance analysis will be used throughout this tutorial as both a checking tool during model curation and an analysis tool. PSAMM allows you to easily integrate analysis tools like this into the various steps during model development.

### Flux Variability Analysis

Another flux analysis tool that can be used in PSAMM is flux variability analysis. This analysis will maximize the objective function that is designated and provide a lower and upper bound of the various reactions in the model that would still allow the model to sustain the same objective function flux. This can provide insights into alternative pathways in the model and allow the identification of reactions that can vary in use.

To run FVA on the model use the following command:

```
(psamm-env) $ psamm-model fva
...
EX_pi_e     -3.44906664664  -3.44906664664  |Phosphate[e]| <=>
EX_pyr_e    -0.0    -0.0    |Pyruvate[e]| <=>
EX_succ_e   -0.0    -0.0    |Succinate[e]| <=>
FBA 7.00227721609   7.00227721609   |D-Fructose-1-6-bisphosphate[c]| <=>␣
↪|Dihydroxyacetone-phosphate[c]| + |Glyceraldehyde-3-phosphate[c]|
FBP 0.0     0.0     |D-Fructose-1-6-bisphosphate[c]| + |H2O[c]| => |D-Fructose-6-
↪phosphate[c]| + |Phosphate[c]|
FORt2       0.0     0.0     |Formate[e]| + |H[e]| => |Formate[c]| + |H[c]|
...
```

The output shows the reaction IDs in the first column and then shows the lower bound of the flux, the upper bound of the flux, and the reaction equations. With the current conditions the flux is not variable through the equations in the model. It can be seen that the upper and lower bounds of each reaction are the same. If another carbon source was added in though it would allow for more reactions to be variable. For example if glucose was added into the media along with mannitol then the results might appear as follows:

```
EX_glc_e    -10.0   -2.0    |D-Glucose[e]| <=>
EX_manni_e  -9.0    -3.0    |Mannitol[e]| <=>
MANNIPTS    3.0     9.0     |Mannitol[e]| + |Phosphoenolpyruvate[c]| => |Mannitol 1-
↪phosphate[c]| + |Pyruvate[c]|
GLCpts      2.0     10.0    |D-Glucose[e]| + |Phosphoenolpyruvate[c]| =>␣
↪|Pyruvate[c]| + |D-Glucose-6-phosphate[c]|
```

It can be seen that in this situation the lower and upper bounds of some reactions are different indicating that their flux can be variable. This indicates that there is some variability in the model as to how certain reactions can be used while still maintaining the same objective function flux.

### Robustness Analysis

Robustness analysis can be used to analyze the model under varying conditions. Robustness analysis will maximize a designated reaction while varying another designated reaction. For example, you could vary the amount of oxygen present while trying to maximize the biomass production to see how the model responds to different oxygen supply. You can specify the number of steps that will be performed in the robustness as well as the reaction that will be varied during the steps.

By default, the reaction that is maximized will be the biomass reaction defined in the `model.yaml` file but a different reaction can be designated with the optional `--objective` option. The flux bounds of this reaction will then be obtained to determine the lower and upper value for the robustness analysis. These values will then be used as the starting and stopping points for the robustness analysis. You can also set a customized upper and lower flux value of the varying reaction using the `--lower` and `--upper` options.

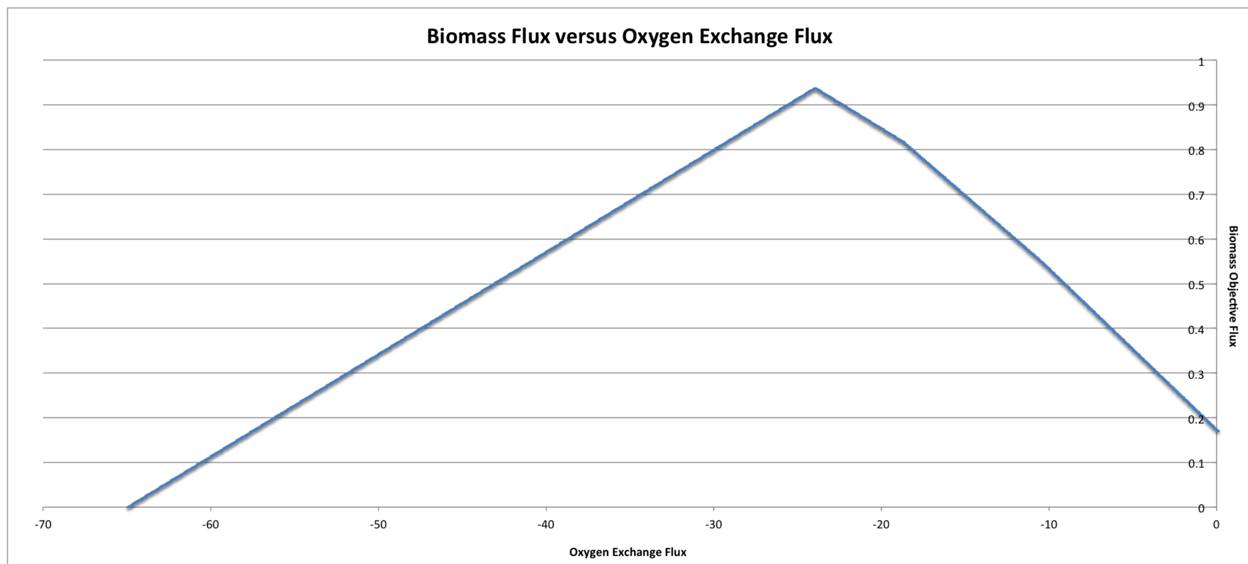
For this model the robustness command will be used to see how the model responds to various oxygen conditions with mannitol as the supplied carbon source. To run the robustness command use the following command:

```
(psamm-env) $ psamm-model robustness --steps 1000 EX_o2_e
```

The output will contain two columns. The first column will be the flux of the varied reaction, in this case the EX_o2_e reaction for oxygen exchange. The second shows the flux of the biomass reaction for the model. The output will look like this:

```
-63.958958959       0.0238161275506
-63.8938938939      0.0253046355225
-63.8288288288      0.0267931434944
-63.7637637638      0.0282816514663
-63.6986986987      0.0297701594383
-63.6336336336      0.0312586674102
-63.5685685686      0.0327471753821
-63.5035035035      0.034235683354
-63.4384384384      0.0357241913259
...
```

If the biomass reaction flux is plotted against the oxygen uptake it can be seen that the biomass flux is low at the highest oxygen uptake, reaches a maximum at an oxygen uptake of about 24, and then starts to decrease with low oxygen uptake.




If a more detailed analysis of internal fluxes is desired the *–all-reaction-fluxes* tag can be added to the command. This will print out all of the internal reaction fluxes for each step in the robustness analysis. The first column printed will be the reaction ID. The second column will be the varying reaction's flux and the last column will be the flux of the reaction listed in the first column. This can be used to look at the effects of a reaction on internal fluxes in the network. The command to run this would be the following:

```
(psamm-env) $ psamm-model robustness --all-reaction-fluxes --steps 1000 EX_o2_e
```

And the output for this command will look like the following:

```
G6PDH2r     -63.958958959   0.0
AKGDH       -63.958958959   0.0
GLNS        -63.958958959   0.00608978381469
ADK1        -63.958958959   0.0
PYRt2r      -63.958958959   0.0
EX_co2_e    -63.958958959   58.986492784
ATPM        -63.958958959   8.39
SUCCt2_2    -63.958958959   0.0
PIt2r       -63.958958959   0.0876123884204
```

```
EX_lac_D_e  -63.958958959   0.0
```

## Deletion Simulations with PSAMM

### Gene Deletion

The `genedelete` command can be used to perform gene deletions in a model and test what effects those deletions have. This command can be used to quickly test if certain genes are essential in the network. The command will take a list of genes in a separate file and will then go through all of the gene associations in the model to determine what reactions require that gene to be present. This uses the gene association logic to determine if the removal of the specified genes would knock out that function. For example if we had the following two reactions:

```
- id: RXN_1
  genes: g0001 and g0002
  equation: '|cpd_a[c]| <=> |cpd_b[c]|'

- id: RXN_2
  genes: g0001 or g0003
  equation: '|cpd_a[c]| <=> |cpd[c]|'
```

Both reactions are associated with the gene 'g0001' but RXN_1 has an 'and' association while RXN_2 has an 'or' association. If the gene 'g0001' were to be deleted from the network RXN_1 would no longer have the required genes for it to be present since both genes are required. RXN_2 would still be satisfied since it would only require one of the two genes to be present. The gene delete command will do this automatically and for the entire network making it much easier to do these kinds of simulations. The gene delete command can be run with the following command.

```
(psamm-env) $ psamm-model genedelete --gene b1779
```

This will produce a flux balance analysis result with a model that has any reactions for which b0118 is necessary limited to zero flux. The output will show a percentage of the biomass flux of the wild type model that can be produced by the deletion model.

```
...
INFO: Objective reaction after gene deletion has flux 0.0
INFO: Objective reaction has 0.00% flux of wild type flux
```

### Random Minimal Network Analysis

The `randomsparse` command can be used to look at gene essentiality in the metabolic network. To use this function the model must contain gene associations for the model reactions. This function works by systematically deleting genes from the network, then evaluating if the associated reaction would still be available after the gene deletion, and finally testing the new network to see if the objective function flux is still above the threshold for viability. If the flux falls too low then the gene is marked as essential and kept in the network. If the flux stays above the threshold then the gene will be marked as non-essential and removed. The program will randomly do this for all genes until the only ones left are marked as essential. This can be done using the `--type=genes` option with the `randomsparse` command:

```
(psamm-env) $ psamm-model randomsparse --type=genes 90%
```

This will produce an output of the gene IDs with a 1 if the gene was kept in the simulation and a 0 if the gene was deleted. Following the list of genes will be a summary of how many genes were kept out of the total as well as list of the reaction IDs that made up the minimal network for that simulation. An example output can be seen as follows:

```
INFO: Essential genes: 58/137
INFO: Deleted genes: 79/137
b0008       0
b0114       1
b0115       1
b0116       1
b0118       0
b0351       0
b0356       0
b0451       0
b0474       0
b0485       0
...
```

The random minimal network analysis can also be used to generate a random subset of reactions from the model that will still allow the model to maintain an objective function flux above a user-defined threshold. This function works on the same principle as the gene deletions but instead of removing individual genes, reactions will be removed. To run random minimal network analysis on the model use the randomsparse command with the `--type=reactions` option. The last parameter for the command is a percentage of the maximum objective flux that will be used as the threshold for the simulation.

```
(psamm-env) $ psamm-model randomsparse --type=reactions 95%
...
FRUKIN      1
...
MANNI1PDEH  0
MANNI1PPHOS 1
MANNIDEH    1
MANNIPTS    1
...
```

The output will be a list of reaction IDs with either a 1 indicating that the reaction was essential or a zero indicating it was removed.

Due to the random order of deletions during this simulation it may be helpful to run this command numerous times in order to gain a statistically significant number of datapoints from which a minimal essential network of reactions can be established.

In this case the program deleted the *MANN1PDEH* reaction blocking the mannitol 1-phosphate to fructose 6-phosphate conversion. In this case the reactions in the other side of the mannitol utilization pathway should all be essential.

You can also use the `randomsparse` command to randomly sample the exchange reactions and generate putative minimal exchange reaction sets. This can be done by using the `--type=exchange` option with the `randomsparse` command:

```
(psamm-env) $ psamm-model randomsparse --type=exchange 90%
```

It can be seen that when this is run on this small network the mannitol exchange as well as some other small molecules are identified as being essential to the network:

```
EX_ac_e     0
EX_acald_e  0
EX_akg_e    0
EX_co2_e    1
EX_etoh_e   0
EX_for_e    0
EX_fru_e    0
EX_fum_e    0
```

```
EX_glc_e    0
EX_gln_L_e  0
EX_glu_L_e  0
EX_h2o_e    1
EX_h_e      1
EX_lac_D_e  0
EX_mal_L_e  0
EX_manni_e  1
EX_nh4_e    1
EX_o2_e     1
EX_pi_e     1
EX_pyr_e    0
EX_succ_e   0
```

# CHAPTER 3

# Install

PSAMM can be installed using the Python package installer `pip`. We recommend that you use a Virtualenv when installing PSAMM. First, create a Virtualenv in your project directory and activate the environment. On Linux/OSX the following terminal commands can be used:

```
$ virtualenv env
$ source env/bin/activate
```

Then, install PSAMM using the `pip` command:

```
(env) $ pip install psamm
```

When returning to the project from a new terminal window, simply reactivate the environment by running

```
$ source env/bin/activate
```

The *psamm-import* tool is developed in a separate Git repository. After installing PSAMM, the *psamm-import* tool can be installed using:

```
(env) $ pip install git+https://github.com/zhanglab/psamm-import.git
```

## Dependencies

- Linear programming solver (*Cplex*, *Gurobi*, *GLPK* or *QSopt_ex*)
- PyYAML (for reading the native model format)
- NumPy (optional; model matrix can be exported to NumPy matrix if available)

PyYAML is installed automatically when PSAMM is installed through `pip`. The linear programming solver is not strictly required but most analyses require one to work. The LP solver *Cplex* is the preferred solver. We recently added support for the LP solver *Gurobi* and *GLPK*.

The rational solver *QSopt_ex* does not support MILP problems which means that some analyses require one of the other solvers. The MILP support in *GLPK* is still experimental so it is disabled by default.

## Cplex

The Cplex Python bindings will have to be installed manually. Make sure that you are using at least **Cplex version 12.6**. If you are using a virtual environment (as described above) this should be done after activating the virtual environment:

1. Locate the directory where Cplex was installed (e.g. `/path/to/IBM/ILOG/CPLEX_StudioXXX`).
2. Locate the appropriate subdirectory based on your platform and Python version: `cplex/python/<version>/<platform>` (e.g. `cplex/python/2.7/x86-64_osx`).
3. Use `pip` to install the package from this directory using the following command.

```
(env) $ pip install \
    /path/to/IBM/ILOG/CPLEX_Studio1262/cplex/python/<version>/<platform>
```

Further documentation on installing Cplex can be found in the Cplex documentation.

**Note:** Python 3 support was added in a recent release of Cplex. Older versions only support Python 2. If you are using Python 3 make sure that you have the latest version of Cplex installed.

## Gurobi

The Gurobi Python bindings will have to be installed into the virtualenv. After activating the virtualenv:

1. Locate the directory where the Guropi python bindings were installed. For example, on OSX this directory is `/Library/gurobiXXX/mac64` where `XXX` is a version code.
2. Use `pip` to install the package from this directory. For example:

```
(env) $ pip install /Library/gurobi604/mac64
```

## GLPK

The GLPK solver requires the GLPK library to be installed. The `swiglpk` Python bindings are required for PSAMM to use the GLPK library.

```
(env) $ pip install swiglpk
```

## QSopt_ex

QSopt_ex is supported through python-qsoptex which requires GnuMP and the QSopt_ex library. After installing these libraries the Python bindings can be installed using `pip`:

```
(env) $ pip install python-qsoptex
```

CHAPTER 4

# Model file format

The primary model definition file is the `model.yaml` file. When creating a new model this file should be placed in a new directory. The following can be used as a template:

```
---
name: Escherichia coli test model
biomass: Biomass
extracellular: e

compartments:
  - id: e
    name: Extracellular
  - id: p
    name: Periplasm
    adjacent_to: [e, c]
  - id: c
    name: Cytosol
    adjacent_to: p

compounds:
  - include: ../path/to/ModelSEED_cpds.tsv
    format: modelseed

reactions:
  - include: reactions/reactions.tsv
  - include: reactions/biomass.yaml

exchange:
  - include: exchange.yaml
limits:
  - include: limits.yaml

model:
  - include: model_def.tsv
```

## Biomass

The optional `biomass` key specifies the default reaction to use for various analyses (e.g. FBA, FVA, etc.)

## Extracellular Compartment

The optional `extracellular` key specifies the default string for the extracellular compartment on compounds. If this option is not specified it will be assumed that the extracellular compartment is called `e`.

## Default Compartment

The optional `default_compartment` key specifies the default compartment that is used if a compound in a reaction does not explicitly specify a compartment. For example, the reaction `|x[e]| + |atp| => |x| + |adp| + |pi|` does not specify a compartment on four of the compounds so those four would automatically be presumed to be in the default compartment (or `c` if no default compartment is specified).

## Compartments

The `compartments` key is a list of compartment information for the model. Compartments must always have an `id` but can also have additional user defined properties. The `adjacent_to` property is used to define the boundaries between compartments. Notice that the adjacency can be specified as a single compartment or a list of compartments. Note that it is sufficient to specify that `p` is adjacent to `e`. It is then inferred that `e` is adjacent to `p` so it is optional to specify both directions of adjacency.

## Compounds

The optional `compounds` key is a list of compound information. For some of the model checks the compound information is required. This section can also include external files that contain compound information. If the file is a ModelSEED compound table, the `format` key must be set to `modelseed`. If the file is a YAML file, the file should have a `.yaml` extension. The following fragment is an example of a YAML formatted compound file:

```
- id: ac
  name: Acetate
  formula: C2H3O2
  charge: -1

- id: acac
  name: Acetoacetate
  # ...
```

The following compound properties are recognized:

| Property | Type | Description |
|---|---|---|
| id | string | Compound ID (*required*) |
| name | string | Name of compound |
| formula | string | Compound formula (e.g. C6H12O6) |
| charge | integer | Formal charge |
| kegg | string | KEGG ID (reference to compound in KEGG database) |
| cas | string | CAS number |

## Reactions

The key `reactions` specifies a list of files that will be used to define the reactions in the model. The reaction files can be formatted as either tab-separated (`.tsv`) or YAML files (`.yaml`). The TSV file may be adequate for most of the reaction definitions while certain particularly complex reactions (e.g. biomass reaction) may be specified using a YAML file.

The TSV format is a tab-separated table where each row contains the reaction ID in addition to other data columns. The header must specify the type of each column. The column `equation` will be parsed as ModelSEED reaction equations.

```
id      equation
ADE2t   |ade[e]| + |h[e]| <=> |ade[c]| + |ade[c]|
ADK1    |amp| + |atp| <=> (2) |adp|
```

Any `.yaml` or `.yml` file in the `reactions` specification will be parsed as a reaction definition file but in YAML format. This format is particularly useful for very long reactions containing many different compounds (e.g. the biomass reaction). It also allows adding more annotations because of the structured nature of the YAML format. The following snippet is an example of a YAML reaction file:

```
# Biomass composition
- id: Biomass
  equation:
    reversible: no
    left:
      - id: cpd00032 # Oxaloacetate
        value: 1
      - id: cpd00022 # Acetyl-CoA
        value: 1
      - id: cpd00035 # L-Alanine
        value: 0.02
      # ...
    right:
      - id: Biomass
        value: 1
      # ...
```

Reactions in YAML files can also be defined using ModelSEED formatted reaction equations. The `|` is a special character in YAML so the reaction equations have to be quoted with `'` or, alternatively, using the > for a multiline quote:

```
- id: ADE2t
  equation: >
    |ade[e]| + |h[e]| <=>
    |ade[c]| + |h[c]|
- id: ADK1
  equation: '|amp| + |atp| <=> (2) |adp|'
```

The following reaction properties are recognized:

| Property | Type | Description |
|---|---|---|
| id | string | Reaction ID (*required*) |
| name | string | Name of reaction |
| equation | string or dict | Reaction equation formula |
| ec | string | EC number |
| genes | string | Gene association rule |

The `genes` property can be used to specifiy which genes enable a reaction. Complex gene association rules can be used when a reaction is enabled by a group of genes or when multiple genes can independently enable a reaction:

```
- id: ADK1
  equation: '|amp| + |atp| <=> (2) |adp|'
  genes: gene_0001 or (gene_0002 and gene_0003)
```

## Exchange compounds

The `exchange` key provides a way of defining the compounds that can enter and exit the model system (the boundary conditions). This includes compounds that can enter the system (*the medium*) and compounds that are allowed to exit the system, like metabolic byproducts. In most cases, all compounds that occur in the extracellular space should also be defined in the exchange compounds (with lower limit of zero) so that they are allowed to leave the model system, and PSAMM will generate a warning if this is not the case for some compounds. Compounds that are allowed to be taken up (*the medium*) should in addition be specified with a negative lower limit indicating the maximum allowed uptake.

The following fragment is an example of the `exchange.yaml` file:

```
compartment: e  # default compartment
compounds:
  - id: ac       # Acetate
  - id: co2
  - id: o2
  - id: glcD     # D-Glucose with uptake limit of 10
    lower: -10
  # ...
```

When an exchange file is specified, the corresponding exchange reactions are automatically added. For example, if the compounds `o2` in compartment `e` is in the exchange file, the exchange reaction `EX_o2_e` is added to the model. The desired ID for the exchange reaction can be set explicitly using the `reaction` attribute.

The exchange set can also be specified using a TSV-file as the following fragment shows. The second column specifies the compartment while third and fourth columns specify the lower and upper bounds, respectively. Both can be omitted or specified as − to use the default flux bounds:

```
# Acetate exchange with default lower and upper bounds
ac      e
# D-Glucose with uptake limit of 10
glcD    e       -10
# CO2 exchange with production limit of 50 and default uptake limit
co2     e       -       50
```

Multiple exchange files can be included from the main `exchange.yaml` file, and these will be combined to form the final set of exchange reactions used for the simulations.

## Reaction flux limits

The optional `limits` property lists the files that are to be combined and applied as the reaction flux limits. This can be used to limit certain reactions in the model. The following fragment is an example of a limits file in the YAML format. The lower and upper specifies the flux bounds and they are both optional. The fixed key is a shortcut to set both lower and upper to its value:

```
- reaction: ADK1
  upper: 10
- reaction: ADE2t
  lower: -50
  upper: 50
- reaction: DHPTDNRN
  fixed: 0
```

The limits can also be specified using a TSV-file as shown in the following fragment:

```
# Make ADE2t irreversible by imposing a lower bound of 0
ADE2t    0
# Only allow limited flux on ADK1
ADK1     -10    10
```

## Model Definition

The `model` property can be used to include a table file that specifies a subset of reactions that are used in the model. If no model definition file is given then all the reactions in the model will be used:

```
ACALD
ACALDt
ACKr
...
```

CHAPTER 5

# Command line interface

The tools that can be applied to metabolic models are run through the `psamm-model` program. To see the full help text of the program use

```
$ psamm-model --help
```

This program allows you to specify a metabolic model and a command to apply to the given model. The available commands can be seen using the help command given above, and are also described in more details below.

To run the program with a model, use

```
$ psamm-model --model model.yaml command [...]
```

In most cases you will probably be running the command from the same directory as where the `model.yaml` file is located, and in that case you can simply run

```
$ psamm-model command [...]
```

To see the help text of a command use

```
$ psamm-model command --help
```

## Linear programming solver

Many of the commands described below use a linear programming (LP) solver in order to perform the analysis. These commands all take an option –*solver* which can be used to select which solver to use and to specify additional options for the LP solver. For example, in order to run the `fba` command with the QSopt_ex solver, the option `--solver name=qsoptex` can be added:

```
$ psamm-model fba --solver name=qsoptex
```

The `--solver` option can also be used to specify additional options for the solver in use. For example, the Cplex solver recognizes the `threads` option which can be used to adjust the maximum number of threads that Cplex will use internally (by default, Cplex will use as many threads as there are cores on the computer):

```
$ psamm-model fba --solver threads=4
```

## Flux balance analysis (`fba`)

This command will try to maximize the flux of the biomass reaction defined in the model. It is also possible to provide a different reaction on the command line to maximize. *[Orth10] [Fell86]*

To run FBA use:

```
$ psamm-model fba
```

or with a specific reaction:

```
$ psamm-model fba --objective=ATPM
```

By default, this performs a standard FBA and the result is output as tab-separated values with the reaction ID, the reaction flux and the reaction equation. If the parameter `--loop-removal` is given, the flux of the internal reactions is further constrained to remove internal loops *[Schilling00]*. Loop removal is more time-consuming and under normal circumstances the biomass reaction flux will *not* change in response to the loop removal (only internal reaction fluxes may change). The `--loop-removal` option is followed by `none` (no loop removal), `tfba` (removal using thermodynamic constraints), or `l1min` (L1 minimization of the fluxes). For example, the following command performs an FBA with thermodynamic constraints:

```
$ psamm-model fba --loop-removal=tfba
```

## Flux variability analysis (`fva`)

This command will find the possible flux range of each reaction when the biomass is at the maximum value *[Mahadevan03]*. The command will use the biomass reaction specified in the model definition, or alternatively, a reaction can be given on the command line following the `--objective` option.

```
$ psamm-model fva
```

The output of the command will show each reaction in the model along with the minimum and maximum possible flux values as tab-separated values.

```
PPCK    0.0     135.266721627   [...]
PTAr    62.3091585921    1000.0   [...]
```

In this example the `PPCK` reaction has a minimum flux of zero and maximum flux of 135.3 units. The `PTAr` reaction has a minimum flux of 62.3 and a maximum of 1000 units.

If the parameter `--loop-removal=tfba` is given, additonal thermodynamic constraints will be imposed when evaluating model fluxes. This automatically removes internal flux loops *[Schilling00]* but is much more time-consuming.

# Robustness (robustness)

Given a reaction to maximize and a reaction to vary, the robustness analysis will run flux balance analysis and flux minimization while fixing the reaction to vary at each iteration. The reaction will be fixed at a given number of steps between the minimum and maximum flux value specified in the model *[Edwards00]*.

```
$ psamm-model robustness \
    --steps 200 --minimum -20 --maximum 160 EX_Oxygen
```

In the example above, the biomass reaction will be maximized while the `EX_Oxygen` (oxygen exchange) reaction is fixed at a certain flux in each iteration. The fixed flux will vary between the minimum and maximum flux. The number of iterations can be set using `--steps`. In each iteration, all reactions and the corresponding fluxes will be shown in a table, as well as the value of the fixed flux. If the fixed flux results in an infeasible model, no output will be shown for that iteration.

The output of the command is a list of tab-separated values indicating a reaction ID, the flux of the varying reaction, and the flux of the reaction with the given ID.

If the parameter `--loop-removal` is given, additional constraints on the model can be imposed that remove internal flux loops. See the section on the *Flux balance analysis (fba)* command for more information on this option.

# Random sparse network (randomsparse)

Delete reactions randomly until the flux of the biomass reaction falls below the threshold. Keep deleting reactions until no more reactions can be deleted. This can also be applied to other reactions than the biomass reaction by specifying the reaction explicitly.

```
$ psamm-model randomsparse 95%
```

When the given reaction is the biomass reaction, this results in a smaller model which is still producing biomass within the tolerance given by the threshold. The tolerance can be specified as a relative value (as above) or as an absolute flux. Aggregating the results from multiple random sparse networks allows classifying reactions as essential, semi-essential or non-essential.

If the option `--exchange` is given, the model will only try to delete exchange reactions. This can be used to provide putative minimal media for the model.

The output of the command is a tab-separated list of reaction IDs and a value indicating whether the reaction was eliminated (`0` when eliminated, `1` otherwise). If multiple minimal networks are desired, the command can be run again and it will sample another random minimal network.

# Gene Deletion (genedelete)

Delete single and multiple genes from a model. Once gene(s) are given the command will delete reactions from the model requiring the gene(s) specified. The reactions deleted will be returned as a set as well as the flux of the model with the specified gene(s) removed.

```
$ psamm-model genedelete
```

To delete genes the option `--gene` must be entered followed by the desired gene ID specified in the model files. To delete multiple genes, each new gene must first be followed by a `--gene` option. For example:

```
$ psamm-model genedelete --gene ExGene1 --gene ExGene2
```

The list of genes to delete can also be specified in a text file. This allows to you perform many gene deletions by simply specifying the file name when running the `genedelete` command. The text file must contain one gene ID per line. For example:

```
$ psamm-model genedelete --gene @gene_file.txt
```

The file gene_file.txt would contain the following lines:

```
ExGene1
ExGene2
```

## Flux coupling analysis (`fluxcoupling`)

The flux coupling analysis identifies any reaction pairs where the flux of one reaction constrains the flux of another reaction. The reactions can be coupled in three distinct ways depending on the ratio between the reaction fluxes *[Burgard04]*.

The reactions can be fully coupled (the ratio is static and non-zero); partially coupled (the ratio is bounded and non-zero); or directionally coupled (the ratio is non-zero).

```
$ psamm-model fluxcoupling
```

## Stoichiometric consistency check (`masscheck`)

A model or reaction database can be checked for stoichiometric inconsistencies (mass inconsistencies). The basic idea is that we should be able to assign a positive mass to each compound in the model and have each reaction be balanced with respect to these mass assignments. If it can be shown that assigning the masses is impossible, we have discovered an inconsistency *[Gevorgyan08]*.

Some variants of this idea is implemented in the *psamm.massconsistency* module. The mass consistency check can be run using

```
$ psamm-model masscheck
```

This will first try to assign a positive mass to as many compounds as possible. This will indicate whether or not the model is consistent but in case it is *not* consistent it is often hard to figure out how to fix the model from this list of masses:

```
[...]
INFO: Checking stoichiometric consistency of reactions...
C0223       1.0     Dihydrobiopterin
C9779       1.0     2-hydroxy-Octadec-ACP(n-C18:1)
EC0065      0.0     H+[e]
C0065       0.0     H+
INFO: Consistent compounds: 832/834
```

In this case the *H+* compounds were inconsistent because they were not assigned a non-zero mass. A different check can be run where the residual mass is minimized for all reactions in the model. This will often give a better idea of which reactions need fixing:

```
.. code-block:: shell
```

$ psamm-model masscheck –type=reaction

The following output might be generated from this command:

```
[...]
INFO: Checking stoichiometric consistency of reactions...
IR01815     7.0     (6) |H+[c]| + |Uroporphyrinogen III[c]| [...]
IR00307     1.0     |H+[c]| + |L-Arginine[c]| => [...]
IR00146     0.5     |UTP[c]| + |D-Glucose 1-phosphate[c]| => [...]
[...]
INFO: Consistent reactions: 946/959
```

This is a list of reactions with non-zero residuals and their residual values. In the example above the three reactions that are shown have been assigned a non-zero residual (7, 1 and 0.5, respectively). This means that there is an issue either with this reaction itself or a closely related one. In this example the first two reactions were missing a number of *H+* compounds for the reaction to balance.

Now the mass check can be run again marking the reactions above as checked:

```
$ psamm-model masscheck --type=reaction --checked IR01815 \
    --checked IR00307 --checked IR00146
[...]
IR00149 0.5     |ATP[c]| + |D-Glucose[c]| => [...]
```

The output has now changed and the remaining residual has been shifted to another reaction. This iterative procedure can be continued until all stoichiometric inconsistencies have been corrected. In this example the *IR00149* reaction also had a missing *H+* for the reaction to balance. After fixing this error the model is consistent and the *H+* compounds can be assigned a non-zero mass:

```
$ psamm-model masscheck
[...]
EC0065      1.0     H+[e]
C0065       1.0     H+
INFO: Consistent compounds: 834/834
```

## Formula consistency check (`formulacheck`)

Similarly, a model or reaction database can be checked for formula inconsistencies when the chemical formulae of the compounds in the model are known.

```
$ psamm-model formulacheck
```

For each inconsistent reaction, the reaction identifier will be printed followed by the elements ("atoms") in, respectively, the left- and right-hand side of the reaction, followed by the elements needed to balance the left- and right-hand side, respectively.

## Charge consistency check (`chargecheck`)

The charge check will evaluate whether the compound charge is balanced in all reactions of the model. Any reactions that have an imbalance of charge will be reported along with the excess charge.

```
$ psamm-model chargecheck
```

## Flux consistency check (`fluxcheck`)

The flux consistency check will report any reactions that are unable to take on a non-zero flux. This is useful for finding any reactions that do not contribute anything to the model simulation. This may indicate that the reaction is part of a pathway that is incompletely modeled.

```
$ psamm-model fluxcheck
```

If the parameter `--loop-removal=tfba` is given, additional thermodynamic constraints are imposed when considering whether reactions can take a non-zero flux. This automatically removes internal flux loops but is also much more time-consuming.

## Reaction duplicates check (`duplicatescheck`)

This command simply checks whether multiple reactions exist in the model that have the same or similar reaction equations. By default, this check will ignore reaction directionality and stoichiometric values when considering whether reactions are identical. The options `--compare-direction` and `--compare-stoichiometry` can be used to make the command consider these properties as well.

```
$ psamm-model duplicatescheck
```

## Gap check (`gapcheck`)

This gap check command will try to identify the compounds in the model that cannot be produced. This is useful for identifying incomplete pathways in the model. The command will report a list of all compounds in the model that are blocked for production.

```
$ psamm-model gapcheck
```

When checking whether a compound can be produced, it is sufficient for production that all precursors can be produced and it is *not* necessary for every compound to also be consumed by another reaction (in other words, for the purpose of this analysis there are implicit sinks for every compound in the model). This means that even if this command reports that no compounds are blocked, it may still not be possible for the model to be viable under the steady-state assumption of FBA. The option `--no-implicit-sinks` can be used to perform the gap check without implicit sinks.

The gap check is performed with the medium that is defined in the model. It may be useful to run the gap check with every compound in the medium available. This can easily be done by specifying the `--unrestricted-exchange` option which removes all limits on the exchange reactions during the check.

There are some additional gap checking methods that can be enabled with the `--method` option. The method `sinkcheck` can be used to find compounds that cannot be synthesized from scratch. The standard gap check will report compounds as produced if they can participate in a reaction, even if the compound itself cannot be synthesized from precursors in the medium. To find such compounds use the `sinkcheck`. This check will generally indicate more compounds as blocked. Lastly, the method `gapfind` can be used. This method should produce the same result as the default method but is implemented in an alternative way that is specified in *[Kumar07]*. This method is *not* used by default because it tends to result in difficulties for the solver when used with larger models.

## GapFill (`gapfill`)

The GapFill algorithm will try to compute an extension of the model with reactions from the reaction database and try to find a minimal subset that allows all blocked compounds to be produced. In addition to suggesting possible database reactions to add to the model, the command will also suggest possible transport and exchange reactions. The GapFill algorithm implemented in this command is a variant of the gap-filling procedure described in *[Kumar07]*.

```
$ psamm-model gapfill
```

The command will first list the reactions in the model followed by the suggested reactions to add to the model in order to unblock the blocked compounds. If `--allow-bounds-expansion` is specified, the procedure may also suggest that existing model reactions have their flux bounds widened, e.g. making an existing irreversible reaction reversible. To unblock only specific compounds, use the `--compound` option:

```
$ psamm-model gapfill --compound leu-L[c] --compound ile-L[c]
```

In this example, the procedure will try to add reactions so that leucine (`leu-L`) and isoleucine (`ile-L`) in the `c` compartment can be produced. Multiple compounds can be unblocked at the same time and the list of compounds to unblock can optionally be specified as a file by prefixing the file name with `@`.

```
$ psamm-model gapfill --compound @list_of_compounds_to_unblock.tsv
```

The GapFind algorithm is defined in terms of a MILP problem and can therefore be computationally expensive to run for larger models.

The original GapFill algorithm uses a solution procedure which implicitly assumes that the model contains implicit sinks for all compounds. This means that even with the reactions proposed by GapFill the model may need to produce compounds that cannot be used anywhere. The implicit sinks can be disabled with the `--no-implicit-sinks` option.

## FastGapFill (`fastgapfill`)

The FastGapFill algorithm tries to reconstruct a flux consistent model (i.e. a model where every reaction takes a non-zero flux for at least one solutions). This is done by extending the model with reactions from the reaction database and trying to find a minimal subset that is flux consistent. The solution is approximate *[Thiele14]*.

The database reactions can be assigned a weight (or "cost") using the `--penalty` option. These weights are taken into account when determining the minimal solution.

```
$ psamm-model fastgapfill --penalty penalty.tsv
```

## SBML Export (`sbmlexport`)

Exports the model to the SBML file format. This command exports the model as an SBML level 3 file with flux bounds, objective and gene information encoded with Flux Balance Constraints version 2.

```
$ psamm-model sbmlexport model.xml
```

If the file name is omitted, the file contents will be output directly to the screen. Using the `--pretty` option makes the output formatted for readability.

## Excel Export (excelexport)

Exports the model to the Excel file format.

```
$ psamm-model excelexport model.xls
```

## Table Export (tableexport)

Exports the model to the tsv file format.

```
$ psamm-model tableexport reactions > model.tsv
```

## Search (search)

This command can be used to search in a database for compounds or reactions. To search for a compound use

```
$ psamm-model search compound [...]
```

Use the `--name` option to search for a compound with a specific name or use the `--id` option to search for a compound with a specific identifier.

To search for a reaction use

```
$ psamm-model search reaction [...]
```

Use the `--id` option to search for a reaction with a specific identifier. The `--compound` option can be used to search for reactions that include a specific compound. If more that one compound identifier is given (comma-separated) this will find reactions that include all of the given compounds.

## Console (console)

This command will start a Python session where the model has been loaded into the corresponding Python object representation.

```
$ psamm-model console
```

# CHAPTER 6

# Development

## Test suite

The python modules have test suites that allows us to automatically test various aspects of the module implementation. It is important to make sure that all tests run without failure *before* committing changes to any of the modules. The test suite is run by changing to the project directory and running

```
$ ./setup.py test
```

To run the tests on all the supported Python platforms with additional tests for coding style (PEP8) and building documentation, use tox:

```
$ tox
```

When testing with tox, the local path for the Cplex python module must be provided in the environment variables `CPLEX_PYTHON2_PACKAGE` and `CPLEX_PYTHON3_PACKAGE` for Python 2 and Python 3, respectively. For example:

```
$ export CPLEX_PYTHON2_PACKAGE=/path/to/IBM/ILOG/CPLEX_StudioXXX/cplex/python/2.7/x86-
↪64_osx
$ export CPLEX_PYTHON3_PACKAGE=/path/to/IBM/ILOG/CPLEX_StudioXXX/cplex/python/3.4/x86-
↪64_osx
$ tox -e py27-cplex,py34-cplex
```

**Note:** Python 3 support was added in a recent release of Cplex. Older versions only support Python 2.

Similarly, the local path to the Gurobi package must be specified in the environment variable `GUROBI_PYTHON_PACKAGE`:

```
$ export GUROBI_PYTHON_PACKAGE=/Library/gurobi604/mac64
$ tox -e py27-gurobi
```

## Adding new tests

Adding or improving tests for python modules is highly encouraged. A test suite for a new module should be created in `tests/test_<modulename>.py`. These test suites use the built-in `unittest` module.

## Documentation tests

In addition, some modules have documentation that can be tested using the `doctest` module. These test suites should also run without failure before any commits. They can be run by specifying the particular module (e.g the `affine` module in `expression`) using

```
$ python -m psamm.expression.affine -v
```

# CHAPTER 7

## FAQ

**When I run PSAMM it exits with the error “No solvers available”. How can I fix this?**

This means that PSAMM is searching for a linear programming solver but was not able to find one. This can occur even when the Cplex solver is installed because the Cplex Python-bindings have to be installed separately from the main Cplex package (see *Cplex*). Also, if using a *Virtualenv* with Python, the Cplex Python-bindings must be installed *in the Virtualenv*. The bindings will *not* be available in the Virtualenv if they are installed globally.

To check whether the Cplex Python-bindings are correctly installed use the following command:

```
(env) $ pip list
```

This will show a list of Python packages that are available. The package `cplex` will appear in this list if the Cplex Python-bindings are correctly installed. If the package does *not* appear in the list, follow the instuctions at *Cplex* to install the package.

# CHAPTER 8

# PSAMM API

## psamm.balancecheck – check balance of charge and formula

psamm.balancecheck.**charge_balance**(*model*)
Calculate the overall charge for all reactions in the model.

Yield (reaction, charge) pairs.

**Parameters** **model** – *psamm.datasource.native.NativeModel*.

psamm.balancecheck.**formula_balance**(*model*)
Calculate formula compositions for each reaction.

Call *reaction_formula()* for each reaction. Yield (reaction, result) pairs, where result has two formula compositions or *None*.

**Parameters** **model** – *psamm.datasource.native.NativeModel*.

psamm.balancecheck.**reaction_charge**(*reaction*, *compound_charge*)
Calculate the overall charge for the specified reaction.

**Parameters**

- **reaction** – *psamm.reaction.Reaction*.
- **compound_charge** – a map from each compound to charge values.

psamm.balancecheck.**reaction_formula**(*reaction*, *compound_formula*)
Calculate formula compositions for both sides of the specified reaction.

If the compounds in the reaction all have formula, then calculate and return the chemical compositions for both sides, otherwise return *None*.

**Parameters**

- **reaction** – *psamm.reaction.Reaction*.
- **compound_formula** – a map from compound id to formula.

## psamm.command – Command line interface

Command line interface.

Each command in the command line interface is implemented as a subclass of `Command`. Commands are also referenced from `setup.py` using the entry point mechanism which allows the commands to be automatically discovered.

The `main()` function is the entry point of command line interface.

**class** `psamm.command.`**`Command`**(*model*, *args*)

Represents a command in the interface, operating on a model.

The constructor will be given the NativeModel and the command line namespace. The subclass must implement `run()` to handle command execution. The doc string will be used as documentation for the command in the command line interface.

In addition, `init_parser()` can be implemented as a classmethod which will allow the command to initialize an instance of `argparse.ArgumentParser` as desired. The resulting argument namespace will be passed to the constructor.

**`argument_error`**(*msg*)

Raise error indicating error parsing an argument.

**`fail`**(*msg*, *exc=None*)

Exit command as a result of a failure.

**classmethod** **`init_parser`**(*parser*)

Initialize command line parser (`argparse.ArgumentParser`)

**`run`**()

Execute command

**exception** `psamm.command.`**`CommandError`**

Error from running a command.

This should be raised from a `Command.run()` if any arguments are misspecified. When the command is run and the `CommandError` is raised, the caller will exit with an error code and print appropriate usage information.

**exception** `psamm.command.`**`ExecutorError`**

Error running tasks on executor.

**class** `psamm.command.`**`FilePrefixAppendAction`**(*option_strings*, *dest*, *nargs=None*, *fromfile_prefix_chars=u'@'*, ***kwargs*)

Action that appends one argument or multiple from a file.

If the argument starts with a character in `fromfile_prefix_chars` the remaining part of the argument is taken to be a file path. The file is read and every line is appended. Otherwise, the argument is simply appended.

**class** `psamm.command.`**`LoopRemovalMixin`**

Mixin for commands that perform loop removal.

**class** `psamm.command.`**`MetabolicMixin`**(**args*, ***kwargs*)

Mixin for commands that use a metabolic model representation.

**class** `psamm.command.`**`ObjectiveMixin`**

Mixin for commands that use biomass as objective.

Allows the user to override the default objective from the command line.

**class** `psamm.command.`**`ParallelTaskMixin`**

Mixin for commands that run parallel computation tasks.

**class** `psamm.command.`**`SolverCommandMixin`**`(`**args*, ***kwargs*`)`
Mixin for commands that use an LP solver.

This adds a `--solver` parameter to the command that the user can use to select a specific solver. It also adds the method `_get_solver()` which will return a solver with the specified default requirements. The user requirements will override the default requirements.

`psamm.command.`**`main`**`(`*command_class=None*, *args=None*`)`
Run the command line interface with the given `Command`.

If no command class is specified the user will be able to select a specific command through the first command line argument. If the `args` are provided, these should be a list of strings that will be used instead of `sys.argv[1]`. This is mostly useful for testing.

## `psamm.database` – Reaction database

Representation of metabolic network databases.

**class** `psamm.database.`**`ChainedDatabase`**`(`**databases*`)`
Links a number of databases so they can be treated a single database

This is a subclass of `MetabolicDatabase`.

**class** `psamm.database.`**`DictDatabase`**
Metabolic database backed by in-memory dictionaries

This is a subclass of `MetabolicDatabase`.

**`set_reaction`**`(`*reaction_id*, *reaction*`)`
Set the reaction ID to a reaction given by a `Reaction`

If an existing reaction exists with the given reaction ID it will be overwritten.

**class** `psamm.database.`**`MetabolicDatabase`**
Database of metabolic reactions.

**`compartments`**
Iterator of compartment IDs in the database.

**`compounds`**
Itertor of `Compounds` in the database.

**`get_compound_reactions`**`(`*compound_id*`)`
Return an iterator of reactions containing the compound.

Reactions are returned as IDs.

**`get_reaction`**`(`*reaction_id*`)`
Return reaction as a `Reaction`.

**`get_reaction_values`**`(`*reaction_id*`)`
Return an iterator of reaction compounds and stoichiometric values.

The returned iterator contains (`Compound`, value)-tuples. The value is negative for left-hand side compounds and positive for right-hand side.

**`has_reaction`**`(`*reaction_id*`)`
Whether the given reaction exists in the database.

**`is_reversible`**`(`*reaction_id*`)`
Whether the given reaction is reversible.

**matrix**
Mapping from compound, reaction to stoichiometric value.

This is an instance of *StoichiometricMatrixView*.

**reactions**
Iterator of reactions IDs in the database.

**reversible**
The set of reversible reactions.

**class** psamm.database.**StoichiometricMatrixView**(*database*)
Provides a sparse matrix view on the stoichiometry of a database.

This object is used internally in the database to expose a sparse matrix view of the model stoichiometry. This class should not be instantied, instead use the *MetabolicDatabase.matrix* property. Any compound, reaction-pair can be looked up to obtain the corresponding stoichiometric value. If the value is not defined (implicitly zero) a KeyError will be raised.

In addition, instances also support the NumPy *__array__* protocol which means that a numpy.array can by created directly from the matrix.

```
>>> model = MetabolicModel()
>>> matrix = numpy.array(model.matrix)
```

## psamm.datasource.context – File system contexts

Utilities for keeping track of parsing context.

**exception** psamm.datasource.context.**ContextError**
Raised when a context failure occurs.

**class** psamm.datasource.context.**FileMark**(*filecontext, line, column*)
Marks a position in a file.

This is used when parsing input files, to keep track of the position that generates an entry.

**class** psamm.datasource.context.**FilePathContext**(*arg*)
File context that keeps track of contextual information.

When a file is loaded, all files specified in that file must be loaded relative to the first file. This is made possible by keeping a context that remembers where a file was loaded so that other files can be loaded relatively.

## psamm.datasource.entry – Model entry representations

Representation of compound/reaction entries in models.

**class** psamm.datasource.entry.**CompartmentEntry**
Abstract compartment entry.

**class** psamm.datasource.entry.**CompoundEntry**
Abstract compound entry.

**charge**
Compound charge value.

**formula**
Chemical formula of compound.

**class** psamm.datasource.entry.**DictCompartmentEntry**(**args*, ***kwargs*)
Compartment entry backed by dictionary.

The given properties dictionary must contain a key 'id' with the identifier.

**Parameters**

- **properties** – dict or *CompartmentEntry* to construct from.
- **filemark** – Where the entry was parsed from (optional)

**class** psamm.datasource.entry.**DictCompoundEntry**(**args*, ***kwargs*)
Compound entry backed by dictionary.

The given properties dictionary must contain a key 'id' with the identifier.

**Parameters**

- **properties** – dict or *CompoundEntry* to construct from.
- **filemark** – Where the entry was parsed from (optional)

**class** psamm.datasource.entry.**DictReactionEntry**(**args*, ***kwargs*)
Reaction entry backed by dictionary.

The given properties dictionary must contain a key 'id' with the identifier.

**Parameters**

- **properties** – dict or *ReactionEntry* to construct from.
- **filemark** – Where the entry was parsed from (optional)

**class** psamm.datasource.entry.**ModelEntry**
Abstract model entry.

**filemark**
Position of entry in the source file (or None).

**id**
Identifier of entry.

**name**
Name of entry (or None).

**properties**
Properties of entry as a Mapping subclass (e.g. dict).

Note that the properties are not generally mutable.

**class** psamm.datasource.entry.**ReactionEntry**
Abstract reaction entry.

**equation**
Reaction equation.

**genes**
Gene association expression.

## psamm.datasource.kegg – KEGG data parser

Module related to loading KEGG database files.

**class** psamm.datasource.kegg.**CompoundEntry**(*entry*)
KEGG entry with mapped compound properties.

**class** psamm.datasource.kegg.**CompoundMapper**
Mapper for raw KEGG compound properties to standard properties.

Public methods are automatically translated into cached properties by the metaclass.

**class** psamm.datasource.kegg.**KEGGEntry**(*properties*, *filemark=None*)
Base class for KEGG entry with raw values from KEGG.

**exception** psamm.datasource.kegg.**ParseError**
Exception used to signal errors while parsing

**class** psamm.datasource.kegg.**ReactionEntry**(*entry*)
KEGG entry with mapped reaction properties.

**class** psamm.datasource.kegg.**ReactionMapper**
Mapper for raw KEGG reaction properties to standard properties.

Methods are automatically translated into cached properties by the metaclass.

psamm.datasource.kegg.**parse_compound_file**(*f*, *context=None*)
Iterate over the compound entries in the given file.

psamm.datasource.kegg.**parse_kegg_entries**(*f*, *context=None*)
Iterate over entries in KEGG file.

psamm.datasource.kegg.**parse_reaction**(*s*)
Parse a KEGG reaction string

psamm.datasource.kegg.**parse_reaction_file**(*f*, *context=None*)
Iterate over the reaction entries in the given file.

## psamm.datasource.modelseed – ModelSEED data parser

Module related to loading ModelSEED database files.

**class** psamm.datasource.modelseed.**CompoundEntry**(*id*, *names*, *formula*, *filemark=None*)
Representation of entry in a ModelSEED compound table

**exception** psamm.datasource.modelseed.**ParseError**
Exception used to signal errors while parsing

psamm.datasource.modelseed.**decode_name**(*s*)
Decode names in ModelSEED files

psamm.datasource.modelseed.**parse_compound_file**(*f*, *context=None*)
Iterate over the compound entries in the given file

## psamm.datasource.native – Native data format parser

Module for reading and writing native formats.

These formats are either table-based or YAML-based. Table-based formats are space-separated and empty lines are ignored. Comments starting with pound (#). YAML-based formats are structured data following the YAML specification.

**class** psamm.datasource.native.**ModelReader**(*model_from*, *context=None*)
Reader of native YAML-based model format.

The reader can be created from a model YAML file or directly from a dict, string or File-like object. Use *reader_from_path()* to read the model from a YAML file or directory and use the constructor to read from other sources. Any externally referenced file (with include) will be read on demand by the parse methods. To read the model fully into memory, use the *create_model()* to create a *NativeModel*.

**biomass_reaction**
Biomass reaction specified by the model.

**create_model**()
Return *NativeModel* fully loaded into memory.

**default_compartment**
Default compartment specified by the model.

The compartment that is implied when not specified. In some contexts (e.g. for exchange compounds) the extracellular compartment may be implied instead. Defaults to 'c'.

**default_flux_limit**
Default flux limit specified by the model.

When flux limits on reactions are not specified, this value will be used. Flux limit of [0;x] will be implied for irreversible reactions and [-x;x] for reversible reactions, where x is this value. Defaults to 1000.

**extracellular_compartment**
Extracellular compartment specified by the model.

Defaults to 'e'.

**has_model_definition**()
Return True when the list of model reactions is set in the model.

**name**
Name specified by the model.

**parse_compartments**()
Parse compartment information from model.

Return tuple of: 1) iterator of *psamm.datasource.entry.CompartmentEntry*; 2) Set of pairs defining the compartment boundaries of the model.

**parse_compounds**()
Yield CompoundEntries for defined compounds

**parse_exchange**()
Yield tuples of exchange compounds.

Each exchange compound is a tuple of compound, reaction ID, lower and upper flux limits.

**parse_limits**()
Yield tuples of reaction ID, lower, and upper bound flux limits

**parse_medium**()
Yield tuples of exchange compounds.

Each exchange compound is a tuple of compound, reaction ID, lower and upper flux limits.

**parse_model**()
Yield reaction IDs of model reactions

**parse_reactions**()
Yield tuples of reaction ID and reactions defined in the model

**classmethod reader_from_path** (*path*)
Create a model from specified path.

Path can be a directory containing a `model.yaml` or `model.yml` file or it can be a path naming the central model file directly.

**class** psamm.datasource.native.**ModelWriter**
Writer for native (YAML) format.

**convert_compartment_entry** (*compartment*, *adjacencies*)
Convert compartment entry to YAML dict.

**Parameters**

- **compartment** – *psamm.datasource.entry.CompartmentEntry*.
- **adjacencies** – Sequence of IDs or a single ID of adjacent compartments (or None).

**convert_compound_entry** (*compound*)
Convert compound entry to YAML dict.

**convert_reaction_entry** (*reaction*)
Convert reaction entry to YAML dict.

**write_compartments** (*stream*, *compartments*, *adjacencies*, *properties=None*)
Write iterable of compartments as YAML object to stream.

**Parameters**

- **stream** – File-like object.
- **compartments** – Iterable of compartment entries.
- **adjacencies** – Dictionary mapping IDs to adjacent compartment IDs.
- **properties** – Set of compartment properties to output (or None to output all).

**write_compounds** (*stream*, *compounds*, *properties=None*)
Write iterable of compounds as YAML object to stream.

**Parameters**

- **stream** – File-like object.
- **compounds** – Iterable of compound entries.
- **properties** – Set of compound properties to output (or None to output all).

**write_reactions** (*stream*, *reactions*, *properties=None*)
Write iterable of reactions as YAML object to stream.

**Parameters**

- **stream** – File-like object.
- **compounds** – Iterable of reaction entries.
- **properties** – Set of reaction properties to output (or None to output all).

**class** psamm.datasource.native.**NativeModel** (*properties={}*)
Represents model in the native format.

**biomass_reaction**
Return biomass reaction property.

**compartment_boundaries**
Return set of compartment boundaries.

**compartments**
Return compartments entry set.

**compounds**
Return compound entry set.

**create_metabolic_model**()
Create a *psamm.metabolicmodel.MetabolicModel*.

**default_compartment**
Return default compartment property.

**default_flux_limit**
Return default flux limit property.

**exchange**
Return dict of exchange compounds and properties.

**extracellular_compartment**
Return extracellular compartment property.

**limits**
Return dict of reaction limits.

**model**
Return dict of model reactions.

**name**
Return model name property.

**reactions**
Return reaction entry set.

**version_string**
Return model version string.

**exception** psamm.datasource.native.**ParseError**
Exception used to signal errors while parsing

psamm.datasource.native.**float_constructor**(*loader*, *node*)
Construct Decimal from YAML float encoding.

psamm.datasource.native.**parse_compound**(*compound_def*, *context=None*)
Parse a structured compound definition as obtained from a YAML file

Returns a CompoundEntry.

psamm.datasource.native.**parse_compound_file**(*path*, *format*)
Open and parse reaction file based on file extension or given format

Path can be given as a string or a context.

psamm.datasource.native.**parse_compound_list**(*path*, *compounds*)
Parse a structured list of compounds as obtained from a YAML file

Yields CompoundEntries. Path can be given as a string or a context.

psamm.datasource.native.**parse_compound_table_file**(*path*, *f*)
Parse a tab-separated file containing compound IDs and properties

The compound properties are parsed according to the header which specifies which property is contained in each column.

psamm.datasource.native.**parse_compound_yaml_file**(*path*, *f*)
Parse a file as a YAML-format list of compounds

Path can be given as a string or a context.

psamm.datasource.native.**parse_exchange**(*exchange_def*, *default_compartment*)
Parse a structured exchange definition as obtained from a YAML file.

Returns in iterator of compound, reaction, lower and upper bounds.

psamm.datasource.native.**parse_exchange_file**(*path*, *default_compartment*)
Parse a file as a list of exchange compounds with flux limits.

The file format is detected and the file is parsed accordingly. Path can be given as a string or a context.

psamm.datasource.native.**parse_exchange_list**(*path*, *exchange*, *default_compartment*)
Parse a structured exchange list as obtained from a YAML file.

Yields tuples of compound, reaction ID, lower and upper flux bounds. Path can be given as a string or a context.

psamm.datasource.native.**parse_exchange_table_file**(*f*)
Parse a space-separated file containing exchange compound flux limits.

The first two columns contain compound IDs and compartment while the third column contains the lower flux limits. The fourth column is optional and contains the upper flux limit.

psamm.datasource.native.**parse_exchange_yaml_file**(*path*, *f*, *default_compartment*)
Parse a file as a YAML-format exchange definition.

Path can be given as a string or a context.

psamm.datasource.native.**parse_limit**(*limit_def*)
Parse a structured flux limit definition as obtained from a YAML file

Returns a tuple of reaction, lower and upper bound.

psamm.datasource.native.**parse_limits_file**(*path*)
Parse a file as a list of reaction flux limits

The file format is detected and the file is parsed accordingly. Path can be given as a string or a context.

psamm.datasource.native.**parse_limits_list**(*path*, *limits*)
Parse a structured list of flux limits as obtained from a YAML file

Yields tuples of reaction ID, lower and upper flux bounds. Path can be given as a string or a context.

psamm.datasource.native.**parse_limits_table_file**(*f*)
Parse a space-separated file containing reaction flux limits

The first column contains reaction IDs while the second column contains the lower flux limits. The third column is optional and contains the upper flux limit.

psamm.datasource.native.**parse_limits_yaml_file**(*path*, *f*)
Parse a file as a YAML-format flux limits definition

Path can be given as a string or a context.

psamm.datasource.native.**parse_medium**(*exchange_def*, *default_compartment*)
Parse a structured exchange definition as obtained from a YAML file.

Returns in iterator of compound, reaction, lower and upper bounds.

psamm.datasource.native.**parse_medium_file**(*path*, *default_compartment*)
Parse a file as a list of exchange compounds with flux limits.

The file format is detected and the file is parsed accordingly. Path can be given as a string or a context.

`psamm.datasource.native.`**`parse_medium_list`**`(`*path*, *exchange*, *default_compartment*`)`
Parse a structured exchange list as obtained from a YAML file.

Yields tuples of compound, reaction ID, lower and upper flux bounds. Path can be given as a string or a context.

`psamm.datasource.native.`**`parse_medium_table_file`**`(`*f*`)`
Parse a space-separated file containing exchange compound flux limits.

The first two columns contain compound IDs and compartment while the third column contains the lower flux limits. The fourth column is optional and contains the upper flux limit.

`psamm.datasource.native.`**`parse_medium_yaml_file`**`(`*path*, *f*, *default_compartment*`)`
Parse a file as a YAML-format exchange definition.

Path can be given as a string or a context.

`psamm.datasource.native.`**`parse_model_file`**`(`*path*`)`
Parse a file as a list of model reactions

The file format is detected and the file is parsed accordinly. The file is specified as a file path that will be opened for reading. Path can be given as a string or a context.

`psamm.datasource.native.`**`parse_model_group`**`(`*path*, *group*`)`
Parse a structured model group as obtained from a YAML file

Path can be given as a string or a context.

`psamm.datasource.native.`**`parse_model_group_list`**`(`*path*, *groups*`)`
Parse a structured list of model groups as obtained from a YAML file

Yields reaction IDs. Path can be given as a string or a context.

`psamm.datasource.native.`**`parse_model_table_file`**`(`*path*, *f*`)`
Parse a file as a list of model reactions

Yields reactions IDs. Path can be given as a string or a context.

`psamm.datasource.native.`**`parse_model_yaml_file`**`(`*path*, *f*`)`
Parse a file as a YAML-format list of model reaction groups

Path can be given as a string or a context.

`psamm.datasource.native.`**`parse_reaction`**`(`*reaction_def*, *default_compartment*, *context=None*`)`
Parse a structured reaction definition as obtained from a YAML file

Returns a ReactionEntry.

`psamm.datasource.native.`**`parse_reaction_equation`**`(`*equation_def*, *default_compartment*`)`
Parse a structured reaction equation as obtained from a YAML file

Returns a Reaction.

`psamm.datasource.native.`**`parse_reaction_equation_string`**`(`*equation*, *default_compartment*`)`
Parse a string representation of a reaction equation.

Converts undefined compartments to the default compartment.

`psamm.datasource.native.`**`parse_reaction_file`**`(`*path*, *default_compartment=None*`)`
Open and parse reaction file based on file extension

Path can be given as a string or a context.

`psamm.datasource.native.`**`parse_reaction_list`**`(`*path*, *reactions*, *default_compartment=None*`)`
Parse a structured list of reactions as obtained from a YAML file

Yields tuples of reaction ID and reaction object. Path can be given as a string or a context.

psamm.datasource.native.**parse_reaction_table_file**(*path*, *f*, *default_compartment*)
Parse a tab-separated file containing reaction IDs and properties

The reaction properties are parsed according to the header which specifies which property is contained in each column.

psamm.datasource.native.**parse_reaction_yaml_file**(*path*, *f*, *default_compartment*)
Parse a file as a YAML-format list of reactions

Path can be given as a string or a context.

psamm.datasource.native.**resolve_format**(*format*, *path*)
Looks at a file's extension and format (if any) and returns format.

psamm.datasource.native.**yaml_load**(*stream*)
Load YAML file using safe loader.

## psamm.datasource.reaction – Parser for reactions

Reaction parser.

**exception** psamm.datasource.reaction.**ParseError**(**args*, ***kwargs*)
Error raised when parsing reaction fails.

**class** psamm.datasource.reaction.**ReactionParser**(*arrows=None*, *parse_global=False*)
Parser of reactions equations.

This parser recognizes:

- Global compartment specification as a prefix (when `parse_global` is `True`)
- Configurable reaction arrow tokens (`arrows`)
- Compounds quoted by pipe (`|`) (required only if the compound name includes a space)
- Compound counts that are affine expressions.

**parse**(*s*)
Parse reaction string.

psamm.datasource.reaction.**parse_compound**(*s*, *global_compartment=None*)
Parse a compound specification.

If no compartment is specified in the string, the global compartment will be used.

psamm.datasource.reaction.**parse_compound_count**(*s*)
Parse a compound count (number of compounds).

psamm.datasource.reaction.**parse_reaction**(*s*)
Parse reaction string using the default parser.

## psamm.datasource.sbml – SBML model parser

Parser for SBML model files.

**class** psamm.datasource.sbml.**CompartmentEntry**(*reader*, *root*, *filemark=None*)
Compartment entry in the SBML file

**properties**
All compartment properties as a dict.

**class** psamm.datasource.sbml.**FluxBoundEntry**(*reader*, *namespace*, *root*)
Flux bound defined with FBC V1.

Flux bounds defined with FBC V2 are instead encoded as `upper_flux` and `lower_flux` properties on the ReactionEntry objects.

**id**
Return ID of flux bound.

**name**
Return name of flux bound.

**operation**
Return the operation of the flux bound.

Returns one of LESS_EQUAL, GREATER_EQUAL or EQUAL.

**reaction**
Return reaction ID that the flux bound pertains to.

**value**
Return the flux bound value.

**class** psamm.datasource.sbml.**ObjectiveEntry**(*reader*, *namespace*, *root*)
Flux objective defined with FBC

**exception** psamm.datasource.sbml.**ParseError**
Error parsing SBML file

**class** psamm.datasource.sbml.**ReactionEntry**(*reader*, *root*, *filemark=None*)
Reaction entry in SBML file

**equation**
Reaction equation is a `Reaction` object

**id**
Reaction ID

**kinetic_law_reaction_parameters**
Iterator over the values of kinetic law reaction parameters

**name**
Reaction name

**properties**
All reaction properties as a dict

**reversible**
Whether the reaction is reversible

**class** psamm.datasource.sbml.**SBMLReader**(*file*, *strict=False*, *ignore_boundary=False*, *context=None*)
Reader of SBML model files

The constructor takes a file-like object which will be parsed as XML and then as SBML according to the specification. If the `strict` parameter is set to False, the parser will revert to a more lenient parsing which is required for many older models. This tries to mimic the inconsistencies employed by COBRA when parsing models.

If `ignore_boundary` is `True`, the species that are marked as boundary conditions will simply be dropped from the species list and from the reaction equations, and any boundary compartment will be dropped too.

**compartments**
Iterator over `CompartmentEntry` entries.

**create_model**()
Create model from reader.

**Returns** `psamm.datasource.native.NativeModel`.

**flux_bounds**
Iterator over `FluxBoundEntry`

**get_compartment**(*compartment_id*)
Return `CompartmentEntry` corresponding to id.

**get_objective**(*objective_id*)
Return `ObjectiveEntry` corresponding to objective_id

**get_reaction**(*reaction_id*)
Return `ReactionEntry` corresponding to reaction_id

**get_species**(*species_id*)
Return `SpeciesEntry` corresponding to species_id

**id**
Model ID

**name**
Model name

**objectives**
Iterator over `ObjectiveEntry`

**reactions**
Iterator over `ReactionEntries`

**species**
Iterator over `SpeciesEntries`

This will not yield boundary condition species if those are ignored.

**class** psamm.datasource.sbml.**SBMLWriter**(*cobra_flux_bounds=False*)
Writer of SBML files

**write_model**(*file*, *model*, *pretty=False*)
Write a given model to file

**class** psamm.datasource.sbml.**SpeciesEntry**(*reader*, *root*, *filemark=None*)
Species entry in the SBML file

**boundary**
Whether this compound is a boundary condition

**charge**
Species charge

**compartment**
Species compartment

**formula**
Species formula

**name**
Species name

**properties**
All species properties as a dict

## psamm.expression.affine – Affine expressions

Representations of affine expressions and variables.

These classes can be used to represent affine expressions and do manipulation and evaluation with substitutions of particular variables.

**class** psamm.expression.affine.**Expression**(**args*)
Represents an affine expression (e.g. 2x + 3y - z + 5)

**simplify**()
Return simplified expression.

If the expression is of the form 'x', the variable will be returned, and if the expression contains no variables, the offset will be returned as a number.

**substitute**(*mapping*)
Return expression with variables substituted

```
>>> Expression('x + 2y').substitute(
...     lambda v: {'y': -3}.get(v.symbol, v))
Expression('x - 6')
>>> Expression('x + 2y').substitute(
...     lambda v: {'y': Variable('z')}.get(v.symbol, v))
Expression('x + 2z')
```

**variables**()
Return iterator of variables in expression

**class** psamm.expression.affine.**Variable**(*symbol*)
Represents a variable in an expression

Equality of variables is based on the symbol.

**simplify**()
Return simplified expression

The simplified form of a variable is always the variable itself.

```
>>> Variable('x').simplify()
Variable('x')
```

**substitute**(*mapping*)
Return expression with variables substituted

```
>>> Variable('x').substitute(lambda v: {'x': 567}.get(v.symbol, v))
567
>>> Variable('x').substitute(lambda v: {'y': 42}.get(v.symbol, v))
Variable('x')
>>> Variable('x').substitute(
...     lambda v: {'x': 123, 'y': 56}.get(v.symbol, v))
123
```

**symbol**
Symbol of variable

```
>>> Variable('x').symbol
'x'
```

## psamm.expression.boolean – Boolean expressions

Representations of boolean expressions and variables.

These classes can be used to represent simple boolean expressions and do evaluation with substitutions of particular variables.

**class** psamm.expression.boolean.**And**(**args*)
Represents an AND term in an expression.

**class** psamm.expression.boolean.**Expression**(*arg*, *_vars=None*)
Boolean expression representation.

The expression can be constructed from an expression string of variables, operators ("and", "or") and parenthesis groups. For example,

```
>>> e = Expression('a and (b or c)')
```

**has_value**()
Return True if the expression has no variables.

**root**
Return root term, variable or boolean of the expression.

**substitute**(*mapping*)
Substitute variables using mapping function.

**value**
The value of the expression if fully evaluated.

**variables**
Immutable set of variables in the expression.

**class** psamm.expression.boolean.**Or**(**args*)
Represents an OR term in an expression.

**exception** psamm.expression.boolean.**ParseError**(**args*, ***kwargs*)
Signals error parsing boolean expression.

**exception** psamm.expression.boolean.**SubstitutionError**
Error substituting into expression.

**class** psamm.expression.boolean.**Variable**(*symbol*)
Represents a variable in a boolean expression

## psamm.fastcore – Fastcore (approximate consistent subset)

Fastcore module implementing the fastcore algorithm

This is an implementation of the algorithms described in *[Vlassis14]*.

**exception** psamm.fastcore.**FastcoreError**
Indicates an error while running Fastcore

psamm.fastcore.**fastcc**(*model*, *epsilon*, *solver*)
Check consistency of model reactions

Yields all reactions in the model that are not part of the consistent subset.

psamm.fastcore.**fastcc_consistent_subset**(*model*, *epsilon*, *solver*)
Return consistent subset of model

The largest consistent subset is returned as a set of reaction names.

psamm.fastcore.**fastcc_is_consistent**(*model*, *epsilon*, *solver*)
Quickly check whether model is consistent

Returns true if the model is consistent. If it is only necessary to know whether a model is consistent, this function is faster as it will return the result as soon as it finds a single inconsistent reaction.

psamm.fastcore.**fastcore**(*model*, *core*, *epsilon*, *solver*, *scaling=100000.0*, *weights={}*)
Find a flux consistent subnetwork containing the core subset

The result will contain the core subset and as few of the additional reactions as possible.

## psamm.fastgapfill – FastGapFill algorithm

Implementation of fastGapFill.

Described in *[Thiele14]*.

psamm.fastgapfill.**fastgapfill**(*model_extended*, *core*, *solver*, *weights={}*, *epsilon=1e-05*)
Run FastGapFill gap-filling algorithm by calling `psamm.fastcore.fastcore()`.

FastGapFill will try to find a minimum subset of reactions that includes the core reactions and it also has no blocked reactions. Return the set of reactions in the minimum subset. An extended model that includes artificial transport and exchange reactions can be generated by calling `create_extended_model()`.

**Parameters**

- **model** – `psamm.metabolicmodel.MetabolicModel`.
- **core** – reactions in the original metabolic model.
- **weights** – a weight dictionary for reactions in the model.
- **solver** – linear programming library to use.
- **epsilon** – float number, threshold for Fastcore algorithm.

## psamm.fluxanalysis – Constraint-based reaction flux analysis

Implementation of Flux Balance Analysis.

**exception** psamm.fluxanalysis.**FluxBalanceError**(**args*, ***kwargs*)
Error indicating that a flux balance cannot be solved.

**class** psamm.fluxanalysis.**FluxBalanceProblem**(*model*, *solver*)
Model as a flux optimization problem with steady state assumption.

Create a representation of the model as an LP optimization problem with steady state assumption, i.e. the concentrations of compounds are always zero.

The problem can be modified and solved as many times as needed. The flux of a reaction can be obtained after solving using `get_flux()`.

**Parameters**

- **model** – MetabolicModel to solve.
- **solver** – LP solver instance to use.

**add_thermodynamic**(*em=1000*)

Apply thermodynamic constraints to the model.

Adding these constraints restricts the solution space to only contain solutions that have no internal loops [Schilling00]. This is solved as a MILP problem as described in [Muller13]. The time to solve a problem with thermodynamic constraints is usually much longer than a normal FBA problem.

The em parameter is the upper bound on the delta mu reaction variables. This parameter has to be balanced based on the model size since setting the value too low can result in the correct solutions being infeasible and setting the value too high can result in numerical instability which again makes the correct solutions infeasible. The default value should work in all cases as long as the model is not unusually large.

**check_constraints**()

Optimize without objective to check that solution is possible.

Raises *FluxBalanceError* if no flux solution is possible.

**flux_bound**(*reaction*, *direction*)

Return the flux bound of the reaction.

Direction must be a positive number to obtain the upper bound or a negative number to obtain the lower bound. A value of inf or -inf is returned if the problem is unbounded.

**flux_expr**(*reaction*)

Get LP expression representing the reaction flux.

**get_flux**(*reaction*)

Get resulting flux value for reaction.

**get_flux_var**(*reaction*)

Get LP variable representing the reaction flux.

**max_min_l1**(*reaction*, *weights={}*)

Maximize flux of reaction then minimize the L1 norm.

During minimization the given reaction will be fixed at the maximum obtained from the first solution. If reaction is a dictionary object, each entry is interpreted as a weight on the objective for that reaction (non-existent reaction will have zero weight).

**maximize**(*reaction*)

Solve the model by maximizing the given reaction.

If reaction is a dictionary object, each entry is interpreted as a weight on the objective for that reaction (non-existent reaction will have zero weight).

**minimize_l1**(*weights={}*)

Solve the model by minimizing the L1 norm of the fluxes.

If the weights dictionary is given, the weighted L1 norm if minimized instead. The dictionary contains the weights of each reaction (default 1).

**prob**

Return the underlying LP problem.

This can be used to add additional constraints on the problem. Calling solve on the underlying problem is not guaranteed to work correctly, instead use the methods on this object that solves the problem or make a subclass with a method that calls _solve().

`psamm.fluxanalysis.`**`consistency_check`**(*model*, *subset*, *epsilon*, *tfba*, *solver*)

Check that reaction subset of model is consistent using FBA.

Yields all reactions that are *not* flux consistent. A reaction is consistent if there is at least one flux solution to the model that both respects the model constraints and also allows the reaction in question to have non-zero flux.

This can be determined by running FBA on each reaction in turn and checking whether the flux in the solution is non-zero. Since FBA only tries to maximize the flux (and the flux can be negative for reversible reactions), we have to try to both maximize and minimize the flux. An optimization to this method is implemented such that if checking one reaction results in flux in another unchecked reaction, that reaction will immediately be marked flux consistent.

**Parameters**

- **`model`** – MetabolicModel to check for consistency.
- **`subset`** – Subset of model reactions to check.
- **`epsilon`** – The threshold at which the flux is considered non-zero.
- **`tfba`** – If True enable thermodynamic constraints.
- **`solver`** – LP solver instance to use.

**Returns** An iterator of flux inconsistent reactions in the subset.

`psamm.fluxanalysis.`**`flux_balance`**(*model*, *reaction*, *tfba*, *solver*)

Run flux balance analysis on the given model.

Yields the reaction id and flux value for each reaction in the model.

This is a convenience function for sertting up and running the FluxBalanceProblem. If the FBA is solved for more than one parameter it is recommended to setup and reuse the FluxBalanceProblem manually for a speed up.

This is an implementation of flux balance analysis (FBA) as described in *[Orth10]* and *[Fell86]*.

**Parameters**

- **`model`** – MetabolicModel to solve.
- **`reaction`** – Reaction to maximize. If a dict is given, this instead represents the objective function weights on each reaction.
- **`tfba`** – If True enable thermodynamic constraints.
- **`solver`** – LP solver instance to use.

**Returns** Iterator over reaction ID and reaction flux pairs.

`psamm.fluxanalysis.`**`flux_minimization`**(*model*, *fixed*, *solver*, *weights={}*)

Minimize flux of all reactions while keeping certain fluxes fixed.

The fixed reactions are given in a dictionary as reaction id to value mapping. The weighted L1-norm of the fluxes is minimized.

**Parameters**

- **`model`** – MetabolicModel to solve.
- **`fixed`** – dict of additional lower bounds on reaction fluxes.
- **`solver`** – LP solver instance to use.
- **`weights`** – dict of weights on the L1-norm terms.

**Returns** An iterator of reaction ID and reaction flux pairs.

psamm.fluxanalysis.**flux_randomization**(*model*, *threshold*, *tfba*, *solver*)
Find a random flux solution on the boundary of the solution space.

The reactions in the threshold dictionary are constrained with the associated lower bound.

**Parameters**

- **model** – MetabolicModel to solve.
- **threshold** – dict of additional lower bounds on reaction fluxes.
- **tfba** – If True enable thermodynamic constraints.
- **solver** – LP solver instance to use.

**Returns** An iterator of reaction ID and reaction flux pairs.

psamm.fluxanalysis.**flux_variability**(*model*, *reactions*, *fixed*, *tfba*, *solver*)
Find the variability of each reaction while fixing certain fluxes.

Yields the reaction id, and a tuple of minimum and maximum value for each of the given reactions. The fixed reactions are given in a dictionary as a reaction id to value mapping.

This is an implementation of flux variability analysis (FVA) as described in *[Mahadevan03]*.

**Parameters**

- **model** – MetabolicModel to solve.
- **reactions** – Reactions on which to report variablity.
- **fixed** – dict of additional lower bounds on reaction fluxes.
- **tfba** – If True enable thermodynamic constraints.
- **solver** – LP solver instance to use.

**Returns** Iterator over pairs of reaction ID and bounds. Bounds are returned as pairs of lower and upper values.

## psamm.fluxcoupling – Flux coupling analysis

Flux coupling analysis

Described in *[Burgard04]*.

**class** psamm.fluxcoupling.**CouplingClass**
Enumeration of coupling types.

**class** psamm.fluxcoupling.**FluxCouplingProblem**(*model*, *bounded*, *solver*)
A specific flux coupling analysis applied to a metabolic model.

**Parameters**

- **model** – MetabolicModel to apply the flux coupling problem to
- **bounded** – Dictionary of reactions with minimum flux values
- **solver** – LP solver instance to use.

**solve**(*reaction_1*, *reaction_2*)
Return the flux coupling between two reactions

The flux coupling is returned as a tuple indicating the minimum and maximum value of the v1/v2 reaction flux ratio. A value of None as either the minimum or maximum indicates that the interval is unbounded in that direction.

psamm.fluxcoupling.**classify_coupling**(*coupling*)
Return a constant indicating the type of coupling.

Depending on the type of coupling, one of the constants from *CouplingClass* is returned.

**Parameters** **coupling** – Tuple of minimum and maximum flux ratio

# psamm.formula – Chemical compound formula

Parser and representation of chemical formulas.

Chemical formulas (*Formula*) are represented as a number of *FormulaElements* with associated counts. A *Formula* is itself a *FormulaElement* so a formula can contain subformulas. This allows some simple structure to be represented.

**class** psamm.formula.**Atom**(*symbol*)
Represent an atom in a chemical formula

```
>>> hydrogen = Atom.H
>>> oxygen = Atom.O
>>> str(oxygen | 2*hydrogen)
'H2O'
```

**symbol**
Atom symbol

```
>>> Atom.H.symbol
'H'
```

**class** psamm.formula.**Formula**(*values={}*)
Representation of a chemial formula

This is represented as a number of *FormulaElements* with associated counts.

```
>>> f = Formula({Atom.C: 6, Atom.H: 12, Atom.O: 6})
>>> str(f)
'C6H12O6'
```

**classmethod balance**(*lhs*, *rhs*)
Return formulas that need to be added to balance given formulas

Given complete formulas for right side and left side of a reaction, calculate formulas for the missing compounds on both sides. Return as a left, right tuple. Formulas can be flattened before balancing to disregard grouping structure.

**flattened**()
Return formula where subformulas have been flattened

```
>>> str(Formula.parse('(CH2)(CH2)2').flattened())
'C3H6'
```

**items**()
Iterate over (*FormulaElement*, value)-pairs

**classmethod parse** (*s*)
Parse a formula string (e.g. C6H10O2)

**class** psamm.formula.**FormulaElement**
Base class representing elements of a formula

**repeat** (*count*)
Repeat formula element by creating a subformula

**substitute** (*mapping*)
Return formula element with substitutions performed

**variables** ()
Iterator over variables in formula element

**class** psamm.formula.**Radical** (*symbol*)
Represents a radical or other unknown subformula

**symbol**
Radical symbol

```
>>> Radical('R1').symbol
'R1'
```

## psamm.gapfill – GapFind/GapFill

Identify blocked metabolites and possible reconstructions.

This implements a variant of the algorithms described in [Kumar07].

**exception** psamm.gapfill.**GapFillError**
Indicates an error while running GapFind/GapFill

psamm.gapfill.**gapfill** (*model*, *core*, *blocked*, *exclude*, *solver*, *epsilon=0.001*, *v_max=1000*, *weights={}*, *implicit_sinks=True*, *allow_bounds_expansion=False*)
Find a set of reactions to add such that no compounds are blocked.

Returns two iterators: first an iterator of reactions not in core, that were added to resolve the model. Second, an iterator of reactions in core that had flux bounds expanded (i.e. irreversible reactions become reversible). Similarly to GapFind, this method assumes, by default, implicit sinks for all compounds in the model so the only factor that influences whether a compound can be produced is the presence of the compounds needed to produce it. This means that the resulting model will not necessarily be flux consistent.

This method is implemented as a MILP-program. Therefore it may not be efficient for larger models.

**Parameters**

- **model** – MetabolicModel containing core reactions and reactions that can be added for gap-filling.
- **core** – The set of core (already present) reactions in the model.
- **blocked** – The compounds to unblock.
- **exclude** – Set of reactions in core to be excluded from gap-filling (e.g. biomass reaction).
- **solver** – MILP solver instance.
- **epsilon** – Threshold amount of a compound produced for it to not be considered blocked.
- **v_max** – Maximum flux.

- **weights** – Dictionary of weights for reactions. Weight is the penalty score for adding the reaction (non-core reactions) or expanding the flux bounds (all reactions).
- **implicit_sinks** – Whether implicit sinks for all compounds are included when gap-filling (traditional GapFill uses implicit sinks).
- **allow_bounds_expansion** – Allow flux bounds to be expanded at the cost of a penalty which can be specified using weights (traditional GapFill does not allow this). This includes turning irreversible reactions reversible.

psamm.gapfill.**gapfind**(*model*, *solver*, *epsilon=0.001*, *v_max=1000*, *implicit_sinks=True*)

Identify compounds in the model that cannot be produced.

Yields all compounds that cannot be produced. This method assumes implicit sinks for all compounds in the model so the only factor that influences whether a compound can be produced is the presence of the compounds needed to produce it.

Epsilon indicates the threshold amount of reaction flux for the products to be considered non-blocked. V_max indicates the maximum flux.

This method is implemented as a MILP-program. Therefore it may not be efficient for larger models.

**Parameters**

- **model** – MetabolicModel containing core reactions and reactions that can be added for gap-filling.
- **solver** – MILP solver instance.
- **epsilon** – Threshold amount of a compound produced for it to not be considered blocked.
- **v_max** – Maximum flux.
- **implicit_sinks** – Whether implicit sinks for all compounds are included when gap-filling (traditional GapFill uses implicit sinks).

## psamm.gapfilling – Gap-filling functions

Functionality related to gap-filling in general.

This module contains some general functions for preparing models for gap-filling. Specific gap-filling methods are implemented in the gapfill and fastgapfill modules.

psamm.gapfilling.**add_all_database_reactions**(*model*, *compartments*)

Add all reactions from database that occur in given compartments.

**Parameters** **model** – *psamm.metabolicmodel.MetabolicModel*.

psamm.gapfilling.**add_all_exchange_reactions**(*model*, *compartment*, *allow_duplicates=False*)

Add all exchange reactions to database and to model.

**Parameters** **model** – *psamm.metabolicmodel.MetabolicModel*.

psamm.gapfilling.**add_all_transport_reactions**(*model*, *boundaries*, *allow_duplicates=False*)

Add all transport reactions to database and to model.

Add transport reactions for all boundaries. Boundaries are defined by pairs (2-tuples) of compartment IDs. Transport reactions are added for all compounds in the model, not just for compounds in the two boundary compartments.

**Parameters**

- **model** – *psamm.metabolicmodel.MetabolicModel*.
- **boundaries** – Set of compartment boundary pairs.

**Returns** Set of IDs of reactions that were added.

psamm.gapfilling.**create_extended_model**(*model*, *db_penalty=None*, *ex_penalty=None*, *tp_penalty=None*, *penalties=None*)

Create an extended model for gap-filling.

Create a *psamm.metabolicmodel.MetabolicModel* with all reactions added (the reaction database in the model is taken to be the universal database) and also with artificial exchange and transport reactions added. Return the extended *psamm.metabolicmodel.MetabolicModel* and a weight dictionary for added reactions in that model.

**Parameters**

- **model** – *psamm.datasource.native.NativeModel*.
- **db_penalty** – penalty score for database reactions, default is *None*.
- **ex_penalty** – penalty score for exchange reactions, default is *None*.
- **tb_penalty** – penalty score for transport reactions, default is *None*.
- **penalties** – a dictionary of penalty scores for database reactions.

## psamm.lpsolver.cplex – CPLEX LP solver

Linear programming solver using Cplex.

**class** psamm.lpsolver.cplex.**Constraint**(*prob*, *name*)

Represents a constraint in a cplex.Problem

**class** psamm.lpsolver.cplex.**CplexRangedProperty**(*get_prop*, *doc=None*)

Decorator for translating Cplex ranged properties.

**class** psamm.lpsolver.cplex.**Problem**(***kwargs*)

Represents an LP-problem of a cplex.Solver

**add_linear_constraints**(**relations*)

Add constraints to the problem

Each constraint is represented by a Relation, and the expression in that relation can be a set expression.

**cplex**

The underlying Cplex object

**define**(**names*, ***kwargs*)

Define a variable in the problem.

Variables must be defined before they can be accessed by var() or set(). This function takes keyword arguments lower and upper to define the bounds of the variable (default: -inf to inf). The keyword argument types can be used to select the type of the variable (Continuous (default), Binary or Integer). Setting any variables different than Continuous will turn the problem into an MILP problem. Raises ValueError if a name is already defined.

**feasibility_tolerance**

Feasibility tolerance.

**has_variable**(*name*)

Check whether variable is defined in the model.

**integrality_tolerance**
Integrality tolerance.

**optimality_tolerance**
Optimality tolerance.

**set_linear_objective**(*expression*)
Set objective expression of the problem.

**set_objective**(*expression*)
Set objective expression of the problem.

**set_objective_sense**(*sense*)
Set type of problem (maximize or minimize)

**solve**(*sense=None*)
Solve problem

**class** psamm.lpsolver.cplex.**Result**(*prob*)
Represents the solution to a cplex.Problem

This object will be returned from the cplex.Problem.solve() method or by accessing the cplex.Problem.result property after solving a problem. This class should not be instantiated manually.

Result will evaluate to a boolean according to the success of the solution, so checking the truth value of the result will immediately indicate whether solving was successful.

**get_value**(*expression*)
Return value of expression.

**status**
Return string indicating the error encountered on failure

**success**
Return boolean indicating whether a solution was found

**unbounded**
Whether solution is unbounded

**class** psamm.lpsolver.cplex.**Solver**
Represents an LP-solver using Cplex

**create_problem**(***kwargs*)
Create a new LP-problem using the solver

## psamm.lpsolver.generic – Generic linear programming solver

Generic interface to LP solver instantiation.

**exception** psamm.lpsolver.generic.**RequirementsError**
Error resolving solver requirements

**class** psamm.lpsolver.generic.**Solver**(***kwargs*)
Generic solver interface based on requirements

Use the any of the following keyword arguments to restrict which underlying solver is used:

- *integer*: Use a solver that support integer variables (MILP)
- *rational*: Use a solver that returns rational results
- *quadratic*: Use a solver that supports quadratic objective/constraints

- *name*: Select a specific solver based on the name

**create_problem**()
Create a `Problem` instance

psamm.lpsolver.generic.**filter_solvers**(*solvers*, *requirements*)
Yield solvers that fullfil the requirements.

psamm.lpsolver.generic.**list_solvers**(*args=None*)
Entry point for listing available solvers.

psamm.lpsolver.generic.**parse_solver_setting**(*s*)
Parse a string containing a solver setting

## psamm.lpsolver.glpk – GLPK LP solver

Linear programming solver using GLPK.

**class** psamm.lpsolver.glpk.**Constraint**(*prob*, *name*)
Represents a constraint in a `Problem`.

**exception** psamm.lpsolver.glpk.**GLPKError**
Error from calling GLPK library.

**class** psamm.lpsolver.glpk.**MIPResult**(*prob*, *ret_val=0*)
Specialization of Result for MIP problems.

**class** psamm.lpsolver.glpk.**Problem**(***kwargs*)
Represents an LP-problem of a `Solver`.

**add_linear_constraints**(**relations*)
Add constraints to the problem.

Each constraint is represented by a Relation, and the expression in that relation can be a set expression.

**define**(**names*, ***kwargs*)
Define a variable in the problem.

Variables must be defined before they can be accessed by var() or set(). This function takes keyword arguments lower and upper to define the bounds of the variable (default: -inf to inf). The keyword argument types can be used to select the type of the variable (Continuous (default), Binary or Integer). Setting any variables different than Continuous will turn the problem into an MILP problem. Raises ValueError if a name is already defined.

**feasibility_tolerance**
Feasibility tolerance.

**glpk**
The underlying GLPK (SWIG) object.

**has_variable**(*name*)
Check whether variable is defined in the model.

**integrality_tolerance**
Integrality tolerance.

**optimality_tolerance**
Optimality tolerance.

**set_linear_objective**(*expression*)
Set objective of problem.

**set_objective**(*expression*)
Set objective of problem.

**set_objective_sense**(*sense*)
Set type of problem (maximize or minimize).

**solve**(*sense=None*)
Solve problem.

**class** psamm.lpsolver.glpk.**Result**(*prob*, *ret_val=0*)
Represents the solution to a *Problem*.

This object will be returned from the solve() method on *Problem* or by accessing the result property on *Problem* after solving a problem. This class should not be instantiated manually.

Result will evaluate to a boolean according to the success of the solution, so checking the truth value of the result will immediately indicate whether solving was successful.

**get_value**(*expression*)
Return value of expression.

**status**
Return string indicating the error encountered on failure.

**success**
Return boolean indicating whether a solution was found.

**unbounded**
Whether solution is unbounded

**class** psamm.lpsolver.glpk.**Solver**
Represents an LP-solver using GLPK.

**create_problem**(***kwargs*)
Create a new LP-problem using the solver.

## psamm.lpsolver.gurobi – Gurobi LP solver

Linear programming solver using Gurobi.

**class** psamm.lpsolver.gurobi.**Constraint**(*prob*, *name*)
Represents a constraint in a gurobi.Problem.

**class** psamm.lpsolver.gurobi.**Problem**(***kwargs*)
Represents an LP-problem of a gurobi.Solver.

**add_linear_constraints**(**relations*)
Add constraints to the problem.

Each constraint is represented by a Relation, and the expression in that relation can be a set expression.

**define**(**names*, ***kwargs*)
Define a variable in the problem.

Variables must be defined before they can be accessed by var() or set(). This function takes keyword arguments lower and upper to define the bounds of the variable (default: -inf to inf). The keyword argument types can be used to select the type of the variable (Continuous (default), Binary or Integer). Setting any variables different than Continuous will turn the problem into an MILP problem. Raises ValueError if a name is already defined.

**feasibility_tolerance**
Feasibility tolerance.

**gurobi**
The underlying Gurobi Model object.

**has_variable**(*name*)
Check whether variable is defined in the model.

**integrality_tolerance**
Integrality tolerance.

**optimality_tolerance**
Optimality tolerance.

**set_linear_objective**(*expression*)
Set linear objective of problem.

**set_objective**(*expression*)
Set linear objective of problem.

**set_objective_sense**(*sense*)
Set type of problem (maximize or minimize).

**solve**(*sense=None*)
Solve problem.

**class** psamm.lpsolver.gurobi.**Result**(*prob*)
Represents the solution to a gurobi.Problem.

This object will be returned from the gurobi.Problem.solve() method or by accessing the gurobi.Problem.result property after solving a problem. This class should not be instantiated manually.

Result will evaluate to a boolean according to the success of the solution, so checking the truth value of the result will immediately indicate whether solving was successful.

**get_value**(*expression*)
Return value of expression.

**status**
Return string indicating the error encountered on failure.

**success**
Return boolean indicating whether a solution was found.

**unbounded**
Whether solution is unbounded

**class** psamm.lpsolver.gurobi.**Solver**
Represents an LP-solver using Gurobi.

**create_problem**(***kwargs*)
Create a new LP-problem using the solver.

## psamm.lpsolver.lp – Linear programming problems

Base objects for representation of LP problems.

A linear programming problem is built from a number of constraints and an objective function. The objective function is a linear expression represented by *Expression*. The constraints are represented by *Relation*, created from a linear expression and a relation sense (equals, greater, less).

Expressions are built from variables defined in the *Problem* instance. In addition, an expression can contain a *VariableSet* instead of a single variable. This allows many similar expressions to be represented by one *Expression* instance which means that the LP problem can be constructed faster.

**class** psamm.lpsolver.lp.**Constraint**
Represents a constraint within an LP Problem

**delete**()
Remove constraint from Problem instance

**class** psamm.lpsolver.lp.**Expression**(*variables={}*, *offset=0*)
Represents a linear expression

The variables can be any hashable objects. If one or more variables are instead *VariableSets*, this will be taken to represent a set of expressions separately using a different element of the *VariableSet*.

```
>>> e = Expression({'x': 2, 'y': 3})
>>> str(e)
'2*x + 3*y'
```

In order to provide a more natural syntax for creating *Relations* the binary relation operators have been overloaded to return *Relation* instances.

```
>>> rel = Expression({'x': 2}) >= Expression({'y': 3})
>>> str(rel)
'2*x - 3*y >= 0'
```

**Warning:** Chained relations cannot be converted to multiple relations, e.g. `4 <= e <= 10` will fail to produce the intended relations!

**offset**
Value of the offset

**value_sets**()
Iterator of expression sets

This will yield an iterator of (variable, value)-pairs for each expression in the expression set (each equivalent to values()). If none of the variables is a set variable then a single iterator will be yielded.

**values**()
Return immutable view of (variable, value)-pairs in expression.

**variables**()
Return immutable view of variables in expression.

**exception** psamm.lpsolver.lp.**InvalidResultError**(*msg=None*)
Raised when a result that has been invalidated is accessed

**class** psamm.lpsolver.lp.**ObjectiveSense**
Enumeration of objective sense values

**class** psamm.lpsolver.lp.**Problem**
Representation of LP Problem instance

Variable names in the problem can be any hashable object. It is the responsibility of the solver interface to translate the object into a unique string if required by the underlying LP solver.

**add_linear_constraints**(**relations*)
Add constraints to the problem.

Each constraint is given as a *Relation*, and the expression in that relation can be a set expression. Returns a sequence of *Constraints*.

**define** (**names*, ***kwargs*)
Define a variable in the problem.

Variables must be defined before they can be accessed by var() or set(). This function takes keyword arguments lower and upper to define the bounds of the variable (default: -inf to inf). The keyword argument types can be used to select the type of the variable (Continuous (default), Binary or Integer). Setting any variables different than Continuous will turn the problem into an MILP problem. Raises ValueError if a name is already defined.

**expr** (*values*, *offset=0*)
Return the given dictionary of values as an *Expression*.

**has_variable** (*name*)
Check whether a variable is defined in the problem.

**namespace** (*names=None*, ***kwargs*)
Return namespace for this problem.

If names is given it should be an iterable of names to define in the namespace. Other keyword arguments can be specified which will be used to define the names given as well as being used as default parameters for names that are defined later.

```
>>> v = prob.namespace(name='v')
>>> v.define([1, 2, 5], lower=0, upper=10)
>>> prob.set_objective(v[1] + 3*v[2] - 5 * v[5])
```

**result**
Result of solved problem

**set** (*names*)
Return variables as a set expression.

This returns an *Expression* containing a *VariableSet*.

**set_linear_objective** (*expression*)
Set objective of the problem to the given *Expression*.

**set_objective** (*expression*)
Set objective of the problem to the given *Expression*.

**set_objective_sense** (*sense*)
Set type of problem (minimize or maximize)

**solve** (*sense=None*)
Solve problem and return result

**var** (*name*)
Return variable as an *Expression*.

class psamm.lpsolver.lp.**Product**
A tuple used to represent a variable product.

class psamm.lpsolver.lp.**RangedProperty** (*fget=None*, *fset=None*, *fdel=None*, *fmin=None*, *fmax=None*, *doc=None*)
Numeric property with minimum and maximum values.

The value attribute is used to get/set the actual value of the propery. The min/max attributes are used to get the bounds. The range is not automatically enforced when the value is set.

**class** psamm.lpsolver.lp.**Relation**(*sense*, *expression*)
Represents a binary relation (equation or inequality)

Relations can be equalities or inequalities. All relations of this type can be represented as a left-hand side expression and the type of relation. In this representation, the right-hand side is always zero.

**expression**
Left-hand side expression

**sense**
Type of relation (equality or inequality)

Can be one of Equal, Greater or Less, or one of the strict relations, StrictlyGreater or StrictlyLess.

**class** psamm.lpsolver.lp.**Result**
Result of solving an LP problem

The result is tied to the solver instance and is valid at least until the problem is solved again. If the problem has been solved again an *InvalidResultError* may be raised.

**get_value**(*expression*)
Get value of variable or expression in result

Expression can be an object defined as a name in the problem, in which case the corresponding value is simply returned. If expression is an actual *Expression* object, it will be evaluated using the values from the result.

**status**
String indicating the status of the problem result

**success**
Whether solution was optimal

**unbounded**
Whether solution is unbounded

**class** psamm.lpsolver.lp.**Solver**
Factory for LP Problem instances

**create_problem**()
Create a new *Problem* instance

**class** psamm.lpsolver.lp.**VariableNamespace**(*problem*, ***kwargs*)
Namespace for defining variables.

Namespaces are always unique even if two namespaces have the same name. Variables defined within a namespace will never clash with a global variable name or a name defined in another namespace. Namespaces should be created using the *Problem.namespace()*.

```
>>> v = prob.namespace(name='v')
>>> v.define([1, 2, 5], lower=0, upper=10)
>>> prob.set_objective(v[1] + 3*v[2] - 5 * v[5])
```

**define**(*names*, ***kwargs*)
Define variables within the namespace.

This is similar to *Problem.define()* except that names must be given as an iterable. This method accepts the same keyword arguments as *Problem.define()*.

**expr**(*items*)
Return the sum of each name multiplied by a coefficient.

```
>>> v = prob.namespace(name='v')
>>> v.define(['a', 'b', 'c'], lower=0, upper=10)
>>> prob.set_objective(v.expr([('a', 2), ('b', 1)]))
```

**set** (*names*)

Return a variable set of the given names in the namespace.

```
>>> v = prob.namespace(name='v')
>>> v.define([1, 2, 5], lower=0, upper=10)
>>> prob.add_linear_constraints(v.set([1, 2]) >= 4)
```

**sum** (*names*)

Return the sum of the given names in the namespace.

```
>>> v = prob.namespace(name='v')
>>> v.define([1, 2, 5], lower=0, upper=10)
>>> prob.set_objective(v.sum([2, 5]))  # v[2] + v[5]
```

**value** (*name*)

Return value of given variable in namespace.

```
>>> v = prob.namespace(name='v')
>>> v.define([1, 2, 5], lower=0, upper=10)
>>> prob.solve()
>>> print(v.value(2))
```

**class** psamm.lpsolver.lp.**VariableSet**

A tuple used to represent sets of variables.

**class** psamm.lpsolver.lp.**VariableType**

Enumeration of variable types

psamm.lpsolver.lp.**ranged_property** (*min=None*, *max=None*)

Decorator for creating ranged property with fixed bounds.

## psamm.lpsolver.qsoptex – QSopt_ex LP solver

Linear programming solver using QSopt_ex.

**class** psamm.lpsolver.qsoptex.**Constraint** (*prob*, *name*)

Represents a constraint in a qsoptex.Problem

**class** psamm.lpsolver.qsoptex.**Problem** (***kwargs*)

Represents an LP-problem of a qsoptex.Solver

**add_linear_constraints** (**relations*)

Add constraints to the problem

Each constraint is represented by a Relation, and the expression in that relation can be a set expression.

**define** (**names*, ***kwargs*)

Define a variable in the problem.

Variables must be defined before they can be accessed by var() or set(). This function takes keyword arguments lower and upper to define the bounds of the variable (default: -inf to inf). The keyword argument types can be used to select the type of the variable (only Continuous is supported). Raises ValueError if a name is already defined.

**feasibility_tolerance**
Feasibility tolerance.

**has_variable**(*name*)
Check whether variable is defined in the model.

**optimality_tolerance**
Optimality tolerance.

**qsoptex**
The underlying qsoptex.ExactProblem object

**set_linear_objective**(*expression*)
Set linear objective of problem

**set_objective**(*expression*)
Set linear objective of problem

**set_objective_sense**(*sense*)
Set type of problem (maximize or minimize)

**solve**(*sense=None*)
Solve problem

**class** psamm.lpsolver.qsoptex.**Result**(*prob*)
Represents the solution to a qsoptex.Problem

This object will be returned from the Problem.solve() method or by accessing the Problem.result property after solving a problem. This class should not be instantiated manually.

Result will evaluate to a boolean according to the success of the solution, so checking the truth value of the result will immediately indicate whether solving was successful.

**get_value**(*expression*)
Return value of expression

**status**
Return string indicating the error encountered on failure

**success**
Return boolean indicating whether a solution was found

**unbounded**
Whether the solution is unbounded

**class** psamm.lpsolver.qsoptex.**Solver**
Represents an LP solver using QSopt_ex

**create_problem**(***kwargs*)
Create a new LP-problem using the solver

## psamm.massconsistency – Mass consistency check

Mass consistency analysis of metabolic databases

A stoichiometric matrix, S, is said to be mass-consistent if S^Tm = 0 has a positive solution (m_i > 0). This corresponds to assigning a positive mass to each compound in the stoichiometric matrix and having each reaction preserve mass. Exchange reactions will have to be excluded from this check, as they are not able to preserve mass (by definition). In addition some databases may contain pseudo-compounds (e.g. “photon”) that also has to be excluded.

**exception** psamm.massconsistency.**MassConsistencyError**

Indicates an error while checking for mass consistency

psamm.massconsistency.**check_compound_consistency**(*database*, *solver*, *exchange=set([])*, *zeromass=set([])*)

Yield each compound in the database with assigned mass

Each compound will be assigned a mass and the number of compounds having a positive mass will be approximately maximized.

This is an implementation of the solution originally proposed by *[Gevorgyan08]* but using the new method proposed by *[Thiele14]* to avoid MILP constraints. This is similar to the way Fastcore avoids MILP contraints.

psamm.massconsistency.**check_reaction_consistency**(*database*, *solver*, *exchange=set([])*, *checked=set([])*, *zeromass=set([])*, *weights={}*)

Check inconsistent reactions by minimizing mass residuals

Return a reaction iterable, and compound iterable. The reaction iterable yields reaction ids and mass residuals. The compound iterable yields compound ids and mass assignments.

Each compound is assigned a mass of at least one, and the masses are balanced using the stoichiometric matrix. In addition, each reaction has a residual mass that is included in the mass balance equations. The L1-norm of the residuals is minimized. Reactions in the checked set are assumed to have been manually checked and therefore have the residual fixed at zero.

psamm.massconsistency.**is_consistent**(*database*, *solver*, *exchange=set([])*, *zeromass=set([])*)

Try to assign a positive mass to each compound

Return True if successful. The masses are simply constrained by m_i > 1 and finding a solution under these conditions proves that the database is mass consistent.

## psamm.metabolicmodel – Metabolic model representation

Representation of metabolic network models.

**class** psamm.metabolicmodel.**FlipableFluxBounds**(*view*, *reaction*)

FluxBounds object for a FlipableModelView.

This object is used internally in the FlipableModelView to represent the bounds of flux on a reaction that can be flipped.

**lower**

Lower bound

**upper**

Upper bound

**class** psamm.metabolicmodel.**FlipableLimitsView**(*view*)

Provides a limits view that flips with the flipable model view.

This object is used internally in FlipableModelView to expose a limits view that flips the bounds of all flipped reactions.

**class** psamm.metabolicmodel.**FlipableModelView**(*model*, *flipped=set([])*)

Proxy wrapper of model objects allowing a flipped set of reactions.

The proxy will forward all properties normally except that flipped reactions will appear to have stoichiometric values negated in the matrix property, and have bounds in the limits property flipped. This view is needed for some algorithms.

**class** psamm.metabolicmodel.**FlipableStoichiometricMatrixView**(*view*)
Provides a matrix view that flips with the flipable model view.

This object is used internally in FlipableModelView to expose a matrix view that negates the stoichiometric values of flipped reactions.

**class** psamm.metabolicmodel.**FluxBounds**(*model*, *reaction*)
Represents lower and upper bounds of flux as a mutable object

This object is used internally in the model representation. Changing the state of the object will change the underlying model parameters. Deleting a value will reset that value to the defaults.

**bounds**
Bounds as a tuple

**lower**
Lower bound

**upper**
Upper bound

**class** psamm.metabolicmodel.**LimitsView**(*model*)
Provides a view of the flux bounds defined in the model

This object is used internally in MetabolicModel to expose a dictonary view of the FluxBounds associated with the model reactions.

**class** psamm.metabolicmodel.**MetabolicModel**(*database*, *v_max=1000*)
Represents a metabolic model containing a set of reactions

The model contains a list of reactions referencing the reactions in the associated database.

**add_reaction**(*reaction_id*)
Add reaction to model

**copy**()
Return copy of model

**get_compound_reactions**(*compound_id*)
Iterate over all reaction ids the includes the given compound

**get_reaction_values**(*reaction_id*)
Return stoichiometric values of reaction as a dictionary

**is_exchange**(*reaction_id*)
Whether the given reaction is an exchange reaction.

**is_reversible**(*reaction_id*)
Whether the given reaction is reversible

**classmethod load_model**(*database*, *reaction_iter=None*, *exchange=None*, *limits=None*, *v_max=None*)
Get model from reaction name iterator.

The model will contain all reactions of the iterator.

**remove_reaction**(*reaction*)
Remove reaction from model

psamm.metabolicmodel.**create_exchange_id**(*existing_ids*, *compound*)
Create unique ID for exchange of compound.

psamm.metabolicmodel.**create_transport_id**(*existing_ids*, *compound_1*, *compound_2*)
Create unique ID for transport reaction of compounds.

# psamm.randomsparse – Find a random minimal network of model reactions

**class** psamm.randomsparse.**GeneDeletionStrategy**(*model*, *gene_assoc*)
Deleting genes strategy class.

When initializing instances of this class, *get_gene_associations()* can be called to obtain the gene association dict from the model.

**class** psamm.randomsparse.**ReactionDeletionStrategy**(*model*, *reaction_set=None*)
Deleting reactions strategy class.

When initializing instances of this class, *get_exchange_reactions()* can be useful if exchange reactions are used as the test set.

psamm.randomsparse.**get_exchange_reactions**(*model*)
Yield IDs of all exchange reactions from model.

This helper function would be useful when creating *ReactionDeletionStrategy* objects.

**Parameters** **model** – *psamm.metabolicmodel.MetabolicModel*.

psamm.randomsparse.**get_gene_associations**(*model*)
Create gene association for class *GeneDeletionStrategy*.

Return a dict mapping reaction IDs to *psamm.expression.boolean.Expression* objects, representing relationships between reactions and related genes. This helper function should be called when creating *GeneDeletionStrategy* objects.

**Parameters** **model** – *psamm.datasource.native.NativeModel*.

psamm.randomsparse.**random_sparse**(*strategy*, *prob*, *obj_reaction*, *flux_threshold*)
Find a random minimal network of model reactions.

Given a reaction to optimize and a threshold, delete entities randomly until the flux of the reaction to optimize falls under the threshold. Keep deleting until no more entities can be deleted. It works with two strategies: deleting reactions or deleting genes (reactions related to certain genes).

**Parameters**

- **strategy** – *ReactionDeletionStrategy* or *GeneDeletionStrategy*.
- **prob** – *psamm.fluxanalysis.FluxBalanceProblem*.
- **obj_reaction** – objective reactions to optimize.
- **flux_threshold** – threshold of max reaction flux.

# psamm.reaction – Reaction equations and compounds

Definitions related to reaction equations and parsing of such equations.

**class** psamm.reaction.**Compound**(*name*, *compartment=None*, *arguments=()*)
Represents a compound in a reaction equation

A compound is a named entity in the reaction equations representing a chemical compound. A compound can represent a generalized chemical entity (e.g. polyphosphate) and the arguments can be used to instantiate a specific chemical entity (e.g. polyphosphate(3)) by passing a number as an argument or a partially specified entity by passing an expression (e.g. polyphosphate(n)).

**arguments**
Expression argument for generalized compounds

**compartment**
Compartment of compound

**in_compartment** (*compartment*)
Return an instance of this compound in the specified compartment

```
>>> Compound('H+').in_compartment('e')
Compound('H+', 'e')
```

**name**
Name of compound

**translate** (*func*)
Translate compound name using given function

```
>>> Compound('Pb').translate(lambda x: x.lower())
Compound('pb')
```

**class** psamm.reaction.**Direction**
Directionality of reaction equation.

**flipped** ()
Return the flipped version of this direction.

**forward**
Whether this direction includes forward direction.

**reverse**
Whether this direction includes reverse direction.

**symbol**
Return string symbol for direction.

**class** psamm.reaction.**Reaction** (**args*)
Reaction equation representation.

Each compound is associated with a stoichiometric value and the reaction has a *Direction*. The reaction is created in one of the three following ways.

It can be created from a direction and two iterables of compound, value pairs representing the left-hand side and the right-hand side of the reaction:

```
>>> r = Reaction(Direction.Both, [(Compound('A'), 1), (Compound('B', 2))],
                                 [(Compound('C'), 1)])
>>> str(r)
'|A| + (2) |B| <=> |C|'
```

It can also be created from a single dict or iterable of compound, value pairs where the left-hand side compounds have negative values and the right-hand side compounds have positive values:

```
>>> r = Reaction(Direction.Forward, {
        Compound('A'): -1,
        Compound('B'): -2,
        Compound('C'): 1
})
>>> str(r)
'|A| + (2) |B| <=> |C|'
```

Lastly, the reaction can be created from an existing reaction object, creating a copy of that reaction.

```
>>> r = Reaction(Direction.Forward, {Compound('A'): -1, Compound('B'): 1})
>>> r2 = Reaction(r)
>>> str(r2)
'|A| => |B|'
```

Reactions can be added to produce combined reactions.

```
>>> r = Reaction(Direction.Forward, {Compound('A'): -1, Compound('B'): 1})
>>> s = Reaction(Direction.Forward, {Compound('B'): -1, Compound('C'): 1})
>>> str(r + s)
'|A| => |C|'
```

Reactions can also be multiplied by a number to produce a new reaction with scaled stoichiometry.

```
>>> r = Reaction(Direction.Forward, {Compound('A'): -1, Compound('B'): 2})
>>> str(2 * r)
'(2) |A| => (4) |B|'
```

Multiplying with a negative value will also flip the reaction, and as a special case, negating a reaction will simply flip it.

```
>>> r = Reaction(Direction.Forward, {Compound('A'): -1, Compound('B'): 2})
>>> str(r)
'|A| => (2) |B|'
>>> str(-r)
'(2) |B| <= |A|'
```

**compounds**
Sequence of compounds on both sides of the reaction equation

The sign of the stoichiometric values reflect whether the compound is on the left-hand side (negative) or the right-hand side (positive).

**direction**
Direction of reaction equation

**left**
Compounds on the left-hand side of the reaction equation.

**normalized**()
Return normalized reaction

The normalized reaction will be bidirectional or a forward reaction (i.e. reverse reactions are flipped).

**right**
Compounds on the right-hand side of the reaction equation.

**translated_compounds**(*translate*)
Return reaction where compound names have been translated.

For each compound the translate function is called with the compound name and the returned value is used as the new compound name. A new reaction is returned with the substituted compound names.

## psamm.util – Internal utilities

Various utilities.

**class** `psamm.util.`**`DictView`**(*d*)
An immutable wrapper around another dict-like object.

**class** `psamm.util.`**`FrozenOrderedSet`**(*seq=[]*)
An immutable set that retains insertion order.

**class** `psamm.util.`**`LoggerFile`**(*logger*, *level*)
File-like object that forwards to a logger.

The Cplex API takes a file-like object for writing log output. This class allows us to forward the Cplex messages to the Python logging system.

**`flush`**()
Flush stream.

This is a noop.

**`write`**(*s*)
Write message to logger.

**class** `psamm.util.`**`MaybeRelative`**(*s*)
Helper type for parsing possibly relative parameters.

```
>>> arg = MaybeRelative('40%')
>>> arg.reference = 200.0
>>> float(arg)
80.0
```

```
>>> arg = MaybeRelative('24.5')
>>> arg.reference = 150.0
>>> float(arg)
24.5
```

**`reference`**
The reference used for converting to absolute value.

**`relative`**
Whether the parsed number was relative.

`psamm.util.`**`convex_cardinality_relaxed`**(*f*, *epsilon=1e-05*)
Transform L1-norm optimization function into cardinality optimization.

The given function must optimize a convex problem with a weighted L1-norm as the objective. The transformed function will apply the iterated weighted L1 heuristic to approximately optimize the cardinality of the solution. This method is described by S. Boyd, "L1-norm norm methods for convex cardinality problems." Lecture Notes for EE364b, Stanford University, 2007. Available online at www.stanford.edu/class/ee364b/.

The given function must take an optional keyword parameter weights (dictionary), and the weights must be set to one if not specified. The function must return the non-weighted solution as an iterator over (identifier, value)-tuples, either directly or as the first element of a tuple.

`psamm.util.`**`create_unique_id`**(*prefix*, *existing_ids*)
Return a unique string ID from the prefix.

First check if the prefix is itself a unique ID in the set-like parameter existing_ids. If not, try integers in ascending order appended to the prefix until a unique ID is found.

`psamm.util.`**`git_try_describe`**(*repo_path*)
Try to describe the current commit of a Git repository.

Return a string containing a string with the commit ID and/or a base tag, if successful. Otherwise, return None.

CHAPTER 9

# References

CHAPTER 10

# Indices and tables

- genindex
- modindex
- search

# Bibliography

[Burgard04] Burgard AP, Nikolaev E V, Schilling CH, Maranas CD. Flux coupling analysis of genome-scale metabolic network reconstructions. Genome Res. 2004;14: 301–312. doi:10.1101/gr.1926504.

[Edwards00] Edwards JS, Palsson BO. Robustness Analysis of the Escherichia coli Metabolic Network. Biotechnol Prog. American Chemical Society; 2000;16: 927–939. doi:10.1021/bp0000712.

[Fell86] Fell DA, Small JR. Fat synthesis in adipose tissue. An examination of stoichiometric constraints. Biochem J. 1986;238. doi:10.1042/bj2380781.

[Gevorgyan08] Gevorgyan A, Poolman MG, Fell DA. Detection of stoichiometric inconsistencies in biomolecular models. Bioinformatics. 2008;24: 2245–2251. doi:10.1093/bioinformatics/btn425.

[Kumar07] Satish Kumar V, Dasika MS, Maranas CD. Optimization based automated curation of metabolic reconstructions. BMC Bioinformatics. 2007;8: 212. doi:10.1186/1471-2105-8-212.

[Mahadevan03] Mahadevan R, Schilling CH. The effects of alternate optimal solutions in constraint-based genome-scale metabolic models. Metab Eng. 2003;5: 264–276. doi:10.1016/j.ymben.2003.09.002.

[Muller13] Müller AC, Bockmayr A. Fast thermodynamically constrained flux variability analysis. Bioinformatics. 2013;29: 903–909. doi:10.1093/bioinformatics/btt059.

[Orth10] Orth JD, Thiele I, Palsson BØ. What is flux balance analysis? Nat Biotechnol. Nature Publishing Group; 2010;28: 245–8. doi:10.1038/nbt.1614.

[Schilling00] Schilling CH, Letscher D, Palsson BO. Theory for the systemic definition of metabolic pathways and their use in interpreting metabolic function from a pathway-oriented perspective. J Theor Biol. 2000;203: 229–48. doi:10.1006/jtbi.2000.1073.

[Thiele14] Thiele I, Vlassis N, Fleming RMT. fastGapFill: efficient gap filling in metabolic networks. Bioinformatics. 2014;30: 2529–31. doi:10.1093/bioinformatics/btu321.

[Vlassis14] Vlassis N, Pacheco MP, Sauter T. Fast Reconstruction of Compact Context-Specific Metabolic Network Models. PLoS Comput Biol. 2014;10: e1003424. doi:10.1371/journal.pcbi.1003424.

[Orth13] Orth JD, Palsson BØ, Fleming RMT. Reconstruction and Use of Microbial Metabolic Networks: the Core Escherichia coli Metabolic Model as an Educational Guide. EcoSal Plus. asm Pub2Web; 2013;1. doi:10.1128/ecosalplus.10.2.1.

[Orth11] Orth JD, Conrad TM, Na J, Lerman JA, Nam H, Feist AM, et al. A comprehensive genome-scale reconstruction of Escherichia coli metabolism–2011. Mol Syst Biol. EMBO Press; 2011;7: 535. doi:10.1038/msb.2011.65.

# Python Module Index

## p

psamm.balancecheck, 61
psamm.command, 62
psamm.database, 63
psamm.datasource.context, 64
psamm.datasource.entry, 64
psamm.datasource.kegg, 65
psamm.datasource.modelseed, 66
psamm.datasource.native, 66
psamm.datasource.reaction, 72
psamm.datasource.sbml, 72
psamm.expression.affine, 75
psamm.expression.boolean, 76
psamm.fastcore, 76
psamm.fastgapfill, 77
psamm.fluxanalysis, 77
psamm.fluxcoupling, 80
psamm.formula, 81
psamm.gapfill, 82
psamm.gapfilling, 83
psamm.lpsolver.cplex, 84
psamm.lpsolver.generic, 85
psamm.lpsolver.glpk, 86
psamm.lpsolver.gurobi, 87
psamm.lpsolver.lp, 88
psamm.lpsolver.qsoptex, 92
psamm.massconsistency, 93
psamm.metabolicmodel, 94
psamm.randomsparse, 96
psamm.reaction, 96
psamm.util, 98

# Index

## A


add_all_database_reactions() (in module psamm.gapfilling), 83
add_all_exchange_reactions() (in module psamm.gapfilling), 83
add_all_transport_reactions() (in module psamm.gapfilling), 83
add_linear_constraints() (psamm.lpsolver.cplex.Problem method), 84
add_linear_constraints() (psamm.lpsolver.glpk.Problem method), 86
add_linear_constraints() (psamm.lpsolver.gurobi.Problem method), 87
add_linear_constraints() (psamm.lpsolver.lp.Problem method), 89
add_linear_constraints() (psamm.lpsolver.qsoptex.Problem method), 92
add_reaction() (psamm.metabolicmodel.MetabolicModel method), 95
add_thermodynamic() (psamm.fluxanalysis.FluxBalanceProblem method), 78
And (class in psamm.expression.boolean), 76
argument_error() (psamm.command.Command method), 62
arguments (psamm.reaction.Compound attribute), 96
Atom (class in psamm.formula), 81


## B


balance() (psamm.formula.Formula class method), 81
biomass_reaction (psamm.datasource.native.ModelReader attribute), 67
biomass_reaction (psamm.datasource.native.NativeModel attribute), 68
boundary (psamm.datasource.sbml.SpeciesEntry attribute), 74
bounds (psamm.metabolicmodel.FluxBounds attribute), 95


## C


ChainedDatabase (class in psamm.database), 63
charge (psamm.datasource.entry.CompoundEntry attribute), 64
charge (psamm.datasource.sbml.SpeciesEntry attribute), 74
charge_balance() (in module psamm.balancecheck), 61
check_compound_consistency() (in module psamm.massconsistency), 94
check_constraints() (psamm.fluxanalysis.FluxBalanceProblem method), 78
check_reaction_consistency() (in module psamm.massconsistency), 94
classify_coupling() (in module psamm.fluxcoupling), 81
Command (class in psamm.command), 62
CommandError, 62
compartment (psamm.datasource.sbml.SpeciesEntry attribute), 74
compartment (psamm.reaction.Compound attribute), 97
compartment_boundaries (psamm.datasource.native.NativeModel attribute), 68
CompartmentEntry (class in psamm.datasource.entry), 64
CompartmentEntry (class in psamm.datasource.sbml), 72
compartments (psamm.database.MetabolicDatabase attribute), 63
compartments (psamm.datasource.native.NativeModel attribute), 68
compartments (psamm.datasource.sbml.SBMLReader attribute), 73
Compound (class in psamm.reaction), 96
CompoundEntry (class in psamm.datasource.entry), 64
CompoundEntry (class in psamm.datasource.kegg), 65
CompoundEntry (class in psamm.datasource.modelseed), 66
CompoundMapper (class in psamm.datasource.kegg), 66
compounds (psamm.database.MetabolicDatabase attribute), 63
compounds (psamm.datasource.native.NativeModel attribute), 69
compounds (psamm.reaction.Reaction attribute), 98
consistency_check() (in module psamm.fluxanalysis), 78

Constraint (class in psamm.lpsolver.cplex), 84
Constraint (class in psamm.lpsolver.glpk), 86
Constraint (class in psamm.lpsolver.gurobi), 87
Constraint (class in psamm.lpsolver.lp), 89
Constraint (class in psamm.lpsolver.qsoptex), 92
ContextError, 64
convert_compartment_entry() (psamm.datasource.native.ModelWriter method), 68
convert_compound_entry() (psamm.datasource.native.ModelWriter method), 68
convert_reaction_entry() (psamm.datasource.native.ModelWriter method), 68
convex_cardinality_relaxed() (in module psamm.util), 99
copy() (psamm.metabolicmodel.MetabolicModel method), 95
CouplingClass (class in psamm.fluxcoupling), 80
cplex (psamm.lpsolver.cplex.Problem attribute), 84
CplexRangedProperty (class in psamm.lpsolver.cplex), 84
create_exchange_id() (in module psamm.metabolicmodel), 95
create_extended_model() (in module psamm.gapfilling), 84
create_metabolic_model() (psamm.datasource.native.NativeModel method), 69
create_model() (psamm.datasource.native.ModelReader method), 67
create_model() (psamm.datasource.sbml.SBMLReader method), 74
create_problem() (psamm.lpsolver.cplex.Solver method), 85
create_problem() (psamm.lpsolver.generic.Solver method), 86
create_problem() (psamm.lpsolver.glpk.Solver method), 87
create_problem() (psamm.lpsolver.gurobi.Solver method), 88
create_problem() (psamm.lpsolver.lp.Solver method), 91
create_problem() (psamm.lpsolver.qsoptex.Solver method), 93
create_transport_id() (in module psamm.metabolicmodel), 95
create_unique_id() (in module psamm.util), 99


## D


decode_name() (in module psamm.datasource.modelseed), 66
default_compartment (psamm.datasource.native.ModelReader attribute), 67
default_compartment (psamm.datasource.native.NativeModel attribute), 69
default_flux_limit (psamm.datasource.native.ModelReader attribute), 67
default_flux_limit (psamm.datasource.native.NativeModel attribute), 69
define() (psamm.lpsolver.cplex.Problem method), 84
define() (psamm.lpsolver.glpk.Problem method), 86
define() (psamm.lpsolver.gurobi.Problem method), 87
define() (psamm.lpsolver.lp.Problem method), 90
define() (psamm.lpsolver.lp.VariableNamespace method), 91
define() (psamm.lpsolver.qsoptex.Problem method), 92
delete() (psamm.lpsolver.lp.Constraint method), 89
DictCompartmentEntry (class in psamm.datasource.entry), 64
DictCompoundEntry (class in psamm.datasource.entry), 65
DictDatabase (class in psamm.database), 63
DictReactionEntry (class in psamm.datasource.entry), 65
DictView (class in psamm.util), 98
Direction (class in psamm.reaction), 97
direction (psamm.reaction.Reaction attribute), 98


## E


equation (psamm.datasource.entry.ReactionEntry attribute), 65
equation (psamm.datasource.sbml.ReactionEntry attribute), 73
exchange (psamm.datasource.native.NativeModel attribute), 69
ExecutorError, 62
expr() (psamm.lpsolver.lp.Problem method), 90
expr() (psamm.lpsolver.lp.VariableNamespace method), 91
Expression (class in psamm.expression.affine), 75
Expression (class in psamm.expression.boolean), 76
Expression (class in psamm.lpsolver.lp), 89
expression (psamm.lpsolver.lp.Relation attribute), 91
extracellular_compartment (psamm.datasource.native.ModelReader attribute), 67
extracellular_compartment (psamm.datasource.native.NativeModel attribute), 69


## F


fail() (psamm.command.Command method), 62
fastcc() (in module psamm.fastcore), 76
fastcc_consistent_subset() (in module psamm.fastcore), 77
fastcc_is_consistent() (in module psamm.fastcore), 77
fastcore() (in module psamm.fastcore), 77
FastcoreError, 76
fastgapfill() (in module psamm.fastgapfill), 77

feasibility_tolerance (psamm.lpsolver.cplex.Problem attribute), 84
feasibility_tolerance (psamm.lpsolver.glpk.Problem attribute), 86
feasibility_tolerance (psamm.lpsolver.gurobi.Problem attribute), 87
feasibility_tolerance (psamm.lpsolver.qsoptex.Problem attribute), 92
FileMark (class in psamm.datasource.context), 64
filemark (psamm.datasource.entry.ModelEntry attribute), 65
FilePathContext (class in psamm.datasource.context), 64
FilePrefixAppendAction (class in psamm.command), 62
filter_solvers() (in module psamm.lpsolver.generic), 86
flattened() (psamm.formula.Formula method), 81
FlipableFluxBounds (class in psamm.metabolicmodel), 94
FlipableLimitsView (class in psamm.metabolicmodel), 94
FlipableModelView (class in psamm.metabolicmodel), 94
FlipableStoichiometricMatrixView (class in psamm.metabolicmodel), 94
flipped() (psamm.reaction.Direction method), 97
float_constructor() (in module psamm.datasource.native), 69
flush() (psamm.util.LoggerFile method), 99
flux_balance() (in module psamm.fluxanalysis), 79
flux_bound() (psamm.fluxanalysis.FluxBalanceProblem method), 78
flux_bounds (psamm.datasource.sbml.SBMLReader attribute), 74
flux_expr() (psamm.fluxanalysis.FluxBalanceProblem method), 78
flux_minimization() (in module psamm.fluxanalysis), 79
flux_randomization() (in module psamm.fluxanalysis), 79
flux_variability() (in module psamm.fluxanalysis), 80
FluxBalanceError, 77
FluxBalanceProblem (class in psamm.fluxanalysis), 77
FluxBoundEntry (class in psamm.datasource.sbml), 73
FluxBounds (class in psamm.metabolicmodel), 95
FluxCouplingProblem (class in psamm.fluxcoupling), 80
Formula (class in psamm.formula), 81
formula (psamm.datasource.entry.CompoundEntry attribute), 64
formula (psamm.datasource.sbml.SpeciesEntry attribute), 74
formula_balance() (in module psamm.balancecheck), 61
FormulaElement (class in psamm.formula), 82
forward (psamm.reaction.Direction attribute), 97
FrozenOrderedSet (class in psamm.util), 99


## G


gapfill() (in module psamm.gapfill), 82
GapFillError, 82
gapfind() (in module psamm.gapfill), 83
GeneDeletionStrategy (class in psamm.randomsparse), 96
genes (psamm.datasource.entry.ReactionEntry attribute), 65
get_compartment() (psamm.datasource.sbml.SBMLReader method), 74
get_compound_reactions() (psamm.database.MetabolicDatabase method), 63
get_compound_reactions() (psamm.metabolicmodel.MetabolicModel method), 95
get_exchange_reactions() (in module psamm.randomsparse), 96
get_flux() (psamm.fluxanalysis.FluxBalanceProblem method), 78
get_flux_var() (psamm.fluxanalysis.FluxBalanceProblem method), 78
get_gene_associations() (in module psamm.randomsparse), 96
get_objective() (psamm.datasource.sbml.SBMLReader method), 74
get_reaction() (psamm.database.MetabolicDatabase method), 63
get_reaction() (psamm.datasource.sbml.SBMLReader method), 74
get_reaction_values() (psamm.database.MetabolicDatabase method), 63
get_reaction_values() (psamm.metabolicmodel.MetabolicModel method), 95
get_species() (psamm.datasource.sbml.SBMLReader method), 74
get_value() (psamm.lpsolver.cplex.Result method), 85
get_value() (psamm.lpsolver.glpk.Result method), 87
get_value() (psamm.lpsolver.gurobi.Result method), 88
get_value() (psamm.lpsolver.lp.Result method), 91
get_value() (psamm.lpsolver.qsoptex.Result method), 93
git_try_describe() (in module psamm.util), 99
glpk (psamm.lpsolver.glpk.Problem attribute), 86
GLPKError, 86
gurobi (psamm.lpsolver.gurobi.Problem attribute), 88


## H


has_model_definition() (psamm.datasource.native.ModelReader method), 67
has_reaction() (psamm.database.MetabolicDatabase method), 63
has_value() (psamm.expression.boolean.Expression method), 76
has_variable() (psamm.lpsolver.cplex.Problem method), 84
has_variable() (psamm.lpsolver.glpk.Problem method), 86
has_variable() (psamm.lpsolver.gurobi.Problem method), 88

has_variable() (psamm.lpsolver.lp.Problem method), 90
has_variable() (psamm.lpsolver.qsoptex.Problem method), 93

## I

id (psamm.datasource.entry.ModelEntry attribute), 65
id (psamm.datasource.sbml.FluxBoundEntry attribute), 73
id (psamm.datasource.sbml.ReactionEntry attribute), 73
id (psamm.datasource.sbml.SBMLReader attribute), 74
in_compartment() (psamm.reaction.Compound method), 97
init_parser() (psamm.command.Command class method), 62
integrality_tolerance (psamm.lpsolver.cplex.Problem attribute), 84
integrality_tolerance (psamm.lpsolver.glpk.Problem attribute), 86
integrality_tolerance (psamm.lpsolver.gurobi.Problem attribute), 88
InvalidResultError, 89
is_consistent() (in module psamm.massconsistency), 94
is_exchange() (psamm.metabolicmodel.MetabolicModel method), 95
is_reversible() (psamm.database.MetabolicDatabase method), 63
is_reversible() (psamm.metabolicmodel.MetabolicModel method), 95
items() (psamm.formula.Formula method), 81

## K

KEGGEntry (class in psamm.datasource.kegg), 66
kinetic_law_reaction_parameters (psamm.datasource.sbml.ReactionEntry attribute), 73

## L

left (psamm.reaction.Reaction attribute), 98
limits (psamm.datasource.native.NativeModel attribute), 69
LimitsView (class in psamm.metabolicmodel), 95
list_solvers() (in module psamm.lpsolver.generic), 86
load_model() (psamm.metabolicmodel.MetabolicModel class method), 95
LoggerFile (class in psamm.util), 99
LoopRemovalMixin (class in psamm.command), 62
lower (psamm.metabolicmodel.FlipableFluxBounds attribute), 94
lower (psamm.metabolicmodel.FluxBounds attribute), 95

## M

main() (in module psamm.command), 63
MassConsistencyError, 93
matrix (psamm.database.MetabolicDatabase attribute), 63
max_min_l1() (psamm.fluxanalysis.FluxBalanceProblem method), 78
maximize() (psamm.fluxanalysis.FluxBalanceProblem method), 78
MaybeRelative (class in psamm.util), 99
MetabolicDatabase (class in psamm.database), 63
MetabolicMixin (class in psamm.command), 62
MetabolicModel (class in psamm.metabolicmodel), 95
minimize_l1() (psamm.fluxanalysis.FluxBalanceProblem method), 78
MIPResult (class in psamm.lpsolver.glpk), 86
model (psamm.datasource.native.NativeModel attribute), 69
ModelEntry (class in psamm.datasource.entry), 65
ModelReader (class in psamm.datasource.native), 66
ModelWriter (class in psamm.datasource.native), 68

## N

name (psamm.datasource.entry.ModelEntry attribute), 65
name (psamm.datasource.native.ModelReader attribute), 67
name (psamm.datasource.native.NativeModel attribute), 69
name (psamm.datasource.sbml.FluxBoundEntry attribute), 73
name (psamm.datasource.sbml.ReactionEntry attribute), 73
name (psamm.datasource.sbml.SBMLReader attribute), 74
name (psamm.datasource.sbml.SpeciesEntry attribute), 74
name (psamm.reaction.Compound attribute), 97
namespace() (psamm.lpsolver.lp.Problem method), 90
NativeModel (class in psamm.datasource.native), 68
normalized() (psamm.reaction.Reaction method), 98

## O

ObjectiveEntry (class in psamm.datasource.sbml), 73
ObjectiveMixin (class in psamm.command), 62
objectives (psamm.datasource.sbml.SBMLReader attribute), 74
ObjectiveSense (class in psamm.lpsolver.lp), 89
offset (psamm.lpsolver.lp.Expression attribute), 89
operation (psamm.datasource.sbml.FluxBoundEntry attribute), 73
optimality_tolerance (psamm.lpsolver.cplex.Problem attribute), 85
optimality_tolerance (psamm.lpsolver.glpk.Problem attribute), 86
optimality_tolerance (psamm.lpsolver.gurobi.Problem attribute), 88
optimality_tolerance (psamm.lpsolver.qsoptex.Problem attribute), 93

Or (class in psamm.expression.boolean), 76

## P

ParallelTaskMixin (class in psamm.command), 62
parse() (psamm.datasource.reaction.ReactionParser method), 72
parse() (psamm.formula.Formula class method), 81
parse_compartments() (psamm.datasource.native.ModelReader method), 67
parse_compound() (in module psamm.datasource.native), 69
parse_compound() (in module psamm.datasource.reaction), 72
parse_compound_count() (in module psamm.datasource.reaction), 72
parse_compound_file() (in module psamm.datasource.kegg), 66
parse_compound_file() (in module psamm.datasource.modelseed), 66
parse_compound_file() (in module psamm.datasource.native), 69
parse_compound_list() (in module psamm.datasource.native), 69
parse_compound_table_file() (in module psamm.datasource.native), 69
parse_compound_yaml_file() (in module psamm.datasource.native), 69
parse_compounds() (psamm.datasource.native.ModelReader method), 67
parse_exchange() (in module psamm.datasource.native), 70
parse_exchange() (psamm.datasource.native.ModelReader method), 67
parse_exchange_file() (in module psamm.datasource.native), 70
parse_exchange_list() (in module psamm.datasource.native), 70
parse_exchange_table_file() (in module psamm.datasource.native), 70
parse_exchange_yaml_file() (in module psamm.datasource.native), 70
parse_kegg_entries() (in module psamm.datasource.kegg), 66
parse_limit() (in module psamm.datasource.native), 70
parse_limits() (psamm.datasource.native.ModelReader method), 67
parse_limits_file() (in module psamm.datasource.native), 70
parse_limits_list() (in module psamm.datasource.native), 70
parse_limits_table_file() (in module psamm.datasource.native), 70
parse_limits_yaml_file() (in module psamm.datasource.native), 70
parse_medium() (in module psamm.datasource.native), 70
parse_medium() (psamm.datasource.native.ModelReader method), 67
parse_medium_file() (in module psamm.datasource.native), 70
parse_medium_list() (in module psamm.datasource.native), 70
parse_medium_table_file() (in module psamm.datasource.native), 71
parse_medium_yaml_file() (in module psamm.datasource.native), 71
parse_model() (psamm.datasource.native.ModelReader method), 67
parse_model_file() (in module psamm.datasource.native), 71
parse_model_group() (in module psamm.datasource.native), 71
parse_model_group_list() (in module psamm.datasource.native), 71
parse_model_table_file() (in module psamm.datasource.native), 71
parse_model_yaml_file() (in module psamm.datasource.native), 71
parse_reaction() (in module psamm.datasource.kegg), 66
parse_reaction() (in module psamm.datasource.native), 71
parse_reaction() (in module psamm.datasource.reaction), 72
parse_reaction_equation() (in module psamm.datasource.native), 71
parse_reaction_equation_string() (in module psamm.datasource.native), 71
parse_reaction_file() (in module psamm.datasource.kegg), 66
parse_reaction_file() (in module psamm.datasource.native), 71
parse_reaction_list() (in module psamm.datasource.native), 71
parse_reaction_table_file() (in module psamm.datasource.native), 72
parse_reaction_yaml_file() (in module psamm.datasource.native), 72
parse_reactions() (psamm.datasource.native.ModelReader method), 67
parse_solver_setting() (in module psamm.lpsolver.generic), 86
ParseError, 66, 69, 72, 73, 76
prob (psamm.fluxanalysis.FluxBalanceProblem attribute), 78
Problem (class in psamm.lpsolver.cplex), 84
Problem (class in psamm.lpsolver.glpk), 86
Problem (class in psamm.lpsolver.gurobi), 87
Problem (class in psamm.lpsolver.lp), 89

Problem (class in psamm.lpsolver.qsoptex), 92
Product (class in psamm.lpsolver.lp), 90
properties (psamm.datasource.entry.ModelEntry attribute), 65
properties (psamm.datasource.sbml.CompartmentEntry attribute), 72
properties (psamm.datasource.sbml.ReactionEntry attribute), 73
properties (psamm.datasource.sbml.SpeciesEntry attribute), 74
psamm.balancecheck (module), 61
psamm.command (module), 62
psamm.database (module), 63
psamm.datasource.context (module), 64
psamm.datasource.entry (module), 64
psamm.datasource.kegg (module), 65
psamm.datasource.modelseed (module), 66
psamm.datasource.native (module), 66
psamm.datasource.reaction (module), 72
psamm.datasource.sbml (module), 72
psamm.expression.affine (module), 75
psamm.expression.boolean (module), 76
psamm.fastcore (module), 76
psamm.fastgapfill (module), 77
psamm.fluxanalysis (module), 77
psamm.fluxcoupling (module), 80
psamm.formula (module), 81
psamm.gapfill (module), 82
psamm.gapfilling (module), 83
psamm.lpsolver.cplex (module), 84
psamm.lpsolver.generic (module), 85
psamm.lpsolver.glpk (module), 86
psamm.lpsolver.gurobi (module), 87
psamm.lpsolver.lp (module), 88
psamm.lpsolver.qsoptex (module), 92
psamm.massconsistency (module), 93
psamm.metabolicmodel (module), 94
psamm.randomsparse (module), 96
psamm.reaction (module), 96
psamm.util (module), 98

## Q

qsoptex (psamm.lpsolver.qsoptex.Problem attribute), 93

## R

Radical (class in psamm.formula), 82
random_sparse() (in module psamm.randomsparse), 96
ranged_property() (in module psamm.lpsolver.lp), 92
RangedProperty (class in psamm.lpsolver.lp), 90
Reaction (class in psamm.reaction), 97
reaction (psamm.datasource.sbml.FluxBoundEntry attribute), 73
reaction_charge() (in module psamm.balancecheck), 61
reaction_formula() (in module psamm.balancecheck), 61
ReactionDeletionStrategy (class in psamm.randomsparse), 96
ReactionEntry (class in psamm.datasource.entry), 65
ReactionEntry (class in psamm.datasource.kegg), 66
ReactionEntry (class in psamm.datasource.sbml), 73
ReactionMapper (class in psamm.datasource.kegg), 66
ReactionParser (class in psamm.datasource.reaction), 72
reactions (psamm.database.MetabolicDatabase attribute), 64
reactions (psamm.datasource.native.NativeModel attribute), 69
reactions (psamm.datasource.sbml.SBMLReader attribute), 74
reader_from_path() (psamm.datasource.native.ModelReader class method), 67
reference (psamm.util.MaybeRelative attribute), 99
Relation (class in psamm.lpsolver.lp), 90
relative (psamm.util.MaybeRelative attribute), 99
remove_reaction() (psamm.metabolicmodel.MetabolicModel method), 95
repeat() (psamm.formula.FormulaElement method), 82
RequirementsError, 85
resolve_format() (in module psamm.datasource.native), 72
Result (class in psamm.lpsolver.cplex), 85
Result (class in psamm.lpsolver.glpk), 87
Result (class in psamm.lpsolver.gurobi), 88
Result (class in psamm.lpsolver.lp), 91
Result (class in psamm.lpsolver.qsoptex), 93
result (psamm.lpsolver.lp.Problem attribute), 90
reverse (psamm.reaction.Direction attribute), 97
reversible (psamm.database.MetabolicDatabase attribute), 64
reversible (psamm.datasource.sbml.ReactionEntry attribute), 73
right (psamm.reaction.Reaction attribute), 98
root (psamm.expression.boolean.Expression attribute), 76
run() (psamm.command.Command method), 62

## S

SBMLReader (class in psamm.datasource.sbml), 73
SBMLWriter (class in psamm.datasource.sbml), 74
sense (psamm.lpsolver.lp.Relation attribute), 91
set() (psamm.lpsolver.lp.Problem method), 90
set() (psamm.lpsolver.lp.VariableNamespace method), 92
set_linear_objective() (psamm.lpsolver.cplex.Problem method), 85
set_linear_objective() (psamm.lpsolver.glpk.Problem method), 86
set_linear_objective() (psamm.lpsolver.gurobi.Problem method), 88
set_linear_objective() (psamm.lpsolver.lp.Problem method), 90

set_linear_objective() (psamm.lpsolver.qsoptex.Problem method), 93
set_objective() (psamm.lpsolver.cplex.Problem method), 85
set_objective() (psamm.lpsolver.glpk.Problem method), 86
set_objective() (psamm.lpsolver.gurobi.Problem method), 88
set_objective() (psamm.lpsolver.lp.Problem method), 90
set_objective() (psamm.lpsolver.qsoptex.Problem method), 93
set_objective_sense() (psamm.lpsolver.cplex.Problem method), 85
set_objective_sense() (psamm.lpsolver.glpk.Problem method), 87
set_objective_sense() (psamm.lpsolver.gurobi.Problem method), 88
set_objective_sense() (psamm.lpsolver.lp.Problem method), 90
set_objective_sense() (psamm.lpsolver.qsoptex.Problem method), 93
set_reaction() (psamm.database.DictDatabase method), 63
simplify() (psamm.expression.affine.Expression method), 75
simplify() (psamm.expression.affine.Variable method), 75
solve() (psamm.fluxcoupling.FluxCouplingProblem method), 80
solve() (psamm.lpsolver.cplex.Problem method), 85
solve() (psamm.lpsolver.glpk.Problem method), 87
solve() (psamm.lpsolver.gurobi.Problem method), 88
solve() (psamm.lpsolver.lp.Problem method), 90
solve() (psamm.lpsolver.qsoptex.Problem method), 93
Solver (class in psamm.lpsolver.cplex), 85
Solver (class in psamm.lpsolver.generic), 85
Solver (class in psamm.lpsolver.glpk), 87
Solver (class in psamm.lpsolver.gurobi), 88
Solver (class in psamm.lpsolver.lp), 91
Solver (class in psamm.lpsolver.qsoptex), 93
SolverCommandMixin (class in psamm.command), 62
species (psamm.datasource.sbml.SBMLReader attribute), 74
SpeciesEntry (class in psamm.datasource.sbml), 74
status (psamm.lpsolver.cplex.Result attribute), 85
status (psamm.lpsolver.glpk.Result attribute), 87
status (psamm.lpsolver.gurobi.Result attribute), 88
status (psamm.lpsolver.lp.Result attribute), 91
status (psamm.lpsolver.qsoptex.Result attribute), 93
StoichiometricMatrixView (class in psamm.database), 64
substitute() (psamm.expression.affine.Expression method), 75
substitute() (psamm.expression.affine.Variable method), 75
substitute() (psamm.expression.boolean.Expression method), 76
substitute() (psamm.formula.FormulaElement method), 82
SubstitutionError, 76
success (psamm.lpsolver.cplex.Result attribute), 85
success (psamm.lpsolver.glpk.Result attribute), 87
success (psamm.lpsolver.gurobi.Result attribute), 88
success (psamm.lpsolver.lp.Result attribute), 91
success (psamm.lpsolver.qsoptex.Result attribute), 93
sum() (psamm.lpsolver.lp.VariableNamespace method), 92
symbol (psamm.expression.affine.Variable attribute), 75
symbol (psamm.formula.Atom attribute), 81
symbol (psamm.formula.Radical attribute), 82
symbol (psamm.reaction.Direction attribute), 97

## T

translate() (psamm.reaction.Compound method), 97
translated_compounds() (psamm.reaction.Reaction method), 98

## U

unbounded (psamm.lpsolver.cplex.Result attribute), 85
unbounded (psamm.lpsolver.glpk.Result attribute), 87
unbounded (psamm.lpsolver.gurobi.Result attribute), 88
unbounded (psamm.lpsolver.lp.Result attribute), 91
unbounded (psamm.lpsolver.qsoptex.Result attribute), 93
upper (psamm.metabolicmodel.FlipableFluxBounds attribute), 94
upper (psamm.metabolicmodel.FluxBounds attribute), 95

## V

value (psamm.datasource.sbml.FluxBoundEntry attribute), 73
value (psamm.expression.boolean.Expression attribute), 76
value() (psamm.lpsolver.lp.VariableNamespace method), 92
value_sets() (psamm.lpsolver.lp.Expression method), 89
values() (psamm.lpsolver.lp.Expression method), 89
var() (psamm.lpsolver.lp.Problem method), 90
Variable (class in psamm.expression.affine), 75
Variable (class in psamm.expression.boolean), 76
VariableNamespace (class in psamm.lpsolver.lp), 91
variables (psamm.expression.boolean.Expression attribute), 76
variables() (psamm.expression.affine.Expression method), 75
variables() (psamm.formula.FormulaElement method), 82
variables() (psamm.lpsolver.lp.Expression method), 89
VariableSet (class in psamm.lpsolver.lp), 92
VariableType (class in psamm.lpsolver.lp), 92

version_string (psamm.datasource.native.NativeModel attribute), 69

## W

write() (psamm.util.LoggerFile method), 99
write_compartments() (psamm.datasource.native.ModelWriter method), 68
write_compounds() (psamm.datasource.native.ModelWriter method), 68
write_model() (psamm.datasource.sbml.SBMLWriter method), 74
write_reactions() (psamm.datasource.native.ModelWriter method), 68

## Y

yaml_load() (in module psamm.datasource.native), 72