>>>py[cellerator]

Cell Simulation

Symbolic Processing in Python

Reference Notes

bruce.e.shapiro@csun.edu
Department of Mathematics
California State University, Northridge

Revised: Saturday 5th September, 2015 at 21:07

© 2015 Bruce E Shapiro

This work is licensed under the Creative Commons Attribution-NonCommercial-NoDerivs 3.0 United States License. To view a copy of this license, visit http://creativecommons.org/licenses/by-nc-nd/3.0/us/ or send a letter to Creative Commons, 444 Castro Street, Suite 900, Mountain View, California, 94041, USA.
The following is a summary of the license.

You are free:
- to Share – to copy, distribute and transmit the work

Under the following conditions:
- Attribution – You must attribute the work in the manner specified by the author or licensor (but not in any way that suggests that they endorse you or your use of the work).
- Noncommercial – You may not use this work for commercial purposes.
- No Derivative Works – You may not alter, transform, or build upon this work.

With the understanding that:
- Waiver – Any of the above conditions can be waived if you get permission from the copyright holder.
- Public Domain – Where the work or any of its elements is in the public domain under applicable law, that status is in no way affected by the license.
- Other Rights – In no way are any of the following rights affected by the license:
  - Your fair dealing or fair use rights, or other applicable copyright exceptions and limitations;
  - The author's moral rights;
  - Rights other persons may have either in the work itself or in how the work is used, such as publicity or privacy rights.

Notice – For any reuse or distribution, you must make clear to others the license terms of this work. The best way to do this is with a link to the web page http://creativecommons.org/licenses/by-nc-nd/3.0/us/.

Figure 1.2 is licensed subject to CC-BY-SA 3.0 and GPL 2.0.

pycellerator is 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.

A copy of the license is provide in Appendix B. For more details visit http://www.gnu.org/licenses/.

Contents

Preface v

1 Introduction 1

2 Installation 5

2.1 Install Python 5

2.2 Install libSBML 8

2.3 Install pycellerator 8

3 Arrows 11

3.1 Introduction 11

3.2 Equations for Rates 13

3.3 Indexed Species 13

3.4 `Nil` & `EmpytSet` Species 14

3.5 Mass Action Arrows 14

3.6 Equilibrium / Steady-State Models . . 18

3.7 Regulatory Arrows 20

3.8 User Defined Stoichiometric Arrows . 26

3.9 Cascades 27

3.10 Flux Arrows 27

4 Functions 29

4.1 Command Line Interface 29

4.2 parser 29

4.3 expander 32

4.4 interpreter 34

4.5 solver 38

4.6 converter 40

4.7 SBML 41

5 Examples 49

5.1 MAP-Kinase with Oscillations 50

5.2 Repressilator 52

5.3 Lineage Determination 55

5.4 NF-$\kappa\beta$ Signaling Model 57

5.5 Minimal Mitotic Model 60

5.6 Ring Oscillator (Mass Action) 61

5.7 Ring Oscillator (MMH) 62

5.8 Ring Oscillator Using GRN Reactions 63

5.9 Ring Oscillator (NHCA) 64

5.10 Recombinant *E coli* growth model . . 65

5.11 MAPK Cascade with Indexed Stages . 67

A File Formats 69

A.1 Reaction Files 69

A.2 Model Files 69

B Software License 73

Preface

The goal of `pycellerator` is to provide a software tool for biological simulation that

- Has all of the capabilities of the `Cellerator` arrow based reaction language[16];
- Is open source;
- Runs under Linux, Windows, and Mac OS X;
- Is written in a multi-platform language;
- May be used either as a stand-alone program, at the command-line, or as a library within a high-level computer language;
- Is written in a computer language that is freely available (at no cost to users) on all platforms;
- Does not depend on any external libraries except for freely available multi-platform libraries, so that it would not be necessary to maintain separated executable (or any executable) builds.

Python 2.7 was chosen as the programming platform because it is implemented on all three platforms, is freely available, has a large user support community, and has an extensive collect of external libraries that support it.

The last requirement is more complicated, because it rules out a lot of useful libraries that are written in C or Fortran and wrapped in Python, and hence required complicated installations (read: recompilations) for each Operating System. It is our contention that this software should not be operating system dependent and thus the final requirement was added.

`pycellerator` was developed using standard distributions provided by the Python community (`http://www.python.org`). It is fully compatible with standard distributions available commercially such as Anaconda and Enthought Canopy. It can also be used in ipython notebooks.

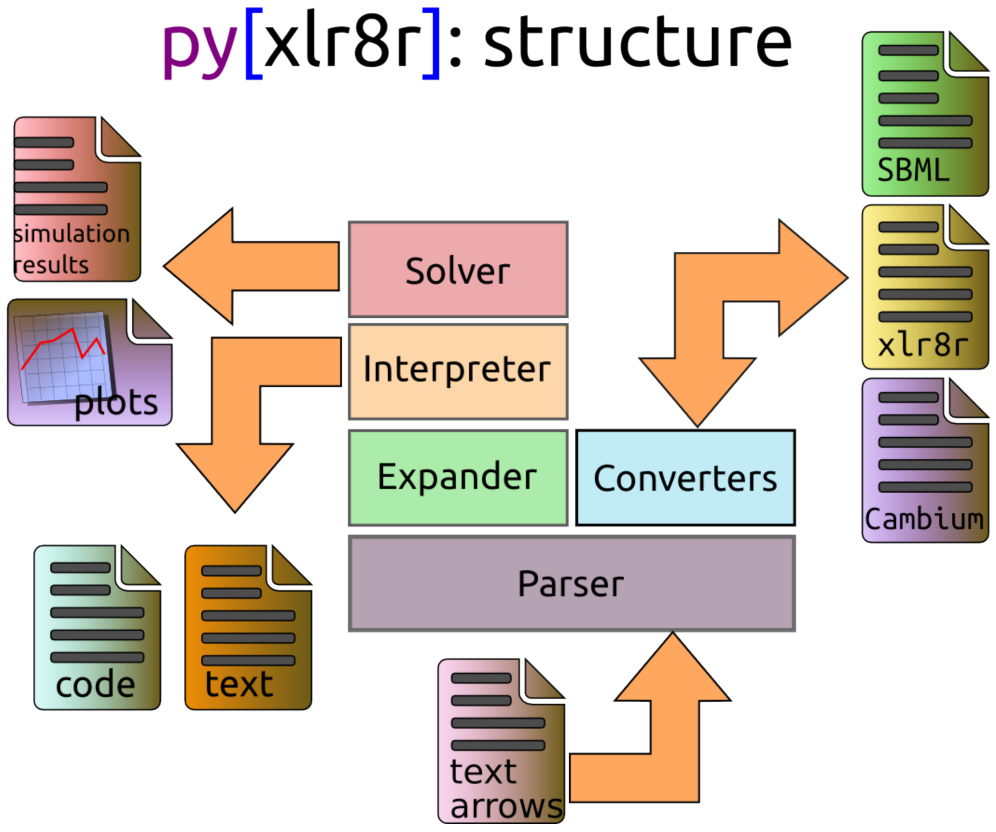
Chapter 1

Introduction

This chapter will provide an overview of pycellerator structure and functionality.

`pycellerator` has the following components, as illustrated in figure 1.1. The functional relationship between these components is illustrated in figure 1.2. In brief, `pycellerator` first reads a model file with the **parser** and converts it to an internal database. This database is then processed by the **expander** module converts the reactions in the database to their most basic form. Control is then passed to the **interpreter**, which converts the reactions to a second database of terms of terms in differential equations, and then combines all of these terms together to form a system of differential equations out of the model. Finally, the **solver** writes the system of differential equations to a python program and then runs the program, if required to perform a numerical simulation.

Figure 1.1: Schematic of `pycellerator` structure.



A Parser: The parser module reads lists of text-formatted reactions, typically from a text file that you will manually edit by hand, that describe a particular biochemical phenomenon. For example, a simplified version of the famous Belousov-Zhabotinsy reaction[1, 22] known as the Oregonator[5, 6] can be written in this language as:

```
[Br + BrO3 -> HBrO2 + HOBr, k1]
[Br + HBrO2 -> 2 HOBr, k2]
[BrO3 + HBrO2 -> 2 Ce + 2 HBrO2, k3]
[2 HBrO2-> BrO3 + HOBr, k4]
[Ce -> 0.5 Br, k5]
```

The parser module converts the reactions into an internal data structure that can be interrogated by the other modules with such questions as "What are the products of this reaction?"

The parser module is not normally invoked directly by the user.

Expander: The expander module converts a list of complex reactions into its most basic form. For example, the input reaction

```
[X => Y, mod[E], rates[k1,k2,k3]]
```

is a shorthand that represents the set of biochemical reactions

$$\mathtt{X{+}E} \overset{k_1}{\rightarrow} \mathtt{X\_E}$$

$$\mathtt{X\_E} \overset{k_2}{\rightarrow} \mathtt{X{+}E}$$

$$\mathtt{X\_E} \overset{k_3}{\rightarrow} \mathtt{X{+}Y}$$

where X_E is the name of the complex formed when E is bound to X. The expander converts the input reaction to its three component reactions:

```
[X+E->X_E,k1]
[X_E->X+E,k2]
[X_E->Y+E,k3]
```

While it is possible for the user to call the expander directly to see what the component reactions look like, usually this will be done automatically by the interpreter or solver modules and the user will generally not be required to directly interact with the expander.

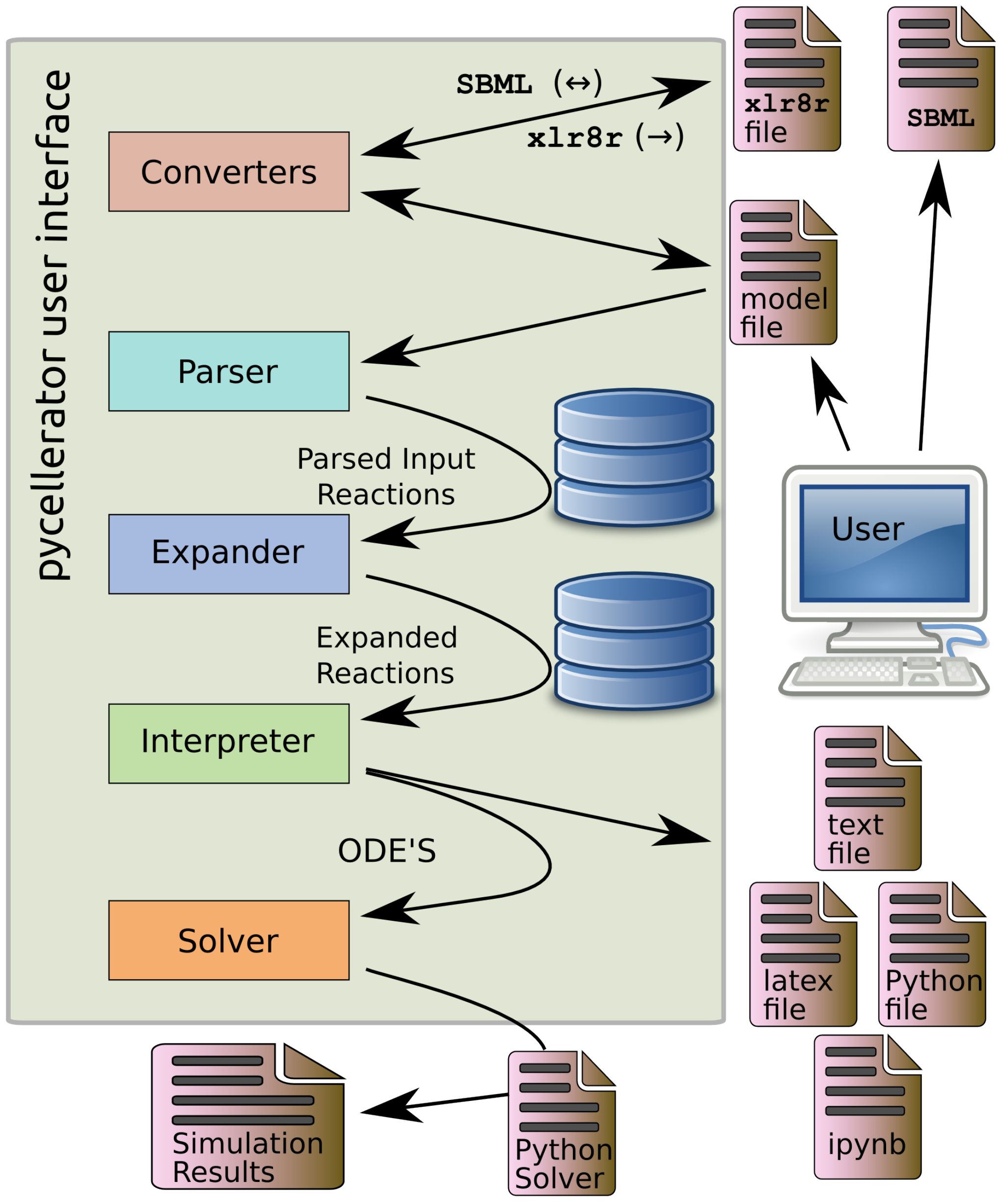
Interpreter: The interpreter can provide a list of differential equations, a simple python program that instantiates the system of differential equations, or a LaTeX representation of the system of differential equations.

The interpreter can be invoked directly, if that is what is required, but if the user is only interested in a simulation, the interpreter need not be invoked manually.

Solver: The primary function of the solver module is to produce a python function that is compatibly with `scipy.tt.odeint`. If requested the solver module will also run a simulation utilizing the python function, plot selected (or all) simulation variables, and/or write the results to an output file in `CSV` or `TSV` format. Because this function is a completely stand-alone program – fully independent of all other `pycellerator` components – multiple instantiations can be easily parallelized, e.g., for parameter optimization or other chores.

Converters: The converters module is used to convert to and from other formats, particularly `Cellerator` (`xlr8r`) arrows. Every arrow implemented in `pycellerator` is equivalent to some arrow in `xlr8r` and vice-versa.

Figure 1.2: Functional overview of `pycellerator` operation. Users will interface with `pycellerator` in any of the following ways: via the `pycellerator` command line interface, e.g., in the mac `terminal` or or windows command prompt; in an ipython notebook; in the python shell; or by direct functions calls from other programs. Every function is available via any of these techniques. [The database icon is taken from the Wikimedia Commons and has a CC-BY-SA 3.0 license. The Computer icon was taken from Wikimedia commons and has a GPL 2.0 license. These licenses supercede the copyright license of this document. This picture may be copied and reused under the terms of the CC-BY-SA 3.0 and GPL 2.0 licenses.]



Chapter 2

Installation

2.1 Install a Basic Python System

To use `pycellerator`, python 2.7 and several python libraries must be installed on your computer. Python is free, open source, and available on all major operating systems. This section provides an overview of how to install a basic python system on Windows, Macintosh OX, and Linux operating systems.

2.1.1 Installing on Windows

There are two basic ways to install python: (1) install a “commercial” base distribution or (2) use a binary installer from `https://www.python.org/downloads/windows/`https://www.python.org/downloads/windows/. There are several commercial distributions that provide free installers that will install the base python system for you. If you use a binary distribution, you will also have to install a number of other packages. The easiest commercial distributions to use are **Anaconda** and **Enthought Canopy**.

Anaconda Python

To install Anaconda Python, go to http://continuum.io/downloads and scroll down to select the appropriate installer for your operating system. This will download a file (e.g., Anaconda-2.3.0-x86_64.exe, or something similar).

Double click on the installer and follow the instructions to install python.

After you are done, locate the `Anaconda Command Prompt` from the windows search menu and type in the following. Type enter after line and wait until the command prompt (the name of the current folder) is shown.

```
conda update conda
conda update ipython ipython-notebook ipython-qtconsole
conda update numpy scipy sympy matplotlib
python -m pip install --upgrade pip
pip install pyparsing
pip install pulp
```

Enthought Canopy

As an alternative to Anaconda, Enthought Canopy is at https://store.enthought.com/downloads/. Pick your operating system from the button on the top of page and download the binary installer.

After downloading the installation package (e.g., `canopy-1.5.5-win-64.msi` or a similarly named file), double click on the installer to install python. It will be sufficient to answer all of the questions in the dialogs with the default values.

When the installer is finished, the canopy dashboard should open automatically. You must run the dashboard at least once to finish the installation. The dashboard will open a menu that says "Welcome to Canopy" at the top. If the dashboard does not open, there should be a Canopy icon on your desktop. Click this to open the Canopy dashboard. If this icon is not installed, open Canopy from the windows search menu. If it does not appear, the installation has not completed properly.

From the Canopy dashboard select `Edit > Preferences > General` and verify that Canopy is set as your default python environment. If it is not, click on the button that says "Set as Default", then click on "OK."

Then go to `tools > package manager > available packages` and click on "install all available packages.'

Then exit from Canopy by selecting `File > Exit`.

From the windows search menu, open `Command Prompt` and type in the following commands, one at a time. Type enter after line and wait until the command prompt (the name of the current folder) is shown.

```
python -m pip install --upgrade pip
pip install --upgrade numpy scipy sympy matplotlib
pip install pyparsing
pip install pulp
```

Using the Binary Installers

From https://www.python.org/downloads/windows, download the latest MSI installer for Python 2.7. At the time this was written, the file name was `python-2.7.9.amd64.msi`.

Locate the installer file, double click, and answer all the prompts. Make sure during the installation that python.exe and pip and checked off as visible to all.

Next, download a Microsoft Visual C compiler for Python from http://www.microsoft.com/en-us /download/details.aspx?id=44266. The installer file is called `VCforPython27.msi`. Locate the installer, double click, and follow the instructions. Restart your computer after the installation is complete.

Open a command prompt (Windows search, type `command prompt`) and enter the following:

```
python -m pip install --upgrade pip
pip install sympy pulp pyparsing setuptools
pip install numpy scipy matplotlib
pip install ipython[notebook]
```

2.1.2 Installing on Macintosh OS

If you have a Mac, then Python is already installed on your computer. The base system is pre-installed as part of the Mac operating system. The base system does not include the numerical libraries that are also required. You can either install these using `pip` or install one of the commercial systems like Anaconda or Enthought Canopy.

Follow the instructions for Windows if you want to install the commercial system. You will not need to install Visual C; instead, you may be prompted to download and install XCode tools from Apple during the installation process.

Upgrade Mac Python using `pip`

Locate the `terminal` application in the `utilities` folder and open it. Enter

```
sudo easy_install pip
```

When requested, enter you password (you must have administrator access on you Mac). If (when) you are prompted to install XCode from Apple, click yes, and follow the instructions on any dialog that follows.

When the XCode installation is completed (or if it was not suggested), open a new terminal session and type in the following. Hit the enter key after each line and wait for the prompt (the name of the current working directory) before typing in the next line.

```
pip install sympy pulp pyparsing
pip install numpy scipy matplotlib
pip install ipython[notebook]
```

If you have a virtual operating system like Parallels Desktop installed on your computer, make sure that Safari (or some other web browser such as Firefox or Chrome) is set as your default browser, and not Parallels. Otherwise ipython will try to open your virtual operating system every time it runs python.

2.1.3 Installing on Linux

You can either install a base python system from your package manager, download binaries from python.org, or build from source (also available at python.org. Pycellerator requires python 2.7.X but is not compatible with python 3.x.

The standard python installation includes its own package manage called `pip`. If you install the base system from python.org this should automatically be installed for your. Otherwise, you should look to also install `pip` from your package manager. This allows you to bypass your package manager when updating python.

To upgrade to the latest version of `pip`,

```
python -m pip install --upgrade pip
```

To add any missing packages to python,

```
pip install packagename
```

To upgrade to the latest version of any package,

```
pip install --upgrade package
```

Pycellerator needs the following packages which you can get from pip: `pyparsing`, `pulp`, `sympy`, `numpy`, `scipy`, and`matplotlib`.

```
pip install pyparsing
pip install pulp
pip install sympy
pip install numpy
pip install scipy
pip install matplotlib
```

You can also put all the function names on a single line:

```
pip install pyparsing pulp sympy numpy scipy matplotlib
```

Binary installers and source code versions are also available for each of these packages.

Package	Project Home Page
numpy	http://numpy.org/
scipy	http://scipy.org/
matplotlib	http://matplotlib.sourceforge.net/
pyparsing	http://pyparsing.wikispaces.com/
sympy	http://code.google.com/p/sympy/
pulp	https://github.com/coin-or/pulp

2.2 Install libSBML

If you want to work with SBML files, you will also need to install a version of libSBML that is appropriate for your operating system. Make sure to install a version that is compatible with Python.

Follow the instructions at http://sbml.org/Software/libSBML to find the appropriate binary installer for your operating system.

You do not have to have libSBML installed if you do not plan on using SBML files. If libSBML is not installed, all non-SBML related functionality of pycellerator will be unaffected.

The following describes how to install libSBML in ubuntu if you are primarily interested in using libSBML for python and not for other languages. It was taken from `http://sourceforge.net/projects/sbml/files/libsbml/5.11.6-experimental/binaries/Linux/` on 5 Sept 2015, and you should check for the latest information.

1. Install the prerequisite software packages installed on your operating system: **`python-dev`**, **`libxml2-dev`**, **`libz-dev`**, and **`libbz2-dev`**.

   ```
   sudo apt-get install python-dev libxml2-dev libz-dev libbz2-dev
   ```

2. From the terminal type:

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

For windows, the latest (as of 5 Sept 2015) binary python installers are at `http://sourceforge.net/projects/sbml/files/libsbml/5.11.4/stable/Windows/64-bit/python/`. These installers will only install libSBML for python, and not for other languages.

The Mac installers are at `http://sourceforge.net/projects/sbml/files/libsbml/5.11.4/stable/Mac%20OS%20X/` and the instructions given for installation are similar to those for linux.

2.3 Install pycellerator

The procedure to install pycellerator follows.

1. Get the latest **release** from github. This can be found at `https://github.com/biomathman/pycellerator/releases`. Download the file **`Install-pycellerator-v-X.zip`**, where **`X`** is the

latest version number. This will be a compressed archive (zip file) of the files you need to install pycellerator.

2. Unzip the archive. Copy the folder **pycellerator** (e.g., drag and drop the entire folder) to wherever you want to keep your pycellerator files. For example, this may be a folder in your home directory. If your home folder is **johnsmith**, and you drag **pycellerator** onto your home folder, then your **pycellerator** folder will be **johnsmith/pycellerator**, i.e., the folder **pycellerator** inside your home folder.

3. Inside the **pycellerator** folder is detailed documentation in the file **pycellerator.pdf** (the file you are reading now). Chapter 2 gives detailed instructions on how to install Python (if you don't already have it installed); and what additional Python libraries are needed and how to get them.

4. A quickstart check of the command line utility, from inside the **pycellerator** folder, type the following into the terminal:

```
python pycellerator.py solve -in Gold1.model -plot
```

5. For a quickstart check of the ipython notebook, type the following into the terminal:

```
ipython notebook
```

   Navigate to the notebook **demo.ipynb** and open it.

Chapter 3

pycellerator Arrow Reference

3.1 Introduction

Reactions in pycellerator are specified in textfiles using a arrow based language that can be typed using any standard ASCII or UTF keyboard. All of the characters needed to type a reaction exist on standard US keyboards. A summary of the arrow forms available in pycellerator is given in Table 3.1.

The canonical form for an arrow form is

`[LHS` *arrow* `RHS, mod[`*modifiers*`], rates[`*rate constants*`]]`

or

`[LHS` *arrow* `RHS, rates[`*rate constants*`]]`

All of the square brackets are required and:

`LHS` indicates a species, list of species, or sum of species that give the input to the reaction. Depending on the type of reaction, their concentrations or amounts may or may not change as a result of the reaction, but they will always affect the calculation of the output species.

`RHS` indicates a species, list of species, or sum of species that give the output of the reaction. Each species in `RHS` will will normally change in concentration or amount as a result of the reaction.

arrow determines the type of reaction. It typically looks something like `->, -->, =>, |->`, etc.

modifiers indicates a list of species that affect the output of the reaction; like input reaction, their concentrations may or may not be affected by the reactions. The definition of whether a species goes in the `mod` or *modifiers* list depends on the definition of the reaction in the following sections, and not on the biochemical process. This is a computational distinction only. In general, `modifiers` correspond to catalysts in enzymatic reactions. The normal distinction used is that for **catalytic** arrows (e.g., `|->`) the amount of catalyst does not change, but in **enzymatic** arrows (e.g., `=>, <=>`) the amount of enzyme may, in fact, change.

Table 3.1

Name	**Arrow**	**Typical ODE Term**	**Ref**
Simple Mass Action	1 `[X − > Y, k]`	$(s_{i,r}-s_{i,l})k\prod_{i\in\mathrm{LHS}}\mathrm{X}_i^{e_i}$	3.5.1
Mass Action, Stoichiometry	2 `[e1 X1 + e2 X2 + ⋯ − > s1 Y1 + s2 Y2 + ⋯, k]`		3.5.2
Mass Action, Reversible	3 `[e1 X1 + ⋯ < − > s1 Y1 + ⋯, rates[k1,k2]]`	Expanded into a pair of type (1) or (2) reactions.	3.5.4
Simple Catalytic	4 `[e1 A1 + e2 A2 + ⋯ --> s1 B1 + s2 B2 +⋯, mod[X], k]`	Expanded into a single type (1) or (2) reaction.	3.5.3
Catalytic, w/ Intermediate Complex	5 `[A => B, mod[X], rates[k1, k2, k3, k4]]`	Expanded into three or four type (1) reactions	3.5.5
Catalytic, Reversible, w/ Int. Complex	6 `[A <=> B, mod[X,Y], rates[k1, k2,...,k8]]`	Expanded into two type (5) reactions.	3.5.6
Catalytic, Two Int. Species	7 `[A :=> B, mod[E], rates[k1,k2,...,k6]]`	Expanded into six type (1) reactions	3.5.7
Hill Functions	8 `[A |-> B, Hill[v,n,K,α,T]]`	$\dfrac{vEk(\sum T_iA_i+\alpha)^n}{K^n+(\sum T_iA_i+\alpha)^n}$	3.7.1
	9 `[A |--> B, mod[E], Hill[v,n,K,α,T]]`		
GRN (Genetic Regulatory Network Model)	10 `[A |-> B, GRN[v,β,n,h]]`	$\dfrac{vE}{1+\exp(-h-\sum\beta_iA_i^{n_i})}$	3.7.2
	11 `[A |--> B, mod[E], GRN[v,β,n,h]]`		
S-System	12 `[A |-> B, SSystem[τ, k+, k−, c+,c−]]`	$\dfrac{k_+\prod A_i^{C_{i,+}}-k_-\prod A_i^{C_{i,-}}}{\tau}$	3.7.3
	13 `[A |--> B, mod[E], SSystem[τ, k+, k−, c+,c−]]`		
MMH (Michaelis-Menten- Henri)	14 `[A :->B, MMH[K,v]]` or `[A :->B, MMH[k1,k2,k3]]`	$\dfrac{vAE}{K+A}$ or $\dfrac{k_3AE}{(k_2+k_3)/k_1+A}$	3.6.1
	15 `[A :-->B, mod[X], MMH[K,v]]` or `[A :->B,mod[X], MMH[k1,k2,k3]]`		
Rational Function	16 `[[[A1,A2,..],[X1,X2,..]]==>S,rational[a,d,m,n]]`	$\dfrac{a_0+\sum a_iA_i^{n_i}}{d_0+\sum d_iX_i^{m_i}}$	3.7.5
MWC (Monod-Wyman-Changeaux)	17 `[S==>P, mod[E], MWC[k, n, c, L, K]]`	$\dfrac{k\mathcal{E}\left(cLs(cs+1)^{n-1}+s(s+1)^{n-1}\right)}{L(cs+1)^n+(s+1)^n}$	
	18 `[S==>P, mod[E, [[A1,A2,..],[I1,I2,..]], MWC[k, n, c, L, K]]`	See reference for multiple Activator/Inhibitor equations.	3.6.2
NHCA (Non-hierarchical coop. act.	19 `A|-> B, NCHA[v, Tp, Tm, n, m, k]`	$\dfrac{vA^m}{kB^m+A^m}$, $A{=}1{+}T_pX^n$, $B{=}1{+}T_mX^n$	3.7.4

Table 3.1 (continued)

Name	**Arrow**	**Typical ODE Term**	**Ref**
USER	20 `[ X1+X2+···|->Y, USER[v, T, n, h, f]]` `f` should by a Python lambda expression of a single variable	$vf(h-\sum T_i P_i^{X_i})$	3.7.6
using	21 `[s1*X1+s2*X2+··· − > q1 * X1+···, using["expr"]]` `expr` should be Python infix expression depending on variables defined in the model.	$X_i' = q_i - s_i)X_i$	3.8

`rates` is a keyword that varies with the type of arrow; typical keywords include `MWC` (for Monod-Wyman-Changeaux); `MMH` (for Michaelis-Menten-Henri); `Hill` (for Hill functions); and so forth.

rate constants is a list of symbols or numbers that give the rate constants or other parameters used in the reaction equation. The definition of these parameters is different for each type of reaction, and is described in the following sections and subsections.

3.2 Substituting Equations for Rate Constants

In any of the arrow expressions that follow in the remainder of the chapter, any rate constant may be replaced by any valid Python expression in quotation marks. Fore example, the hypothetical reaction

```
[X -> Y, "v/(K+X)"]
```

is perfectly valid. Since the reaction itself is written in the form of a mass action equation, the expression is treated as a rate constant k and the odes are `[Y]`$'$ == `[X]`$' = k$`[X]`, where k is to be replaced with the expression in quotes. It will thus produce the differential equation terms:

$$\texttt{[Y]}' = -\texttt{[X]}' = \frac{v\texttt{[X]}}{K + \texttt{[X]}}$$

This allows users to define virtually any rate law in a reaction.

See also User Defined Regulatory Arrows in section 3.7.6 and User Defined Stoichiometric Arrows in section 3.8. User defined stoichiometric arrows will not multiply the expression by the mass-action product (the product of the species on the left, each raised to their individual stoichiometries), as this method will, but will retain a balance of stoichiometry, with the each reactant and product changing unless their stoichiometries are balanced. The user defined regulatory reactions, on the other hand, will not generate any ODE terms for the reactants on the left-hand side of the arrow.

3.3 Indexed Species

An array index may be attached to a species variable using parenthesis as a delimiter, as in the following example:

```
[K(3,0) <=> K(3,1), mod[RAFK, RAFph], rates[a1,d1,k1,0,a2,d2,k2,0]]
```

The index may also be applied to a modifier variable, and within the initial condition section of the model file, e.g.,

```
      K[3,0] = 0.7
```

See the examples chapter (MAPK cascade with indexed stages) for an example.

Implementation Note: When instantiated in the `Reaction` class, the indices are expanded into the variable name, and separated with underscores e.g., `K[i,j]` becomes `K_i_j`; this variables are implemented in this expanded form in the solver and displayed this way on plots.

3.4 The `Nil` and `EmptySet` Species

The species `Nil` is used to represent the vacuum or the empty set. No ODE term is ever generated for the species `Nil`.

The variable `EmptySet` is predefined in Sympy and should not be used in models. If it is used in an SBML file it will be converted to `Nil`.

For example, the reaction

```
[Nil->A, k]
```

denotes creation and represents a special case of mass-action reaction, in which $A' = k$ (rather than $A' = k \times$ [`Nil`] as it would be in any other reaction).

Similarly, the reaction

```
[A->Nil, k]
```

represents destruction, and is treated normally for the species `A`, but no differential equation term is generated for `Nil`.

3.5 Mass Action Arrows

3.5.1 Simple Mass Action

pycellerator Arrow:

```
     [X -> Y, k]}}
```

`X, Y` are identifiers representing chemical species; `k` is either an identifier or a number representing a rate constant.

Equivalent `xlr8r` Arrow:

$$\{\mathtt{X} \to \mathtt{Y},\ \mathtt{k}\}$$

Typical biochemical notation:

$$\mathrm{X} \xrightarrow{k} \mathrm{Y}$$

Interpreted differential equation terms:

$$\frac{d\mathrm{Y}}{dt} = -\frac{d\mathrm{X}}{dt} = k\mathrm{X}$$

3.5.2 Simple Mass Action, with stoichiometry

pycellerator Arrows:

```
[e1 X1 + e2 X2 + ... -> s1 Y1 + s2 Y2 +... , k]
[e1*X1 + e2*X2 + ... -> s1*Y1 + s2*Y2 +... , k]
```

Species: `X1, X2, ..., Y1, Y2, ...` are identifiers representing chemical species.

Stoichiometry: `e1, e2, ..., s1, s2, ...` are either identifiers or numbers representing stoichiometries before and after the reaction. The multiplication symbols ($*$) are optional.

If any stoichiometry is an identifier, there must be either an $*$ or a blank space between it and the corresponding species identifier. If the stoichiometry is numerical the blank space is optional. For example `3X`, `3*X`, and `3 X`, will all be treated as a species `X` with stoichiometry 3, and `s H2O` and `s*H2O` will both be interpreted as a species `H2O` with stoichiometry `s`, whereas `sH2O` will be interpreted as a single species `sH2O` with a stoichiometry of 1.

Equivalent xlr8r Arrow:

$$\{\texttt{e1 X1 + e2 X2 + } \cdots \rightarrow \texttt{ s1 Y1 + s2 Y2 + } \cdots\texttt{, k}\}$$

Typical biochemical notation:

$$e_1\mathrm{X}_1 + e_2\mathrm{X}_2 + \cdots \xrightarrow{k} s_1\mathrm{Y}_1 + s_2\mathrm{Y}_2 + \cdots$$

Interpreted differential equation terms:

$$\frac{d\mathrm{U}_i}{dt} = (s_{i,r} - s_{i,l})k\mathrm{X}_1^{e_1}\mathrm{X}_2^{e_2}\cdots$$

where $s_{i,r}$ and $s_{i,l}$ are the stoichiometry of U_i on the right-hand-side and left-hand-side of the reaction, respectively, and U_i refers to any species either on either side of the equation (either an X_i or a Y_i). The product is taken over all the terms on the left-hand-side of the reaction, each term raised to its respective stoichiometry (this is also known as the law of mass action).

3.5.3 Simple Mass Action, Catalyzed

pycellerator Arrows:

```
[e1 A1 + e2 A2 + ... --> s1 B1 + s2 B2 + ..., mod[X], k]
[e1*A1 + e2*A2 + ... --> s1*B1 + s2*B2 + ..., mod[X], k]
```

The stoichiometries are optional. Multiplication symbols ($*$) or spaces are optional if numerical stoichiometries are used, but are required if symbolic stoichiometries are used.

Equivalent xlr8r Arrow:

$$\{\mathtt{e}_1\mathtt{A}_1 + \mathtt{e}_2\mathtt{A}_2 + \cdots \xrightarrow{\mathtt{X}} \mathtt{s}_1\mathtt{B}_1 + \mathtt{s}_2\mathtt{B}_2 + \cdots, \mathtt{k}\}$$

Typical biochemical notation:

$$\mathtt{X} + \mathtt{e}_1\mathtt{A}_1 + \mathtt{e}_2\mathtt{A}_2 + \cdots \xrightarrow{\mathtt{k}} \mathtt{S} + \mathtt{s}_1\mathtt{B}_1 + \mathtt{s}_2\mathtt{B}_2 + \cdots$$

Interpreted differential equation terms: The reaction is expanded into a single reaction of the form

```
[X + e1 A1 + e2 A2 + ... -> X + s1 B1 + s2 B2 + ..., mod[X], k]
```

as described in section 3.5.2.

3.5.4 Reversible Mass Action

pycellerator Arrow:

```
[e1 X1 + e2 X2 + ... <-> s1 Y1 + s2 Y2 + ... , rates[k1,k2]]
```

The stoichiometries are optional.

Equivalent xlr8r Arrow:

$$\{\texttt{e1 X1 + e2 X2 + } \cdots \rightleftarrows \texttt{ s1 Y1 + s2 Y2 + } \cdots\texttt{, k1, k2}\}$$

Typical biochemical notation:

$$e_1\mathrm{X}_1 + e_2\mathrm{X}_2 + \cdots \underset{k2}{\overset{k}{\rightleftarrows}} s_1\mathrm{Y}_1 + s_2\mathrm{Y}_2 + \cdots$$

Interpreted differential equation terms:

This arrow is reduced to the pair of simple mass action equations, as described in section 3.5.2:

```
[e1 X1 + e2 X2 + ... -> s1 Y1 + s2 Y2 + ... , k1]
[s1 Y1 + s2 Y2 + ... -> e1 X1 + e2 X2 + ... , k2]
```

The contributions to the differential equations are then computed as described in section 3.5.2.

3.5.5 Catalytic Mass Action, Intermediate Substrate/Catalyst Complex Formation

pycellerator Arrow:

```
[A => B, mod[X], rates[k1, k2, k3, k4]]
```

Equivalent xlr8r Arrow:

$$\{\mathtt{A} \overset{\mathtt{X}}{\rightleftarrows} \mathtt{B}, \mathtt{k1}, \mathtt{k2}, \mathtt{k3}, \mathtt{k4}\}$$

Typical biochemical notation:

$$X + A \underset{k_2}{\overset{k_1}{\rightleftarrows}} XA \underset{k_4}{\overset{k_3}{\rightleftarrows}} X + B$$

Typically k4 is omitted, in which case it is assumed to be zero. If fewer than four parameters are given they are zero-filled to the right.

This arrow is expanded into the following collection of simple mass action arrows as described in section 3.5.2:

```
[X + A -> A_X, k1]
[A_X -> X + A, k2]
[A_X -> X + B, k3]
[X + B -> B_X, k4]
```

The name of the intermediate complex is automatically generated by concatenating the names of the substrate and the catalyst with an underscore. The resulting contributions to the differential equations for [A], [B], [X] and [X_A] in the above scheme are then:

$$
\begin{aligned}
\texttt{[A]}' &= k_2\texttt{[A\_X]} - k_1\texttt{[A]}\texttt{[X]} \\
\texttt{[B]}' &= k_3\texttt{[A\_X]} - k_4\texttt{[B]}\texttt{[X]} \\
\texttt{[X]}' &= -k_1\texttt{[A]}\texttt{[X]} + (k_2 + k_3)\texttt{[A\_X]} - k_4\texttt{[B]}\texttt{[X]} \\
\texttt{[A\_X]}' &= k_1\texttt{[A]}\texttt{[X]} - (k_2 + k_3)\texttt{[A\_X]} + k_4\texttt{[B]}\texttt{[X]}
\end{aligned}
$$

3.5.6 Catalytic Mass Action, Intermediate Substrate/Catalyst Complex Formation, Reversible

pycellerator Arrow:

```
[A <=> B, mod[X, Y], rates[k1, k2, k3, k4, k5, k6, k7, k8]]
```

If fewer than 8 rate constants are given they are zero-filled to the right.

Equivalent xlr8r Arrow:

$$\{\texttt{A} \underset{\texttt{Y}}{\overset{\texttt{X}}{\rightleftarrows}} \texttt{B}, \texttt{k1}, \texttt{k2}, \texttt{k3}, \texttt{k4}, \texttt{k5}, \texttt{k6}, \texttt{k7}, \texttt{k8}\}$$

Typical biochemical notation:

$$X + A \underset{k_2}{\overset{k_1}{\rightleftarrows}} XA \underset{k_4}{\overset{k_3}{\rightleftarrows}} X + B$$

$$Y + B \underset{k_6}{\overset{k_5}{\rightleftarrows}} YB \underset{k_8}{\overset{k_7}{\rightleftarrows}} Y + A$$

When computing the differential equation contribution, this arrow is expanded into the following pair of catalytic mass action reactions with intermediate complex formation, as described in section 3.5.5:

```
[A => B, mod[X], rates[k1, k2, k3, k4]]
[B => A, mod[Y], rates[k5, k6, k7, k8]]
```

These reactions are subsequently broken down in simple mass action reactions as described in section 3.5.2.

```
[A+X->A_X,k1]}
[A_X->A+X,k2]}
[A_X->B+X,k3]}
[B+X->A_X,k4]}
[B+Y->B_Y,k5]}
[B_Y->B+Y,k6]}
[B_Y->A+Y,k7]}
[A+Y->B_Y,k8]}
```

The contributions to the ODE are computed as described based on these reduced reactions, as described in 3.5.2:

$$
\begin{aligned}
\texttt{[A]}' &= k_2\texttt{[A\_X]} + k_7\texttt{[B\_Y]} - A * X * k1 - A * Y * k8 \\
\texttt{[B]}' &= k_3\texttt{[A\_X]} + k_6\texttt{[B\_Y]} - B * X * k4 - B * Y * k5 \\
\texttt{[A\_X]}' &= -(k_2 + k_3)\texttt{[A\_X]} + k_1\texttt{[A]}\texttt{[X]} + k_4\texttt{[B]}\texttt{[X]} \\
\texttt{[B\_Y]}' &= -(k_6 + k_7)\texttt{[B\_Y]} + k_8\texttt{[A]}\texttt{[Y]} + k_5\texttt{[B]}\texttt{[Y]} \\
\texttt{[Y]}' &= (k_6 + k_7)\texttt{[B\_Y]} - k_8\texttt{[A]}\texttt{[Y]} - k_5\texttt{[B]}\texttt{[Y]} \\
\texttt{[X]}' &= (k_2 + k_3)\texttt{[A\_X]} - k_8\texttt{[A]}\texttt{[X]} - k_4\texttt{[B]}\texttt{[X]}
\end{aligned}
$$

3.5.7 Catalytic Mass Action with Substrate/Enzyme and Product/Enzyme Intermediate Complexes

pycellerator Arrow:

```
[A :=> B, mod[X], rates[k1, k2, k3, k4, k5, k6]]
```

Equivalent xlr8r Arrow:

$$\{\mathtt{A} \overset{\mathtt{X}}{\rightleftharpoons} \mathtt{B}, \mathtt{k1}, \mathtt{k2}, \dots, \mathtt{k6}\}$$

Typical biochemical notation:

$$A + X \underset{k_2}{\overset{k_1}{\rightleftarrows}} AX \underset{k_4}{\overset{k_3}{\rightleftarrows}} BX \underset{k_6}{\overset{k_5}{\rightleftarrows}} B + X$$

Interpreted differential equation terms: This arrow is expanded into the following collection of simple mass action arrows as described in section 3.5.2:

```
[A+X->A_X,k1]
[A_X->A+X,k2]
[A_X->B_X,k3]
[B_X->A_X\,k4]
[B_X->B+X,k5]
[B+X->B_X,k6]
```

These are then converted into terms in the differential equations as described above in 3.5.2:

$$\begin{aligned}
[\mathtt{A}]' &= k_2[\mathtt{A\_X}] - k_1[\mathtt{A}][\mathtt{X}] \\
[\mathtt{X}]' &= k_2[\mathtt{A\_X}] + k_5[\mathtt{B\_X}] - k_1[\mathtt{A}][\mathtt{X}] - k_6[\mathtt{B}][\mathtt{X}] \\
[\mathtt{B}]' &= k_5[\mathtt{B\_X}] - k_6[\mathtt{B}][\mathtt{X}] \\
[\mathtt{B\_X}]' &= k_3[\mathtt{A\_X}] - (k_4 + k_5)[\mathtt{B\_X}] + k_6[\mathtt{B}][\mathtt{X}] \\
[\mathtt{A\_X}]' &= k_4[\mathtt{B\_X}] - (k_2 + k3_)[\mathtt{A\_X}] + k_1[\mathtt{A}][\mathtt{X}]
\end{aligned}$$

3.6 Equilibrium / Steady-State Models

3.6.1 Michaelis-Menten-Henri (MMH) Reactions

pycellerator Arrows:

```
[A :-> B, MMH[K, v]]
[A :-> B, MMH[k1, k2, k3]]
[A :-> B, mod[X], MMH[K, v]]
[A :-> B, mod[X], MMH[k1, k2, k3]]}
```

Equivalent xlr8r Arrow:

$$\{\mathtt{A} \Longrightarrow \mathtt{B}, \mathtt{MM}[\mathtt{K}, \mathtt{v}]\}$$
$$\{\mathtt{A} \Longrightarrow \mathtt{B}, \mathtt{MM}[\mathtt{k1}, \mathtt{k2}, \mathtt{k3}]\}$$
$$\{\mathtt{A} \overset{\mathtt{X}}{\Longrightarrow} \mathtt{B}, \mathtt{MM}[\mathtt{K}, \mathtt{v}]\}$$
$$\{\mathtt{A} \overset{\mathtt{X}}{\Longrightarrow} \mathtt{B}, \mathtt{MM}[\mathtt{k1}, \mathtt{k2}, \mathtt{k3}]\}$$

Typical Biochemical Notation:

Non-standard.

Interpreted Differential Equation:

$$[\texttt{B}]' = -[\texttt{A}]' = \frac{v[\texttt{A}]}{K+[\texttt{A}]} \qquad \text{for first form}$$

$$[\texttt{B}]' = -[\texttt{A}]' = \frac{k_3[\texttt{A}]}{(k_2+k_3)/k_1+[\texttt{A}]} \qquad \text{for second form}$$

$$[\texttt{B}]' = -[\texttt{A}]' = \frac{v[\texttt{A}][\texttt{X}]}{K+[\texttt{A}]} \qquad \text{for third form}$$

$$[\texttt{B}]' = -[\texttt{A}]' = \frac{k_3[\texttt{A}][\texttt{X}]}{(k_2+k_3)/k_1+[\texttt{A}]} \qquad \text{for fourth form}$$

3.6.2 Monod-Wyman-Changeaux (MWC) Reactions

`pycellerator` Arrows:

1. Basic MWC:

```
[S==>P,mod[E],MWC[k,n,c,L,K]]
```

2. MWC with additional activators and inhibitors:

```
[[S1,S2,..]==>P,mod[E,[A1,A2,..],[I1,I2,..]],MWC[k,n,c,L,},K1]]
```

where **K1** = `[[KS1,KS2,...],[KA1,KA1,..], [KI1,KI2,...]]`.

3. MWC with Competitive inhibition.

```
[[S1,S2,..]==>P}, mod[E,[A1,A2,..],[I1,I2,..],X ], MWC[k,n,c,L,}, K2 ]]
```

where **X** has the form

```
     [[[CS11, CS12,..],     # S1 Competitive Inhibitors
       [CS21, CS22,..],..]  # S2 Competitive Inhibitors
      [[CA11, CA11,..],     # A1 Competitive Inhibitors
       [CA21, CA22,..],..]  # A2 Competitive Inhibitors
```

and **K2**= `[[KS1, KS2,..], [KA1, KA2, ..], [KI1, KI2, ..], [KCS11, CS12, ..], [KCS21, KCS22, ..], [KCA11, KCA12, ..], [KCA21, KCA22, ..]]`.

Equivalent `xlr8r` Arrow:

$$\{\texttt{S} \overset{\texttt{Enz}}{\Longrightarrow} \texttt{P},\ \texttt{MWC[k,n,c,L,K]}\} \qquad \text{(for (1))}$$

$$\{\underset{\{\{\texttt{A1,A2,..}\},\{\texttt{I1,I2,..}\}\}}{\texttt{S} \overset{\texttt{Enz}}{\Longrightarrow} \texttt{P}},\ \texttt{MWC[k,n,c,L,K,..]}\} \qquad \text{(for (2))}$$

$$\{\underset{\{\{\texttt{A1,A2,..}\},\{\texttt{I1,I2,..}\},\{\texttt{CS1,..}\},..,\{\texttt{CA1,..}\},..\}}{\texttt{S} \overset{\texttt{Enz}}{\Longrightarrow} \texttt{P}},\ \texttt{MWC[k,n,c,L,K,..]}\} \qquad \text{(for (3))}$$

Typical Biochemical Notation:

Not standardized.

Interpreted Differential Equation:

The ODE terms are based on the theory of [12]. Let $c =$ `[S]/K`. Then

$$\texttt{[P]}' = -\texttt{[S]}' = \texttt{[E]}\frac{s(1+s)^{n-1} + Lsc(1+sc)^{n-1}}{(1+s)^n + L(1+sc)^{n-1}} \qquad \text{for form (1)}$$

The terms for forms (2) and (3) are described in [13]. For the first arrow, let

$$s_j = \frac{\texttt{[S]}_j}{\texttt{K}_{\texttt{S}j}}, a_j = \frac{\texttt{[A]}_j}{\texttt{K}_{\texttt{a}j}}, i_j = \frac{\texttt{[I]}_j}{\texttt{K}_{\texttt{I}j}}$$

Then

$$\texttt{[P]}' = -\texttt{[S]}' = \texttt{[E]}\frac{\prod(1+a_j)^n \prod s_j \prod(1+s_j)^{n-1} + L\prod(1+i_j)^n \prod(cs_j)\prod(1+cs_j)^{n-1}}{\prod(1+a_j)^n \prod(1+s_j)^n + L\prod(1+i_j)^n\prod(1+cs_j)^{n-1}} \qquad \text{for form (2)}$$

For form (3) we can also normalize the competitive inhibitors as

$$\overline{s_j} = c\sum_k \frac{\texttt{[CS]}_{jk}}{\texttt{K}_{\texttt{CS}jk}}, \qquad \overline{a_j} = c\sum_k \frac{\texttt{[CA]}_{jk}}{\texttt{K}_{\texttt{CA}jk}}$$

and then

$$\begin{aligned}\texttt{[P]}' &= -\texttt{[S]}'\\ &= \texttt{[E]}\frac{\prod(1+a_j\overline{a_j})^n \prod s_j \prod(1+s_j+\overline{s_j})^{n-1} + L\prod(1+i_j)^n\prod(cs_j)\prod(1+cs_j+\overline{sj})^{n-1}}{\prod(1+a_j+\overline{a_j})^n\prod(1+s_j)^n + L\prod(1+i_j)^n\prod(1+cs_j+\overline{s_j})^{n-1}} \qquad \text{for form(3)}\end{aligned}$$

3.6.3 Non-regulatory Hill Functions

These are described below in section 3.7.1.

3.7 Regulatory Arrows

3.7.1 Hill Functions

pycellerator Arrows:

There are three Hill function forms.

```
[A |-> B, Hill[v, n, K, a, T]]                            # Hill (1)
[[P,Q,...] |-> R, Hill[v, n, K, a, [TP,TQ,...]]]          # Hill (2)
[[X,Y,Z,...] |--> U, mod[E], Hill[v, n, K, a, [TX,TY, TZ,...]]] # Hill (3)
```

Note that the third form (Hill(3)) is not actually a regulatory arrow, as each of the species on the left is also affected by the arrow.

Equivalent xlr8r Arrows:

$$\{\texttt{A} \mapsto \texttt{B}, \texttt{Hill[v, n, K, a, T]}\} \qquad \text{for Hill (1)}$$

$$\{\{\texttt{P},\texttt{Q},\ldots\} \mapsto \texttt{R}, \texttt{Hill[v, n, K, a, }\{\texttt{TP, TQ, ...}\}\texttt{]}\} \qquad \text{for Hill (2)}$$

$$\{\{\texttt{X},\texttt{Y},\texttt{Z},\ldots\} \overset{\texttt{E}}{\mapsto} \texttt{U}, \texttt{Hill[v, n, K, a, }\{\texttt{TX, TY, TZ, ...}\}\texttt{]}\} \qquad \text{for Hill (3)}$$

Typical Biochemical Notation:

Not standardized.

Interpreted Differential Equation:

For (Hill (1)) and (Hill (2)) only the amount of the product changes, and not the amount of any of the reactants on the left-hand-side of the reaction:

$$[\mathtt{B}]' = \frac{v(a+T[\mathtt{A}])^n}{K^n + (a+T[\mathtt{A}])^n} \quad \text{for Hill (1)} \tag{3.1}$$

$$[\mathtt{R}]' = \frac{v(a+T_P[\mathtt{P}]+T_Q[\mathtt{Q}]+\cdots)^n}{K^n + (a+T_P[\mathtt{P}]+T_Q[\mathtt{Q}]+\cdots)^n} \quad \text{for Hill (2)} \tag{3.2}$$

$$[\mathtt{U}]' = \frac{v[\mathtt{E}](a+T_X[\mathtt{X}]+T_Y[\mathtt{Y}]+\cdots)^n}{K^n + (a+T_X[\mathtt{X}]+T_Y[\mathtt{Y}]+\cdots)^n} \quad \text{for Hill (3)} \tag{3.3}$$

For the third form (Hill (3)), the products on the left all change also,

$$[\mathtt{X}]' = [\mathtt{Y}]' = \cdots = -\frac{v[\mathtt{E}](a+T_X[\mathtt{X}]+T_Y[\mathtt{Y}]+\cdots)^n}{K^n + (a+T_X[\mathtt{X}]+T_Y[\mathtt{Y}]+\cdots)^n} \quad \text{for Hill (3) only} \tag{3.4}$$

Difference from `xlr8r`: In `xlr8r` the combination of arrows

$$\begin{aligned} \{&\{\mathtt{A1} \mapsto \mathtt{B}, \mathtt{Hill}[\mathtt{v},\mathtt{n},\mathtt{K},\mathtt{a},\mathtt{T1}]\},\\ &\{\mathtt{A2} \mapsto \mathtt{B}, \mathtt{Hill}[\mathtt{v},\mathtt{n},\mathtt{K},\mathtt{a},\mathtt{T2}]\},\\ &\{\mathtt{A3} \mapsto \mathtt{B}, \mathtt{Hill}[\mathtt{v},\mathtt{n},\mathtt{K},\mathtt{a},\mathtt{T3}]\},\dots\} \end{aligned} \tag{3.5}$$

will be given ODE terms

$$[\mathtt{B}]' = \frac{v(a+T_1[\mathtt{A1}]+T_2[\mathtt{A2}]+T_3[\mathtt{A3}]+\cdots)^n}{K^n + (a+T_1[\mathtt{A1}]+T_2[\mathtt{A2}]+T_3[\mathtt{A3}]+\cdots)^n} \tag{3.6}$$

whereas in `pycellerator` it will be interpreted as:

$$[\mathtt{B}]' = \frac{v(a+T_1[\mathtt{A1}])^n}{K^n + (a+T_1[\mathtt{A1}])^n} + \frac{v(a+T_2[\mathtt{A2}])^n}{K^n + (a+T_2[\mathtt{A2}])^n} + \frac{v(a+T_3[\mathtt{A3}])^n}{K^n + (a+T_3[\mathtt{A3}])^n} \tag{3.7}$$

To get an interpretation of the form (3.6) one must use the arrow form Hill (2). Unfortunately, it is not possible to represent equations of the form given by (3.7) in `xlr8r`, even though such combinations are frequently used in the modeling literature, unless the user explicitly coded the equation as part of the rate constant. This is the motivation for the change in `pycellerator`.

3.7.2 GRN Arrows

`pycellerator` Arrows:

There are three forms of GRN arrows in pycellerator.

```
[A |-> B, GRN[v, T, n, h]]                             # GRN (1)
[[P,Q,...] |-> R, GRN[v, [TP,TQ,...], n, h]]           # GRN (2)
[[X,Y,...] |--> U, mod[E], GRN[v, [TX,TY,...], n, h]] # GRN (3)
```

Equivalent `xlr8r` Arrow:

$$\{\mathtt{A} \mapsto \mathtt{B}, \mathtt{GRN[v,T,n,h]}\} \qquad \text{for GRN (1)}$$
$$\{\{\mathtt{P},\mathtt{Q},\ldots\} \mapsto \mathtt{R}, \mathtt{GRN[v,\ \{TP,\ TQ,\ \ldots\},n,h]}\} \qquad \text{for GRN (2)}$$
$$\{\{\mathtt{X},\mathtt{Y},\ldots\} \overset{\mathtt{E}}{\mapsto} \mathtt{U}, \mathtt{GRN[v,\ \{TX,\ TY,\ \ldots\},n,h]}\} \qquad \text{for GRN (3)}$$

Note that the third form of this arrow, GRN (3) is not actually implemented in `xlr8r`, only in pycellerator.

Typical Biochemical Notation:

Not standardized.

Interpreted Differential Equation:

$$\mathtt{[B]}' = \frac{v}{1+\exp(-h-T\,\mathtt{[A]}^n)} \qquad \text{for GRN (1)} \tag{3.8}$$
$$\mathtt{[R]}' = \frac{v}{1+\exp(-h-T_P\,\mathtt{[P]}^n - T_Q\,\mathtt{[Q]}^n + \cdots)} \qquad \text{for GRN (2)} \tag{3.9}$$
$$\mathtt{[U]}' = \frac{v}{1+\exp(-h-T_X\,\mathtt{[X]}^n - T_Y\,\mathtt{[Y]}^n + \cdots)} \qquad \text{for GRN(3)} \tag{3.10}$$

Difference from `xlr8r`: In `xlr8r` the combination of arrows

$$\begin{aligned}&\{\{\mathtt{A1} \mapsto \mathtt{B}, \mathtt{GRN}[\mathtt{v},\mathtt{T1},\mathtt{n},\mathtt{h}]\},\\ &\{\mathtt{A2} \mapsto \mathtt{B}, \mathtt{GRN}[\mathtt{v},\mathtt{T2},\mathtt{n},\mathtt{h}]\},\\ &\{\mathtt{A3} \mapsto \mathtt{B}, \mathtt{GRN}[\mathtt{v},\mathtt{T2},\mathtt{n},\mathtt{h}]\},\ldots\}\end{aligned} \tag{3.11}$$

will be given ODE terms

$$\mathtt{[B]}' = \frac{v}{1+\exp\left(-h-T_1\,\mathtt{[A1]}^n - T_2\,\mathtt{[A2]}^n - \cdots\right)} \tag{3.12}$$

whereas in `pycellerator` it will be given the ODE

$$\mathtt{[B]}' = \frac{v}{1+\exp\left(-h-T_1\,\mathtt{[A1]}^n\right)} + \frac{v}{1+\exp\left(-h-T_2\,\mathtt{[A2]}^n\right)} + \cdots$$

The version given by equation (3.12) can still be obtained by using the arrow form **`[[A1,A2,..]|-->B]`**.

3.7.3 S-Systems

`pycellerator` Arrows:

There are two forms of S-System arrows in pycellerator:

```
[S |-> P, SSystem[tau, a, b, g, h]]                          # S-System (1)
[[S1,S2,..] |-> P, SSystem[tau, a, b, [g1,g2,..], [h1,h2,..]] # S-System (2)
```

Equivalent `xlr8r` Arrow:

$$\{\mathtt{S{\mapsto}P,\ SSystem[tau,a,b,g,h]}\} \qquad \text{for S-System (1)}$$
$$\{\mathtt{S1,S2,..{\mapsto}P,\ SSystem[tau,a,b,\{g1,g2,..\},\{g1,h2,..\}]}\} \qquad \text{for S-System (2)}$$

Typical Biochemical Notation:

Not standardized.

Interpreted Differential Equation:

$$\texttt{[P]}' = \frac{1}{\tau}(a\,\texttt{[S]}^g - b\,\texttt{[S]}^h) \qquad \text{for S-System(1)} \qquad (3.13)$$

$$\texttt{[P]}' = \frac{1}{\tau}(a\prod_i \texttt{[Si]}_i^{g_i} - b\prod_i \texttt{[S]}_i^{h_i}) \qquad \text{for S-System (2)} \qquad (3.14)$$

S-System reactions are numerically unstable when any $\texttt{[S]}_i \to 0$ if the corresponding $g_i < 0$, because this would lead to a division by zero condition. The numerical "trick" that is used to fix this is to set concentrations to a small number ϵ rather than 0 in this case. The default value for ϵ is 10^{-37} but may be reset in the solver module with the keyword `-epsilon` *`value`*.

This reaction is described in detail in [14, 15].

3.7.4 NHCA

`pycellerator` Arrows:

There are two pycellerator Arrows for non-hierarchical cooperative activation.

```
[A |-> B, mod[X], NHCA[v,TP,TM,n,m,k]]                            # NHCA1
[[A1,A2,..] |-> B, mod[X], NHCA[v,[TP1,..],[TM1,..],[n1,..],m,k]]  # NHCA2
```

Equivalent `xlr8r` Arrow:

$$\{\texttt{A} \overset{\texttt{X}}{\mapsto} \texttt{B, NHCA[v,}\{\texttt{TP,TM}\}\texttt{,n,m,k]}\} \qquad \text{for NHCA1}$$

$$\{\{\texttt{A1,A2,..}\} \overset{\texttt{X}}{\mapsto} \texttt{B, NHCA[v,}\{\{\texttt{TP1,TP2,..}\},\{\texttt{TM1,TM2,..}\}\},\{\texttt{n1,n2,..}\}\texttt{,m,k]}\} \qquad \text{for NHCA2}$$

Note that the second form, (NHCA2), is not actually implemented in `xlr8r`, only in pycellerator.

Typical Biochemical Notation:

Not standardized.

Interpreted Differential Equation:

This reaction is described in more detail in [17].

$$\texttt{[B]}' = v\,\texttt{[E]}\,\frac{(1+T_P\,\texttt{[A]}^n)^m}{k(1+T_M\,\texttt{[A]}^n)^m + (1+T_P\,\texttt{[A]}^n)^m} \qquad \text{for NHCA1}$$

$$\texttt{[B]}' = v\,\texttt{[E]}\,\frac{\prod_i(1+T_{Pi}\,\texttt{[A]}_i^{n_i})^m}{k\prod_i(1+T_{Mi}\,\texttt{[A]}_i^{n_i})^m + \prod_i(1+T_{Pi}\,\texttt{[A]}_i^{n_i})^m} \qquad \text{for NHCA2}$$

3.7.5 Rational Functions

`pycellerator` Arrows:

```
 [[[X1,X2,..],[Y1,Y2,..]]]==>Z,
      rational[[a0,a1,a2,...],    # coefficients of numerator
               [d0,d1,d2,...],    # coefficients of denominator
               [m0,m1,m2,...],    # exponents, numerator
               [n0,n1,n2,...]]]    # exponents, denominator
```

As with `xlr8r`, any of the input species may represent a product, for example,

```
[[[O*S,O*S*N],[O,O*S,O*S*N,O*G]] ==>N,
               rational[[e0,e1,e2],[1,f0,f1,f2,f3],[],[]]]
```

The asterisks must be specified between the species if the product-form is used.

Equivalent `xlr8r` Arrow:

```
{{{X1,X2,..},{Y1,Y2,..}}} ⟹ Z, rational{{a0,a1,a2,...},
                                         {d0,d1,d2,...},
                                         {m1,m2,...},
                                         {n1,n2,...}}}
```

Typical Biochemical Notation:

Not standardized.

Interpreted Differential Equation: For the input reaction

```
[[[X1,X2,X3,...], [Y1,Y2,...]]==>Z,
   rational[[a0,a1,a2,a3,...], [d0,d1,d2,d3,...],
            [m0,m1,m2,m3,...], [n0,n1,n2,n3,...]]]}
```

the ODE term produced is:

$$[\mathtt{Z}]' = \frac{a_0^{m_0} + a_1[\mathtt{X1}]^{m_1} + a_2[\mathtt{X2}]^{m_2} + a_3[\mathtt{X3}]^{m_2} + \cdots}{d_0^{n_0} + d_1[\mathtt{Y1}]^{n_1} + d_2[\mathtt{Y2}]^{n_2} + d_3[\mathtt{Y3}]^{n_3} + \cdots}$$

Difference From `xlr8r`: In `pycellerator` the first value in each list of exponents is applied to the leading constants, rather than to the first parameter, so that `pycellerator` expects the same number of exponents as coefficients. In `xlr8r` there is no way to specify an exponent on the first coefficient, and `xlr8r` expects one fewer exponents than coefficients. Thus `xlr8r` would ignore the extra exponents and interpret the above reaction as follows:

$$[\mathtt{Z}]' = \frac{a_0 + a_1[\mathtt{X1}]^{m_0} + a_2[\mathtt{X2}]^{m_1} + a_3[\mathtt{X3}]^{m_2} + \cdots}{d_0 + d_1[\mathtt{Y1}]^{n_0} + d_2[\mathtt{Y2}]^{n_1} + d_3[\mathtt{Y3}]^{n_2} + \cdots}$$

3.7.6 User Defined Regulatory Arrows

`pycellerator` Arrows:

There are several different forms. In the first form, there is a single reactant forming a single product.

```
[A |-> B, USER[v, T, n, h, f]]                          # USER  (1)
```

In the second form, multiple reactants interact to produce a single product vectorally.

```
[[P1,P2,...]|->  Q, USER[v, T, n, h, f]]                # USER (2)
```

In the third form, a modifier can be specified.

```
[[X1,X2,...]|--> Y, mod[E], USER[v, T, n, h, f]]}}    # USER (3)
```

In USER(2) and USER (3) the expressions for **T** and **n** may be either single symbols, numbers, or lists of length up to the length of the number of reactants on the left-hand side of the arrow. If there are fewer values, the list will be extended with the final value provided (the rightmost value).

In each case, the function `f` must be a valid Python `lambda` expression of a single variable, enclosed in quotation marks. For example,

```
[[P,Q] |-> R, USER[v, T, n, h, "lambda x: 1/(1+exp(-x))"]]}
```

represents a `USER` equivalent version of the standard GRN reaction.

Equivalent `xlr8r` Arrow:

$$\{\texttt{A} \mapsto \texttt{ B, USER[v, T, n, h, f]}\} \qquad \text{for USER (1)}$$
$$\{\{\texttt{P1,P2,...}\} \mapsto \texttt{ Q, USER[v, T, n, h, f]}\} \qquad \text{for USER (2)}$$

The equivalent of example in xlr8r would be

$$\{\{\texttt{P,Q}\} \mapsto \texttt{ R, USER[v, T, n, h, 1/(1+Exp[\#])\&]}\}$$

In `xlr8r` there is no equivalent reaction for USER (3); it is new in pycellerator.

Typical Biochemical Notation:

Not standardized.

Interpreted Differential Equation: For the input reactions USER (1) through USER (3),

$$[\mathtt{B}]' = \mathtt{v}f(\mathtt{h} - \mathtt{T}[\mathtt{A}]^{\mathtt{n}}) \qquad \text{for USER (1)}$$
$$[\mathtt{Q}]' = \mathtt{v}f(\mathtt{h} - \mathtt{T}_1[\mathtt{P1}]^{\mathtt{n}_1} - \mathtt{T}_2[\mathtt{P2}]^{\mathtt{n}_2} - \cdots) \qquad \text{for USER (2)}$$
$$[\mathtt{Y}]' = \mathtt{v}[\mathtt{E}]f(\mathtt{h} - \mathtt{T}_1[\mathtt{X1}]^{\mathtt{n}_1} - \mathtt{T}_2[\mathtt{X2}]^{\mathtt{n}_2} - \cdots) \qquad \text{for USER (3)}$$

Difference From `xlr8r`:

In `pycellerator` the head of the rate list must be the word `USER`; in `xlr8r` it may be any symbol not otherwise assigned.

In `pycellerator` the function `f` must be a Python `lambda` expression. In `xlr8r`, while a Mathematica lambda expression (otherwise known as a pure function, such as `1/(1+Exp[#])&`) is permitted but it is not required, and the name of any previously defined function may be substituted.

3.8 User Defined Stoichiometric Arrows

`pycellerator` Arrows:

```
[s1*A1+s2*A2+... -> q1*A1+q2*A2+..., using["expression"]]
```

where the `s1,s2,..` and `q1,q2,...` are (possibly zero) stoichiometries and `"`*`expression`*`"` is a valid mathematical expression expressed in syntactically correct python. It must be enclosed in quotation marks. The asterisks between the stoichiometries and the species in the reaction are optional.

Interpreted Differential Equation:

$$\frac{d\,[\mathtt{X}]}{dt} = (\mathtt{qi} - \mathtt{si}) \times (\text{expression}) \tag{3.15}$$

For example:

```
[B + 3S -> 4P + 5S, using["A*B**3"]]
```

returns the collection of ODE terms:

$$[\mathtt{P}]' = 4\,[\mathtt{A}]\,[\mathtt{B}]^3$$
$$[\mathtt{S}]' = 2\,[\mathtt{A}]\,[\mathtt{B}]^3$$
$$[\mathtt{B}]' = -\,[\mathtt{A}]\,[\mathtt{B}]^3$$

Compare with User Defined Regulatory Arrows (Section 3.7.6) and Equations as Rate Constants (Section 3.2), which are similar but produce slightly different results.

3.9 Cascades

Any mass action, catalytic mass action, MMH, Hill Function, GRN, S-System, or NHCA reaction can be written in a cascade as a single reaction. A **cascade** is defined as sequence of repeated reactions with the same arrow and the same rate constants. For example, the reactions

```
    [A => B, mod[E], rates[k1,k2,k3]]
    [B => C, mod[F], rates[k1,k2,k3]]
    [C => D, mod[G], rates[k1,k2,k3]]
```

can be written as a single reaction cascade:

```
    [A => B => C => D, mod[E, F, G], rates[k1,k2,k3]]
```

Reactions without modifies can also be written as cascades:

```
    [P :-> Q :-> R, MMH[KD, v]]
```

which represents the pair of reactions

```
    [P :-> Q, MMH[KD, v]]
    [Q :-> R, MMH[KD, v]]
```

Note that if different rate constants are required at different states in the cascade, the reactions must be written separately, and not as part of a cascade.

Different types of arrows cannot be combined together, thus a cascade cannot be written as `A->B|->C` but must be written as two separate reactions.

An example of a model with a cascade is given by the MAP-Kinase model with oscillations in section 5.1.

3.10 Flux Arrows

Reactions that represent fluxes are a fundamentally different type of entity than reactions used in kinetic models, as described in theprevious sections. This is because `Flux` reactions do not (necessarily) have a rate law (or ODE) associated with them, although they normally have a total rate, given by product of a velocity and a stoichiometry.

Normally a model will be composed **either** entirely of kinetic arrows **or** entirely of flux arrows. In the present implementation of `pycellerator` one is not allowed to combine the two in a model. The Mathematica implementation of `xlr8r` does not support `Flux` reactions at all (at the current time).

The format of a flux arrow is

$$\texttt{[}\mathit{lhs}\texttt{->}\mathit{rhs}\texttt{,Flux[}\mathit{low}\texttt{<}\mathit{id}\texttt{<}\mathit{up}\texttt{,}\mathit{obj}\texttt{,}\mathit{flux}\texttt{]]} \tag{3.16}$$

where *lhs* and *rhs* are stoichiometric expressions such as `3A + B` or `X`, and the other terms are defined in the following table:

`pycellerator`	COBRA SBML Variable	Description
`low`	`LOWER_BOUND`	numerical lower bound or `-inf` if unbounded
`up`	`UPPER_BOUND`	numerical upper bound or `inf` if unbounded
`id`	SBML reaction `Id`	variable name used to refer to reaction flux
`obj`	`OBJECTIVE_COEFFICIENT`	Used to determine objective function component of **f** vector.
`flux`	`FLUX_VALUE`	Value to assign to flux. Not required to perform FBA optimization. Sometimes used for SBML L2.4 kinetic law.

An example is

```
    [ES -> E + S, Flux[0 < v < 1, 1, 0]}
```

Note that the "less-than" sign must be used, though the constraint interval is typically closed, and not open. In the case of equality, a constraint of the form

$$\texttt{1} < \texttt{v} < \texttt{1} \tag{3.17}$$

would be used to indicate a constraint meaning $\texttt{v} = 1$.

Flux optimization will solve the linear programming problem

$$\begin{aligned} \text{maximize} \quad & \mathbf{v}^{\mathbf{T}}\mathbf{f} && (3.18) \\ \text{subject to} \quad & \mathbf{N}\mathbf{v} = \mathbf{0} && (3.19) \\ \text{and} \quad & \mathit{low}_1 < v_1 < \mathit{up}_1 && (3.20) \\ \text{and} \quad & \mathit{low}_2 < v_2 < \mathit{up}_2 && (3.21) \\ & \vdots \end{aligned}$$

where $\mathbf{v}$ is the vector of fluxes $(v_1, v_2, \dots)^{\mathtt{T}}$, $\mathbf{N}$ is the stoichiometry matrix, and $\mathbf{f}$ is the vector of objective coefficients.

Chapter 4

pycellerator Function Reference

Conventions used in this chapter:

For the command line syntax:

`typewriter font` is used to represent required input exactly as it should be typed

italicized Roman font is used to represent input that should be replaced with a value, such as the name of an input file.

Items enclosed in square brackets, e.g., `[ like this ]` are optional. This remark only applies to expressions typed on the command line (e.g., the terminal in linux or MacOS, or at the command prompt in Windows). It **does not apply** to reactions.

The `$` is used to indicate the command prompt is several examples.

Reaction Syntax:

Examples of reactions, such as those that should be written in input files, are written in `typewriter` font `like this`. Note that reaction syntax requires the use of square brackets, and the contents of square brackets are not optional in this case.

4.1 Command Line Interface

The generic command line syntax is

```
$ python pycellerator.py keyword [options]
```

where **`keyword`** is one of **`EXPAND`**, **`INTERPRET`**, **`CONVERT`**, **`PARSE`**, **`SOLVE`**, **`SBML`**, **`FLUX`** and **`VERSION`**, and `[options]` is an optional sequence of command line options that is difference for each keyword. The case (upper or lower) of the tt keyword does not matter. The syntax and options for each keyword are discussed in the following sections.

4.2 parser

The parser converts from text-based arrow notation into an internal python class (data structure) called a `Reaction`.

Command Line Syntax:

```
$ python pycellerator parse -in \textit{filename} [-dump] [-trace]}
```

```
$ python parser.py -in \textit{filename} [-dump] [-trace]}
```

The `in` *`filename`* specifies the input file name, which contains a list of text reactions. An example is given by the following:

```
[X  <=> XP,   mod[Z, ZPP], rates[a, d, k, 0, a, d, k]]  # first reaction
[XP <=> XPP,  mod[Z, ZPP], rates[a, d, k, 0, a, d, k]]  # second reaction
[Y  <=> YP,   mod[X, XPP], rates[a, d, k, 0, a, d, k]]
[YP <=> YPP,  mod[X, XPP], rates[a, d, k, 0, a, d, k]]
[Z  <=> ZP,   mod[Y, YPP], rates[a, d, k, 0, a, d, k]]
[ZP <=> ZPP,  mod[Y, YPP], rates[a, d, k, 0, a, d, k]]
```

One reaction may be placed per line; additional white space is ignored. The # character indicates a comment; anything written to the right of this character will be ignored.

If the `-trace` option is used, the text reaction will be written to the screen after each reaction is successfully read and parsed.

If the `-dump` option is used, the `Dump()` method will be applied (and written to the screen) for each reaction after it is processed, whether the reaction is successfully parsed or not.

Command Line in Shell:

```
>>> import pycellerator
>>> pycellerator.run("parse -in  filename [-dump]  [-trace]")
```

API (functions):

`parser(*keywords)`

input:	(keyword arguments only)
return value:	list of `Reactionss`

Optional keyword arguments:

`inputfile=""`	Name of input file containing reactions.
`trace=False`	Turns on tracing, same as command line `-trace`

`ParseArrow(r)`

input:	`r`: String reaction in pycellerator format.
return value:	`Reaction` object representing the input text reaction. Operationally, there is no difference between invoking `ParseArrow(r)` and `Reaction(r)`; the difference is that one invokes the class directly and the other does it via a functional interface.

API (class):

`Reaction(r)`

input:	`r`: String reaction in pycellerator format.

Methods:

`r.Input()`	value of the original text reaction
`r.Parsed()`	output of the `pyparsing` module
`r.Number()`	number of sequential calls to parser
`r.Length()`	length of the `pyparsing` object
`r.ArrowType()`	Cellerator arrow type as a string
`r.Arrow()`	Actual arrow as a string
`r.LHS()`	List of reactants
`r.RHS()`	List of products
`r.LH_STOIC()`	Stoichiometry of reactants (mass action only)
`r.RH_STOIC()`	Stoichiometry of products (mass action only)
`r.MHS()`	List of modifiers
`r.Rates()`	List of rate constants or parameters in reaction
`r.Dump()`	Produce a formatted dump of the `Reaction` class object

Comparison of `Arrow()` and `ArrowType()`. See previous chapter for descriptions of corresponding reactions.

`r.Arrow()`	`r.ArrowType()`
`"->"`	`"Mass Action"`
`"-->"`	`"Mass Action"`
`"<->"`	`"Mass Action"`
`"=>"`	`"Mass Action"`
`"<=>"`	`"Mass Action"`
`":->"`	`"MMH"`
`":-->"`	`"MMH"`
`":=>"`	`"Mass Action"`
`"|->"`	`"GRN"`
`|-->"`	`"GRN"`
`"==>"`	`"Multiuse-Arrow"`

Examples.

1. Input with -trace

```
$more catalytic.dat
 [X => Y, mod[E], rates[k1,k2,k3]]
 [Y => X, mod[F], rates[k4,k5,k6]]
$python pycellerator.py -in catalytic.dat -trace
 py[cellerator]: parser (2015-03-02 17:58:18)
 input: [X => Y, mod[E], rates[k1,k2,k3]]
 input: [Y => X, mod[F], rates[k4,k5,k6]]
 [<__main__.Reaction object at 0x7f0442369a50>,
   <__main__.Reaction object at 0xe47dd0>]
```

2. Input with -dump

```
$cat sto.dat
  [ 3A + 4B -> P + Q + R,  k]
$ python pyx.py -in sto.dat -dump
py[xlr8r]: parser (2012-03-02 18:08:54)
====================== Reaction Number  1  ======================
Input:     [ 3A + 4B -> P + Q + R,  k]
Parsed:    [(['3', 'A', '4', 'B'], {}), '->',
            (['P', 'Q', 'R'], {}), (['k'], {})]
Arrow:     ->
ArrowType: Mass Action
LHS:       ['A', 'B']
LH_STOIC:  ['3', '4']
RHS:       ['P', 'Q', 'R']
RH_STOIC:  ['1', '1', '1']
modifiers: []
rates:     ['k']
[<__main__.Reaction object at 0x7f6b1c3d2c90>]
```

4.3 expander

Normally there is no need for the user to call the expander directly.

The expander module expands complex py[xlr8r] reactions into their constituent parts; for example, the catalytic reaction

$$S + E \underset{k_2}{\overset{k_1}{\rightleftharpoons}} SE \overset{k_3}{\rightarrow} P + E$$

is broken down into its constituent reactions

$$\begin{aligned} S + E &\overset{k_1}{\rightarrow} SE \\ SE &\rightarrow \overset{k_2}{S} + E \\ SE &\rightarrow \overset{k_3}{P} + E \end{aligned}$$

The input to the expander is a list of reactions in text from, or the name of a file containing a list of such reactions, one line per file.

The output of the expander is a list of the expanded reactions, either as `Reaction` objects, or a list of text reactions, or both.

Command Line Syntax:

```
$ python expander.py -in  inputfile  [-dump] [-out   outputfile]
```

```
$ python pycellerator.py expand -in inputfile [-dump] [-out  outputfile]
```

The *`inputfile`* may be either a reaction file or a model file. If it is a model file it must have file extension `.model`.

If the `-dump` option is specified the text output of the expander will be spewed to the screen as a list of text output reactions (compatible in format to input text reactions).

If the `-out` option is specified the text output of the expander will be written to the requested file. If no file name is given a default file name will be generated (e.g., `expanded-reactions.out`).

Existing files will not be over-written; if the requested file already exists a new file name will be generated that is based on the requested file name but has a unique string attached to the end of it.

Command Line in Shell:

```
>>> import pycellerator
>>> pycellerator.run ("expand -in inputfile [-dump] [-out } outputfile]")
```

API:

`expand(r)`

input:	single `Reaction` or list of `Reactions`
return value:	list of expanded `Reactions` (see parser)

Optional keyword arguments:
none

`expandReactions(filename)`

input:	name of input file
return value:	list of `Reactions` (see parser)

Optional keyword arguments:

`dump = False`	if `True`, print text reactions to screen
`text = False`	if `True`, return list of text reactions instead of `Reactions`.

Example Input File: Here is an example of a typical input file, followed by a full example of the dump command using th an shorter input file.

```
[X  <=> XP,   mod[Z, ZPP], rates[a, d, k, 0, a, d, k]]
[XP <=> XPP,  mod[Z, ZPP], rates[a, d, k, 0, a, d, k]]
[Y  <=> YP,   mod[X, XPP], rates[a, d, k, 0, a, d, k]]
[YP <=> YPP,  mod[X, XPP], rates[a, d, k, 0, a, d, k]]
[Z  <=> ZP,   mod[Y, YPP], rates[a, d, k, 0, a, d, k]]
[ZP <=> ZPP,  mod[Y, YPP], rates[a, d, k, 0, a, d, k]]
```

Full Example In the following example, we will dump a model that is generated from a single line file.

```
$ more test.dat
  [X :=> Y,   mod[Z], rates[k1,k2,k3,k4,k5]]
$
$ python expander.py -in test.dat -dump -out test.out
py[cellerator]: expander (2015-09-01 10:39:34)
[X+Z->X_Z,k1]
[X_Z->X+Z,k2]
[X_Z->Z_Y,k3]
[Z_Y->X_Z,k4]
```

```
[Z_Y->Y+Z,k5]
[Y+Z->Z_Y,0]
Output written to: test.out
$
$ more test.out
[X+Z->X_Z,k1]
[X_Z->X+Z,k2]
[X_Z->Z_Y,k3]
[Z_Y->X_Z,k4]
[Z_Y->Y+Z,k5]
[Y+Z->Z_Y,0]
$
```

4.4 interpreter

The interpreter converts reactions from arrow format into differential equations. The functionality of the interpreter module is analogous to that of the Cellerator `interpret` command.

Command Line Syntax:

```
 $ python interpreter.py -in filename -out filename]
     [ -dump ]
     [ -format FORMAT]
     [ -frozen variable variable ...]
```

```
  $ python pycellerator.py interpret -in  filename -out filename]
     [ -dump ]
     [ -format FORMAT]
     [ -frozen variable variable ...]
```

The input *filename* may be either a reaction file or a model file. If it is a model file it must have a file extension of **`.model`**. Here is an example reaction file:

```
[Br + BrO3 -> HBrO2 + HOBr, k1]
[Br + HBrO2 -> 2*HOBr, k2]
[BrO3 + HBrO2 -> 2*Ce + 2*HBrO2, k3]
[2*HBrO2-> BrO3 + HOBr, k4]
[Ce -> 0.5*Br, k5]
```

If the **`-dump`** option is specified the **`Reaction`** object database will be dumped to the screen.

If the **`-out`** option is specified the text output of the interpreter will be written to the requested file. If no file name is given a default file name will be generated (e.g., **`interpreted -out reactions.out`**).

If the **`-out`** option is not used, the result will be written to the screen.

Existing files will not be over-written; if the requested file already exists a new file name will be generated that is based on the requested file name but has a unique string attached to the end of it.

Output formats are **`ODE`** (default), **`CODE`**, **`PYTHON`**, **`LATEX`**, **`DICT`**, **`JACOBIAN`**, **`STEADYSTATE`** and are not case-sensitive.

-format ODE will return a text-formatted table of differential equations. Here is an example of **ODE** format:

```
BrO3' = -Br*BrO3*k1 - BrO3*HBrO2*k3 + k4*HBrO2**2.0
HOBr' = Br*BrO3*k1 + 2.0*Br*HBrO2*k2 + k4*HBrO2**2.0
Br' = 0.5*Ce*k5 - Br*BrO3*k1 - Br*HBrO2*k2
HBrO2' = Br*BrO3*k1 + BrO3*HBrO2*k3 - Br*HBrO2*k2 - 2.0*k4*HBrO2**2.0
Ce' = -Ce*k5 + 2.0*BrO3*HBrO2*k3
```

-format CODE or **-format=PYTHON** will return a python function that will evaluate the differential equations. Here is an example of the output of the **CODE** or **PYTHON** format:

```
def f(y,t):
    y[0] = BrO3
    y[0] = HOBr
    y[0] = Br
    y[0] = HBrO2
    y[0] = Ce
    yp[0] = -Br*BrO3*k1 - BrO3*HBrO2*k3 + k4*HBrO2**2.0
    yp[1] = Br*BrO3*k1 + 2.0*Br*HBrO2*k2 + k4*HBrO2**2.0
    yp[2] = 0.5*Ce*k5 - Br*BrO3*k1 - Br*HBrO2*k2
    yp[3] = Br*BrO3*k1 + BrO3*HBrO2*k3 - Br*HBrO2*k2 - 2.0*k4*HBrO2**2.0
    yp[4] = -Ce*k5 + 2.0*BrO3*HBrO2*k3
    return (yp)
```

Note that this code is not directly useable because the values of parameters are not specified. Also, this is not the same code that is returned by the solver module, as that code has the rate constants fully instantiated, unless the *inputfile* following the **-in** option is a model file with file extension **.model** that has all of the rate constants fully defined in the **$RATES** block.

-format LATEX will return a LaTeX encoding of the equations within a minimal wrapper that makes it compatible with both **latex** and **pdflatex**. This option uses the **sympy latex** function and may not produce the expected output. For example, the above file will produce

```
        \documentclass[12pt,letterpaper]{article}
        \usepackage[latin1]{inputenc}
        \usepackage{amsmath, amsfonts, amssymb}
        \author{py[xlr8r]}
        \date{\today}
        \title{Automatically Generated Equations}
        \begin{document}
        \maketitle
        \begin{align*}
        BrO3' &=  Br BrO_{3} k_{1} - BrO_{3} HBrO_{2} k_{3} +
                HBrO_{2}^{2.0} k_{4\\
        HOBr' &= r BrO_{3} k_{1} + 2.0 Br HBrO_{2} k_{2} +
                HBrO_{2}^{2.0} k_{4\\
        Br' &=  Br BrO_{3} k_{1} - Br HBrO_{2} k_{2} + 0.5 Ce k_{5\\
        HBrO2' &= r BrO_{3} k_{1} - Br HBrO_{2} k_{2} +
                BrO_{3} HBrO_{2} k_{3} - 2.0 HBrO_{2}^{2.0} k_{4\\
        Ce' &= .0 BrO_{3} HBrO_{2} k_{3} - Ce k_{5\\
        \end{align*}
        \end{document}
```

-format DICT will return a python dictionary of the form **"variable":ode, "variable":ode,...** where each **ode** is the right-hand side of the symbolic differential equation for the corresponding variable. Here is an example of a dictionary:

```
{'BrO3': -Br*BrO3*k1 - BrO3*HBrO2*k3 + k4*HBrO2**2.0,
 'HOBr': Br*BrO3*k1 + 2.0*Br*HBrO2*k2 + k4*HBrO2**2.0,
 'Br': 0.5*Ce*k5 - Br*BrO3*k1 - Br*HBrO2*k2,
 'HBrO2': Br*BrO3*k1 + BrO3*HBrO2*k3 - Br*HBrO2*k2 - 2.0*k4*HBrO2**2.0,
 'Ce': -Ce*k5 + 2.0*BrO3*HBrO2*k3}
```

-format JACOBIAN will return the Jacobian matrix of the system:

```
$ python pyx.py interpret -in oregonator.dat -format Jacobian
[-Br*k1 - HBrO2*k3, 0,            -BrO3*k1,        -BrO3*k3 + 2*HBrO2*k4,      0]
[            Br*k1, 0, BrO3*k1 + 2*HBrO2*k2,         2*Br*k2 + 2*HBrO2*k4,      0]
[           -Br*k1, 0,  -BrO3*k1 - HBrO2*k2,                       -Br*k2, 0.5*k5]
[ Br*k1 + HBrO2*k3, 0,   BrO3*k1 - HBrO2*k2, BrO3*k3 - Br*k2 - 4*HBrO2*k4,      0]
[       2*HBrO2*k3, 0,                    0,                    2*BrO3*k3,    -k5]
```

-format STEADYSTATE will return the steady state of the system. Consider, for example, the following simple model **simple.model**:

```
$Reactions
 [2*A -> B, k]
 [B -> Nil, k1]
 [B -> A, k2]
 [Nil -> A, k3]
$IC
 A=1
 B=1
$Rates
 k=1
 k1=1
 k2=.1
 k3 = 1
$
```

This model has a single realizable (physical) steady state and a second non-realizable steady state (with negative concentrations).

```
$ python pycellerator.py interpret -in simple.model -format steadystate
There are 2 steady states
(A, B) = (-0.723746864455746, 0.476190476190476) has eigenvalues
        [2.85841999579395, -1.06343253797097]
(A, B) = (0.723746864455746, 0.476190476190476) has eigenvalues
        [-1.02268894521036, -2.97229851261262]
$
```

The **-symbolic** option may be used in conjunction with **-format STEADYSTATE** to return a tuple **(vars,ss,evs)**, where **vars** is a tuple of the system variables; **ss** is a list of the steady states, each as a tuple; and **evs** is a list of the eigenvalues of the Jacobian at that steady state. This format is probably more meaningful form within the Python Shell:

```
>>> import pycellerator.interpreter as interp
>>> s,ss,evs=interp.interpret("simple.model",
    format="steadystate",symbolic=True)
>>> s
[A, B]
>>> ss
[(-0.723746864455746, 0.476190476190476),
 (0.723746864455746, 0.476190476190476)]
>>> evs
[[2.85841999579395, -1.06343253797097],
 [-1.02268894521036, -2.97229851261262]]
>>>
```

If the **-frozen** option is used it should be followed by one or more unquoted variable names in the model. These variables will be treated as frozen variables, i.e., the right hand sides of their differential equations will be set to zero. Note that these may also be additional variables that are not otherwise specified in the model equations, in which case additional equations will be added to the file.

Command Line in Shell:

```
 >>> import pycellerator
 >>> pycellerator.run ("interpret -in inputfile [ -out filename]
          [ -dump ] [ -format FORMAT][ -frozen variable variable ..]
          [ -symbolic] ")
```

API:

makeODETerms(r, s, *keywords)

input:	`r`: list of reactions as `Reaction` class objects
	`s`: dictionary of sympy symbols as `{"var":symbol,"var":symbol,...}`
return value:	dictionary of ode right-hand-sides by text name of variable, as in `{"var":ode, "var":ode,...}` - the `var`'s in the dictionary are the same as those in `s`.

Optional keyword arguments:

`frozen = []`	list of variables, given by the text equivalents, whose differential equations are held fixed, as in `frozen=["var1","var2",..]`. The entries in the return dictionary will be `.."var1":0,"var2":0,..` for these variables.

`interpret(filename, *keywords)`

input:	name of input file
return value:	text table of differential equations, as per keywords

Optional keyword arguments:

`format = "ODE"`	same as command line `ODE` format other options are `"DICT"`,`"CODE"`,`"PYTHON"`, `"JACOBIAN"`,`"STEADYSTATE"`
`symbolic = False`	set to `True` to get a symbolic Steady state tuple
`dumpparser="False"`	same as command line `Dump`
`ofile=""`	name of output file

Example: This example uses the input file illustrated above.

```
$ python interpreter.py -in oregonator.dat -frozen BrO3 -format code
py[cellerator]: interpreter (2015-09-02 11:53:09)
def f(y,t):
    y[0] = Ce
    y[0] = HOBr
    y[0] = HBrO2
    y[0] = BrO3
    y[0] = Br
    yp[0] = -Ce*k5 + 2.0*BrO3*HBrO2*k3
    yp[1] = Br*BrO3*k1 + 2.0*Br*HBrO2*k2 + k4*HBrO2**2.0
    yp[2] = Br*BrO3*k1 + BrO3*HBrO2*k3 - Br*HBrO2*k2 - 2.0*k4*HBrO2**2.0
    yp[3] = 0
    yp[4] = 0.5*Ce*k5 - Br*BrO3*k1 - Br*HBrO2*k2
    return (yp)
$
```

4.5 solver

The solver will write a stand-alone python script to run a simulation on a model, and then will execute that script.

Command Line Syntax:

```
     $ python pycellerator.py solve -in filename
              [-run  stepsize duration]
              [-plot  variable variable variable ...]]
              [-plotcolumns  n]
              [-sameplot]
              [-out filename} ]
              [-format  FORMAT} ]
              [-pyfile  filename}]
              [-norun]
```

or

```
$ python solver.py -in  filename
         [-run  stepsize duration]
         [-plot  variable variable variable ...]]
         [-plotcolumns  n]
         [-sameplot]
         [-out filename} ]
         [-format  FORMAT} ]
         [-pyfile  filename}]
         [-norun]
```

`-in filename` specifies the name of the model file.

`-run stepsize duration` specifies the step size (for plotting and output files) and duration of the run. The default values are 100 and 1.

`-plot variable variable variable` - if **`-plot`** is used alone, without any variable names, all variables are plotted on a grid. If any variables are listed, then only the requested variables are plotted.

`-plotcolumns n` defines the number of columns to use in the grid of plots (default is 3). Only used in conjunction with the **`-plot`** option.

`-sameplot` - the variables will all be plotted on a single graph rather than a grid. This options supersedes **`-plotcolumns`**.

`-out filename` - optional output file to write the solution to at the requested step size (as given in **`-run`** option).

`-format FORMAT` - format of output file; values of **`FORMAT`** are **`CSV`** (comma-separated values); **`TSV`** (tab-separated values); and **`TABLE`** (space-separated values). The default format is **`CSV`**.

`-norun` - just generate the code, don't run it.

`-mxstep n` - the default number of steps in **`ODEINT`** is 500. This is increased to 500000 by `pycellerator`.

`-pyfile filename` gives the name of the python code to be generated, otherwise a default file name will be used. Existing files will not be overwritten, so if the file already exists, and approximation to the requested file name will be used instead.

Command Line in Shell:

```
>>>import pycellerator
>>> pycellerator.run("interpret -in inputfile
                      [-run stepsize duration]
                      [-plot [ variable variable variable ...]
                      [-plotcolumns n]
                      [-sameplot]
                      [-out filename ]
                      [-format FORMAT ]
                      [-pyfile filename ]
                      [-norun]")
```

4.6 converter

The converter module is will convert **pycellerator** text reactions to `Cellerator` (`xlr8r`) *Mathematica* reaction notebooks.

The reactions will be specified in their *Mathematica* `FullForm` notation, so that they can be written as ASCII text files. This notation can be read directly or copied and pasted directly into an xlr8r notebook and is fully compatible with the xlr8r `interpret` and `run` commands.

Note that LaTeX output is provided by the interpreter module, and SBML input and output is provided by the SBML module.

Command Line Syntax:

```
$ python pycellerator.py convert  -in filename [-out filename] [-dump]
```

The **`-in filename`** specifies the input file name, which is a list of text pycellerator reactions, identical in format to the reactions sent to the parser, such as:

```
[X  <=> XP,   mod[Z, ZPP], rates[a, d, k, 0, a, d, k]]
[XP <=> XPP,  mod[Z, ZPP], rates[a, d, k, 0, a, d, k]]
[Y  <=> YP,   mod[X, XPP], rates[a, d, k, 0, a, d, k]]
[YP <=> YPP,  mod[X, XPP], rates[a, d, k, 0, a, d, k]]
[Z  <=> ZP,   mod[Y, YPP], rates[a, d, k, 0, a, d, k]]
[ZP <=> ZPP,  mod[Y, YPP], rates[a, d, k, 0, a, d, k]]
```

The input **`filename`** may also be a model file (with file extension **`.model`**. In this case the output will include not only `xlr8r` reactions, but lists of initial conditions and rate constants in formats compatible with the `xlr8r run` function.

The **`-out filename`** specifies the output file name. which is typically a *Mathematica* notebook. If it is not specified a unique file name will be generated. The contents of this file will typically look something like this:

```
model = {
 List[Underoverscript[RightArrowLeftArrow[X, XP], ZPP, Z],
      a, d, k, 0, a, d, k ,0],
 List[Underoverscript[RightArrowLeftArrow[XP, XPP], ZPP, Z],
      a, d, k, 0, a, d, k, 0],
 List[Underoverscript[RightArrowLeftArrow[Y, YP], XPP, X],
      a, d, k, 0, a, d, k, 0],
 List[Underoverscript[RightArrowLeftArrow[YP, YPP], XPP, X],
      a, d, k, 0, a, d, k, 0],
 List[Underoverscript[RightArrowLeftArrow[Z, ZP], YPP, Y],
      a, d, k, 0, a, d, k, 0],
 List[Underoverscript[RightArrowLeftArrow[ZP, ZPP], YPP, Y],
      a, d, k, 0, a, d, k, 0]}
```

The **`-dump`** option indicates that the intermediate `Reaction` database should be dumped to the screen.

Command Line in Shell:

```
>>> import pycellerator
>>> pycellerator.run("convert -in  filename [-out  filename] [-dump]")
```

API:

`convert(*keywords)`

input:	(by keyword only)
return value:	full path name of output file

Optional keyword arguments:

`inputfile=""`	Name of file containing pycellerator reactions to be converted
`outputfile=""`	Requested name of outputfile. If the file already exists the name will be modified to generate a unique file name. If not specified a unique default name will be generated.

4.7 SBML Support

4.7.1 Reading SBML Files

When an SBML model is read, it is converted to a `py[cellerator]` model file. A model file that is generated from an SBML model will typically look somewhat different from a hand-generated model because the reactions and other structures will not normally correspond to the typical biological entities that we think of as reactions.

Rather than running models directly from SBML, the **`SBML read`** command converts an SBML model directly into a pycellerator model file. This arrow file can be used like any other pycellerator model file. SBML entities that are not implemented in pycellerator but are referenced in the SBML file cannot be converted to pycellerator.

Syntax

```
  python pycellerator.py sbml read -in filename [-model filename]
```

If **`-model fjiename`** is specified, the output will be written to the specified file. Otherwise the output will be written to the file **`tmp.model`**. Existing files will not be overwritten, and an approximation to the requested name will be used if the file already exists.

Caution: Unexpected errors could occur if variables (e.g., **`species`**) in the SBML model are the same as predefined functions in python, particularly **`sympy`** functions. For example, model 9 (Huang, 1996, "Ultrasensitivity in the MAPK Cascade") in the Biomodels database (`https://www.ebi.ac.uk/biomodels-main/BIOMD0000000009`) contains the variable **`E1`** which is also the name of an elliptic integral function in **`sympy`**. Attempting to convert this model directly would produce the somewhat obscure looking error

```
TypeError: unsupported operand type(s) for *: 'Symbol' and 'function'
```

To fix this, the user is fore-warned to change the names of all symbols, such as **`E1`**, to non-conflicting symbols.

In the following example, Biomodel 9 species **E1** was converted globally (in a text editor) to a new species **Species_E1** prior to use. The revised SBML file was saved in a new file **BM9.xml**. To convert this file to a pycellerator model file **BM9.model**, use

```
python pycellerator.py SBML READ -in BM9.xml -model BM9.model
```

The following model file was generated from the biomodels file

```
$ASSIGNMENTS
 K_PP_norm=(KPase_PP_K + PP_K)/(K + KPase_PP_K + KPase_P_K +
           PP_K + PP_KK_K + PP_KK_P_K + P_K)
 rel_K_PP_max=1.11105062057732*K_PP_norm
 KK_PP_norm=(KKPase_PP_KK + PP_KK + PP_KK_K + PP_KK_P_K)/(KK + KKPase_PP_KK +
 KKPase_P_KK + PP_KK + PP_KK_K + PP_KK_P_K + P_KK + P_KKK_KK + P_KKK_P_KK)
 KKK_P_norm=(P_KKK + P_KKK_KK + P_KKK_P_KK)/(KKK + P_KKK + P_KKK_KK + P_KKK_P_KK)
 r1a=compartment*(1000.0*KKK*Species_E1 - 150.0*Species_E1_KKK)
 r1b=150.0*Species_E1_KKK*compartment
 r2a=compartment*(1000.0*E2*P_KKK - 150.0*E2_P_KKK)
 r2b=150.0*E2_P_KKK*compartment
 r3a=compartment*(1000.0*KK*P_KKK - 150.0*P_KKK_KK)
 r3b=150.0*P_KKK_KK*compartment
 r4a=compartment*(1000.0*KKPase*P_KK - 150.0*KKPase_P_KK)
 r4b=150.0*KKPase_P_KK*compartment
 r5a=compartment*(1000.0*P_KK*P_KKK - 150.0*P_KKK_P_KK)
 r5b=150.0*P_KKK_P_KK*compartment
 r6a=compartment*(1000.0*KKPase*PP_KK - 150.0*KKPase_PP_KK)
 r6b=150.0*KKPase_PP_KK*compartment
 r7a=compartment*(1000.0*K*PP_KK - 150.0*PP_KK_K)
 r7b=150.0*PP_KK_K*compartment
 r8a=compartment*(1000.0*KPase*P_K - 150.0*KPase_P_K)
 r8b=150.0*KPase_P_K*compartment
 r9a=compartment*(1000.0*PP_KK*P_K - 150.0*PP_KK_P_K)
 r9b=150.0*PP_KK_P_K*compartment
 r10a=compartment*(1000.0*KPase*PP_K - 150.0*KPase_PP_K)
 r10b=150.0*KPase_PP_K*compartment
$REACTIONS
 [Species_E1 -> Nil, using["r1a/compartment"]]
 [Nil -> Species_E1_KKK, using["r1a/compartment"]]
 [KKK -> Nil, using["r1a/compartment"]]
 [Nil -> Species_E1, using["r1b/compartment"]]
 [Species_E1_KKK -> Nil, using["r1b/compartment"]]
 [Nil -> P_KKK, using["r1b/compartment"]]
 [Nil -> E2_P_KKK, using["r2a/compartment"]]
 [P_KKK -> Nil, using["r2a/compartment"]]
 [E2 -> Nil, using["r2a/compartment"]]
 [E2_P_KKK -> Nil, using["r2b/compartment"]]
 [Nil -> KKK, using["r2b/compartment"]]
 [Nil -> E2, using["r2b/compartment"]]
 [KK -> Nil, using["r3a/compartment"]]
 [Nil -> P_KKK_KK, using["r3a/compartment"]]
 [P_KKK -> Nil, using["r3a/compartment"]]
 [P_KKK_KK -> Nil, using["r3b/compartment"]]
 [Nil -> P_KK, using["r3b/compartment"]]
 [Nil -> P_KKK, using["r3b/compartment"]]
 [Nil -> KKPase_P_KK, using["r4a/compartment"]]
 [P_KK -> Nil, using["r4a/compartment"]]
 [KKPase -> Nil, using["r4a/compartment"]]
 [KKPase_P_KK -> Nil, using["r4b/compartment"]]
 [Nil -> KK, using["r4b/compartment"]]
 [Nil -> KKPase, using["r4b/compartment"]]
 [Nil -> P_KKK_P_KK, using["r5a/compartment"]]
 [P_KK -> Nil, using["r5a/compartment"]]
 [P_KKK -> Nil, using["r5a/compartment"]]
 [P_KKK_P_KK -> Nil, using["r5b/compartment"]]
 [Nil -> PP_KK, using["r5b/compartment"]]
 [Nil -> P_KKK, using["r5b/compartment"]]
 [Nil -> KKPase_PP_KK, using["r6a/compartment"]]
 [PP_KK -> Nil, using["r6a/compartment"]]
 [KKPase -> Nil, using["r6a/compartment"]]
 [KKPase_PP_KK -> Nil, using["r6b/compartment"]]
 [Nil -> P_KK, using["r6b/compartment"]]
 [Nil -> KKPase, using["r6b/compartment"]]
 [Nil -> PP_KK_K, using["r7a/compartment"]]
 [K -> Nil, using["r7a/compartment"]]
 [PP_KK -> Nil, using["r7a/compartment"]]
 [PP_KK_K -> Nil, using["r7b/compartment"]]
 [Nil -> PP_KK, using["r7b/compartment"]]
 [Nil -> P_K, using["r7b/compartment"]]
 [Nil -> KPase_P_K, using["r8a/compartment"]]
 [P_K -> Nil, using["r8a/compartment"]]
 [KPase -> Nil, using["r8a/compartment"]]
 [KPase_P_K -> Nil, using["r8b/compartment"]]
 [Nil -> K, using["r8b/compartment"]]
 [Nil -> KPase, using["r8b/compartment"]]
 [Nil -> PP_KK_P_K, using["r9a/compartment"]]
 [PP_KK -> Nil, using["r9a/compartment"]]
 [P_K -> Nil, using["r9a/compartment"]]
 [Nil -> PP_K, using["r9b/compartment"]]
 [PP_KK_P_K -> Nil, using["r9b/compartment"]]
 [Nil -> PP_KK, using["r9b/compartment"]]
 [PP_K -> Nil, using["r10a/compartment"]]
 [Nil -> KPase_PP_K, using["r10a/compartment"]]
 [KPase -> Nil, using["r10a/compartment"]]
 [Nil -> KPase, using["r10b/compartment"]]
 [KPase_PP_K -> Nil, using["r10b/compartment"]]
 [Nil -> P_K, using["r10b/compartment"]]
$RATES
 K_PP_norm_max=0.900049
 compartment=4e-12
$IC
 Species_E1=3e-05
 E2=0.0003
 KKK=0.003
 P_KKK=0.0
 KK=1.2
 P_KK=0.0
 PP_KK=0.0
 K=1.2
 P_K=0.0
 PP_K=0.0
 KPase=0.12
 KKPase=0.0003
 Species_E1_KKK=0.0
 E2_P_KKK=0.0
 P_KKK_KK=0.0
```

```
P_KKK_P_KK=0.0
PP_KK_K=0.0
PP_KK_P_K=0.0
KKPase_PP_KK=0.0
KKPase_P_KK=0.0
KPase_PP_K=0.0
KPase_P_K=0.0
K_PP_norm=0.0
KK_PP_norm=0.0
KKK_P_norm=0.0
rel_K_PP_max=0.0
```

This file can support simulations as with any other model file, e.g.

```
python pycellerator.py solve -in BM9.model -plot
```

Supported Features

The following SBML features are not supported by `pycellerator` and will be ignored in SBML models: **`unitDefinitions`**, **`compartmentTypes`**, **`speciesTypes`**, **`initialAssignments`**, **`algebraicRules`**, **`constraints`**, and **`events`**. Support for these features may be added at a later date. Since some of the structures need to be translated, and the way things are evaluated may not correspond to the expected rules of SBML evaluation, as described in the following paragraphs.

1. All variables that have either their **`boundaryCondition`** or **`constant`** flag set to **`True`** are converted to **`Frozen`** variables in the model. This means that they may be set by **`using`** reactions or by **`assignment`** statements. Either of these could potentially conflict with the intent of the SBML model. For example, if the **`constant`** field is set to **`True`**, SBML says that such variables should never change, but if either a rate or assignment rule is present in the SBML model it will be used in the `pycellerator` model (technically this would be an invalid SBML model, so should not lead to problems). However, if both values are set to **`False`** and the variable is set in both a rule and a reaction, then there could be a combination of assignments and ODES for that variable, in conflict with the intent of the SBML (again this would be invalid SBML, so the situation should not arise).

2. If a **`parameter`** is a **`variable`** in an **`assignmentRule`** and has a **`value`** in a **`parameter`** statement, then the **`value`** in the **`parameter`** statement will take precedence and the **`assignmentRule`** will be ignored.

3. All **`rateRules`** are converted to **`"using"`** reactions of the form **`Nil -> Variable`** and the **`variable`** is added to the list of **`frozen`** variables, so that it cannot be changed by any other reactions.

4. All reaction **`kineticLaws`** are assigned to `assignment` statements with the same variable as the reaction **`id`**.

5. Each **`reaction`** is converted to a collection of **`using`** creation and annihilation reactions, with one reaction for each species in the reaction. The reaction will always have the form of **`X -> Nil`** or **`Nil -> X`**, depending on the net stoichiometry of the species in the reaction. Thus, for example, if the SBML file has a reaction

   ```
   A + 3B -> 2A + B + C
   ```

   with a kinetic law **`K`**, the following `pycellerator` reactions will be generated:

```
[Nil -> A, using["K/VA"]]
[B -> Nil, using["2*K/VB"]]
[Nil -> C, using["K/VC"]]
```

where **VA**, **VB**, and **VC** are the volumes of the compartments that species **A**, **B** and **C** reside in, respectively. All of the stoichiometries inside the **using** expression are expressed as positive quantities because the sign will be calculated automatically by the interpreter module, because the reaction `[X -> Nil, using["K"]]` will produce the same ODE term as **[Nil -> X, using["-K"]]**, namely, $[X]' = -K$.

6. At each integration step, all **assignmentRules** (with their parameter values instantiated) are evaluated before the differential equations are updated. Thus a differential equation may refer to any variable defined by an **assignmentRule**

7. All functions defined in the **listOfFunctions** are instantiated globally so that any rule or reaction may refer to them.

4.7.2 Writing SBML Files

Not all features of SBML are supported. Only reactions, initial conditions, and parameters are written to the SBML file. Functions and assignment rules are not written to SBML in the present version.

Syntax

```
python pycellerator.py SBML write -in file [-out filename]
```

SBML files are very lengthy so only a very simple example will be given here. Consider the model file **catalytic.model**

```
$Reactions
 [X => Y, mod[E], rates[k1,k2,k3]]
 [Y -> Nil, k]
$IC
 X = 1.0
 Y = 0.0
 E = 1.0
$Rates
 k1 = 1.0
 k2 = 1.0
 k3 = 1.0
 k = .001
```

To convert this to SBML, the syntax is

```
python pycellerator.py SBML WRITE -in catalytic.model
```

Here is the output.

```
<?xml version="1.0" encoding="UTF-8"?>
<sbml xmlns="http://www.sbml.org/sbml/level3/version1/core" level="3"
                version="1">
  <model substanceUnits="item" timeUnits="dimensionless" extentUnits="item">
    <listOfCompartments>
      <compartment id="compartment" spatialDimensions="3" size="1"
```

```
         units="dimensionless" constant="true"/>
    </listOfCompartments>
    <listOfSpecies>
      <species id="Y" compartment="compartment" initialAmount="0"
        hasOnlySubstanceUnits="true" boundaryCondition="false"
        constant="false"/>
      <species id="X" compartment="compartment" initialAmount="1"
        hasOnlySubstanceUnits="true" boundaryCondition="false"
        constant="false"/>
      <species id="E" compartment="compartment" initialAmount="1"
        hasOnlySubstanceUnits="true" boundaryCondition="false"
        constant="false"/>
      <species id="Nil" compartment="compartment" initialAmount="0"
        hasOnlySubstanceUnits="true" boundaryCondition="false"
        constant="false"/>
    </listOfSpecies>
    <listOfParameters>
      <parameter id="k3" value="1" units="dimensionless" constant="true"/>
      <parameter id="k2" value="1" units="dimensionless" constant="true"/>
      <parameter id="k1" value="1" units="dimensionless" constant="true"/>
      <parameter id="k" value="0.001" units="dimensionless"
      constant="true"/>
    </listOfParameters>
    <listOfReactions>
      <reaction id="r1" reversible="false" fast="false">
        <listOfReactants>
          <speciesReference species="X" stoichiometry="1" constant="false"/>
          <speciesReference species="E" stoichiometry="1" constant="false"/>
        </listOfReactants>
        <listOfProducts>
          <speciesReference species="X_E" stoichiometry="1" constant="false"/>
        </listOfProducts>
        <kineticLaw>
          <math xmlns="http://www.w3.org/1998/Math/MathML">
            <apply>
              <times/>
              <ci> E </ci>
              <ci> X </ci>
              <ci> k1 </ci>
            </apply>
          </math>
        </kineticLaw>
      </reaction>
      <reaction id="r2" reversible="false" fast="false">
        <listOfReactants>
          <speciesReference species="X_E" stoichiometry="1" constant="false"/>
        </listOfReactants>
        <listOfProducts>
          <speciesReference species="X" stoichiometry="1" constant="false"/>
          <speciesReference species="E" stoichiometry="1" constant="false"/>
        </listOfProducts>
        <kineticLaw>
          <math xmlns="http://www.w3.org/1998/Math/MathML">
            <apply>
              <times/>
              <ci> X_E </ci>
              <ci> k2 </ci>
            </apply>
          </math>
        </kineticLaw>
      </reaction>
      <reaction id="r3" reversible="false" fast="false">
        <listOfReactants>
          <speciesReference species="X_E" stoichiometry="1" constant="false"/>
        </listOfReactants>
        <listOfProducts>
          <speciesReference species="Y" stoichiometry="1" constant="false"/>
          <speciesReference species="E" stoichiometry="1" constant="false"/>
        </listOfProducts>
        <kineticLaw>
          <math xmlns="http://www.w3.org/1998/Math/MathML">
            <apply>
              <times/>
              <ci> X_E </ci>
              <ci> k3 </ci>
            </apply>
          </math>
        </kineticLaw>
      </reaction>
      <reaction id="r4" reversible="false" fast="false">
        <listOfReactants>
          <speciesReference species="Y" stoichiometry="1" constant="false"/>
        </listOfReactants>
        <listOfProducts>
          <speciesReference species="Nil" stoichiometry="1" constant="false"/>
        </listOfProducts>
        <kineticLaw>
          <math xmlns="http://www.w3.org/1998/Math/MathML">
            <apply>
              <times/>
              <ci> Y </ci>
              <ci> k </ci>
            </apply>
          </math>
        </kineticLaw>
      </reaction>
    </listOfReactions>
  </model>
</sbml>
```

Chapter 5

Examples

This chapter present some models that are illustrative of pycellerator. The following matrix maps some of the features that are used against the various models.

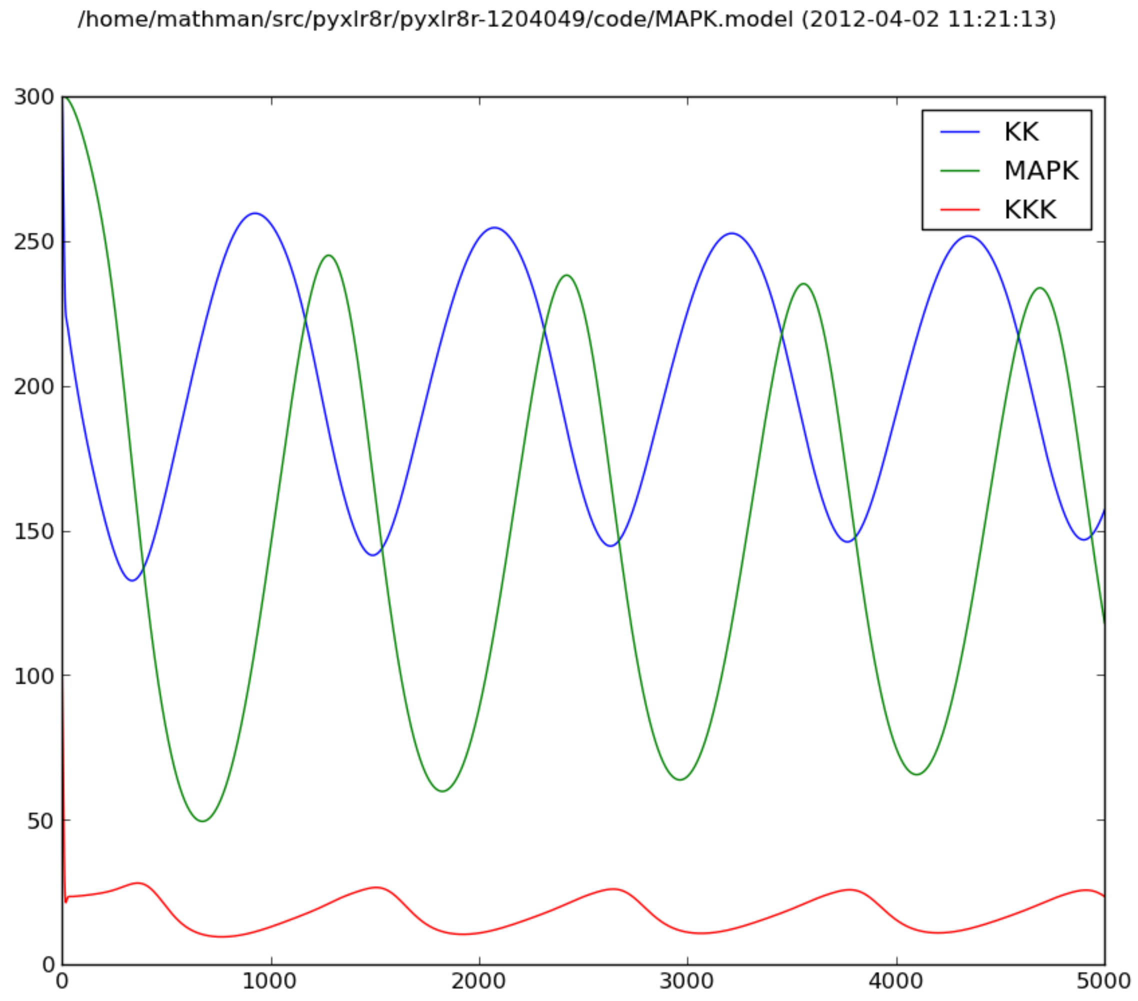
Model	Mass Action (->)	Mass Action (-->)	Mass Action (<->)	Mass Action (=>)	Mass Action (<=>)	Mass Action (:=>)	MMH (:->, :-->)	GRN (|->, |-->)	Hill (|->, |-->)	NHCA (|->, |-->)	SSystem (|->)	rational (==>)	MWC (==>)	Equations / USER	Cascades
E coli Growth											✓				
Lineage Determination	✓											✓		✓	
MAPK Oscillations		✓		✓											✓
Mitotic Oscillator	✓								✓					✓	
NF-$\kappa\beta$	✓		✓						✓						
Oregonator	✓														
Repressilator	✓		✓						✓			✓		✓	
Ring Oscillator (MA)					✓										
Ring Oscillator (MMH)							✓								
Ring Oscillator (GRN)	✓							✓							
Ring Oscillator (NHCA)	✓									✓					

5.1 MAP-Kinase with Oscillations

The following model (unpublished) demonstrates MAPK oscillations and illustrates the use of both catalyzed reactions and enzymatic cascacdes.

```
$Reactions
#
# kinases
#
 [Nil<->S, rates[a0, d0]]
 [KKK => KKKp, mod[S], rates[a1,d1,k1]]
 [KK => KKp =>KKpp, mod[KKKp,KKKp], rates[a3,d3,k3]]
 [MAPK => Kp => Kpp, mod[KKpp], rates[a3,d3,k3]]
#
# competitive inhibition
#
 [KKK_S + Kpp <-> KKK_S_Kpp, rates[a7, d7]]
#
# phosphatases
#
 [KKKp => KKK, mod[KKKph], rates[a4,d4,k4]]
 [KKpp => KKp => KK, mod[KKph], rates[a5,d5,k5]]
 [Kpp => Kp => MAPK, mod[Kph], rates[a6,d6,k6]]
$IC
  KKK =  100
  KKKp =  0
  KK =  300
  KKp =  0
  KKpp =  0
  MAPK =  300
  Kp =  0
  Kpp =  0
  S =  1
  Kph =  1
  KKph =  1
  KKKph =  10
$Rates
 a0 =  1
 d0 =  1
 a1 = 1
 d1 = 7.5
 k1 = 2.5
 a3 =  1
 d3 =  10
 k3 = 0.025
 a4 =  1
 d4 = 1
 k4 = 1
 a5 =  1
 d5 =  1
 k5 =  1
 a6 =  1
 d6 =   1
 k6 =  1
 a7 =  1
 d7 =  1
$
```

```
$ python pycelerator.py solve -in MAPK.model -plot KK KKK MAPK
    -sameplot -run 5000 2
```



5.2 Repressilator

This model demonstrates the use of `Hill` and `rational` arrows. An alternative version that uses `USER` reactions is given at the end of this section. A simulation model for the repressilator [4] is given by[1]

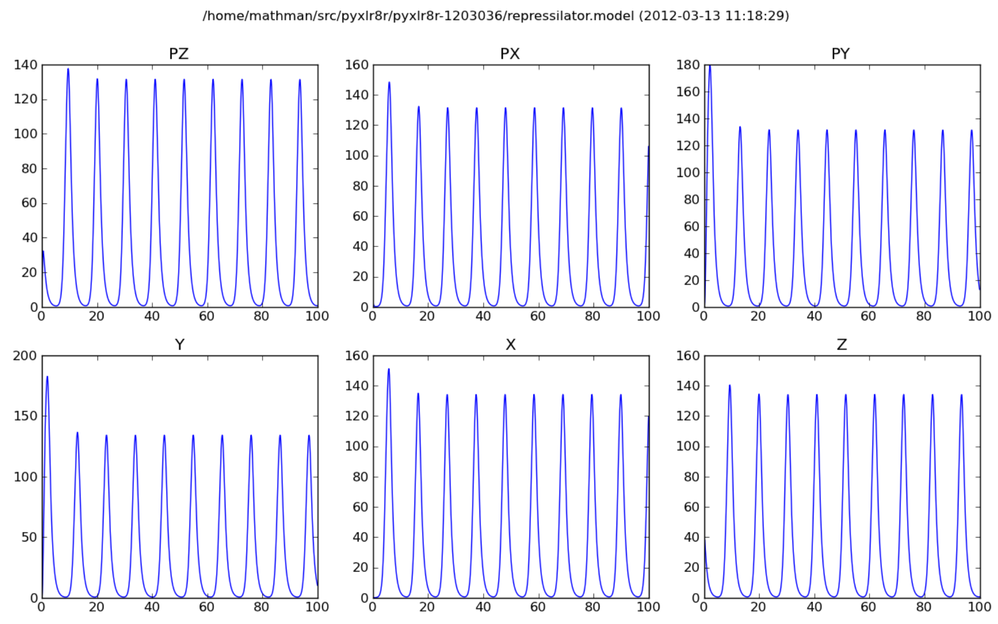
```
$reactions
 [PX -> Nil, Beta]
 [PY -> Nil, Beta]
 [PZ -> Nil, Beta]
 [X -> X + PX, Beta]
 [Y -> Y + PY, Beta]
 [Z -> Z + PZ, Beta]
 [PZ |-> X, Hill[alpha1, n, K, 0, 1]]
 [PX |-> Y, Hill[alpha1, n, K, 0, 1]]
 [PY |-> Z, Hill[alpha1, n, K, 0, 1]]
 [X <-> Nil, rates[k1, alpha0]]
 [Y <-> Nil, rates[k1, alpha0]]
 [Z <-> Nil, rates[k1, alpha0]]
 [[[Nil],[PY]] ==> Z, rational[[alpha], [K, 1],[1], [n,n]]]
 [[[Nil],[PZ]] ==> X, rational[[alpha], [K, 1],[1], [n,n]]]
 [[[Nil],[PX]] ==> Y, rational[[alpha], [K, 1],[1], [n,n]]]
$rates
 alpha = 250
 alpha0 = 0
 alpha1 = 0
 Beta = 5
 n = 2.1
 k1 = 1
 K = 1
$ic
 PX = 5
 PZ = 15
```

If we call this file **`repressilator.model`** then the command[2]

```
$ python pycellerator solve -in repressilator.model -plot -run 100 .1
```

will produce a plot such as this:

[1]The terminating dollar sign at the end of the models is optional.
[2]The leading dollar sign in the command refers to the bash prompt and should not be manually entered.



as well as a CSV file that looks something like this:

```
0.0,15.0,5.0,0.0,0.0,0.0,0.0
0.1,14.2278260331,3.05384634588,0.247394702031,1.34912572721,0.0990958803047,23.562567514
0.2,21.504406326,1.90401628677,1.28225870666,4.57557848574,0.149799701424,37.9011356108
0.3,28.2774224855,1.21721312786,3.86093341284,11.2385014195,0.16314039654,38.2545583466
0.4,31.5798946122,0.803272174073,8.97682635583,22.2830086972,0.166187145597,35.2157392832
0.5,32.317811614,0.552727585611,17.2339097555,36.8646688505,0.166663798479,31.9877430887
...
99.8 0.648217893072,98.8724564077,14.3446430729,11.4817962739,113.312240676,0.649415887468
99.9,0.655884845799,105.886314242,12.9821534961,10.3905866114,119.451386878,0.685929274901
100.0,0.678921934741,112.385765424,11.7488271934,9.4030395272,124.765357623,0.74177680618
```

To see the differential equations produced by this model one could isolate the reactions in a file such as

```
[PX -> Nil, Beta]
[PY -> Nil, Beta]
[PZ -> Nil, Beta]
[X -> X + PX, Beta]
[Y -> Y + PY, Beta]
[Z -> Z + PZ, Beta]
[PZ |-> X, Hill[alpha1, n, K, 0, 1]]
[PX |-> Y, Hill[alpha1, n, K, 0, 1]]
[PY |-> Z, Hill[alpha1, n, K, 0, 1]]
[X <-> Nil, rates[k1, alpha0]]
[Y <-> Nil, rates[k1, alpha0]]
[Z <-> Nil, rates[k1, alpha0]]
[[[Nil],[PY]] ==> Z, rational[[alpha], [K, 1],[1], [n,n]]]
[[[Nil],[PZ]] ==> X, rational[[alpha], [K, 1],[1], [n,n]]]
[[[Nil],[PX]] ==> Y, rational[[alpha], [K, 1],[1], [n,n]]]
```

If we call this file **repressilator.dat** then the command

```
python interpreter.py -in repressilator.dat
```

will produce the output

```
PZ′ = Beta*Z − Beta*PZ
PX′ = Beta*X − Beta*PX
PY′ = Beta*Y − Beta*PY
Y′ = alpha0 − Y*k1 + alpha/(K**n + PX**n) + alpha1*PX**n/(K**n + PX**n)
X′ = alpha0 − X*k1 + alpha/(K**n + PZ**n) + alpha1*PZ**n/(K**n + PZ**n)
Z′ = alpha0 − Z*k1 + alpha/(K**n + PY**n) + alpha1*PY**n/(K**n + PY**n)
```

Within the ipython interface. the simpler command **`cellerator.printODES()`** would be sufficient to do this after loading model.

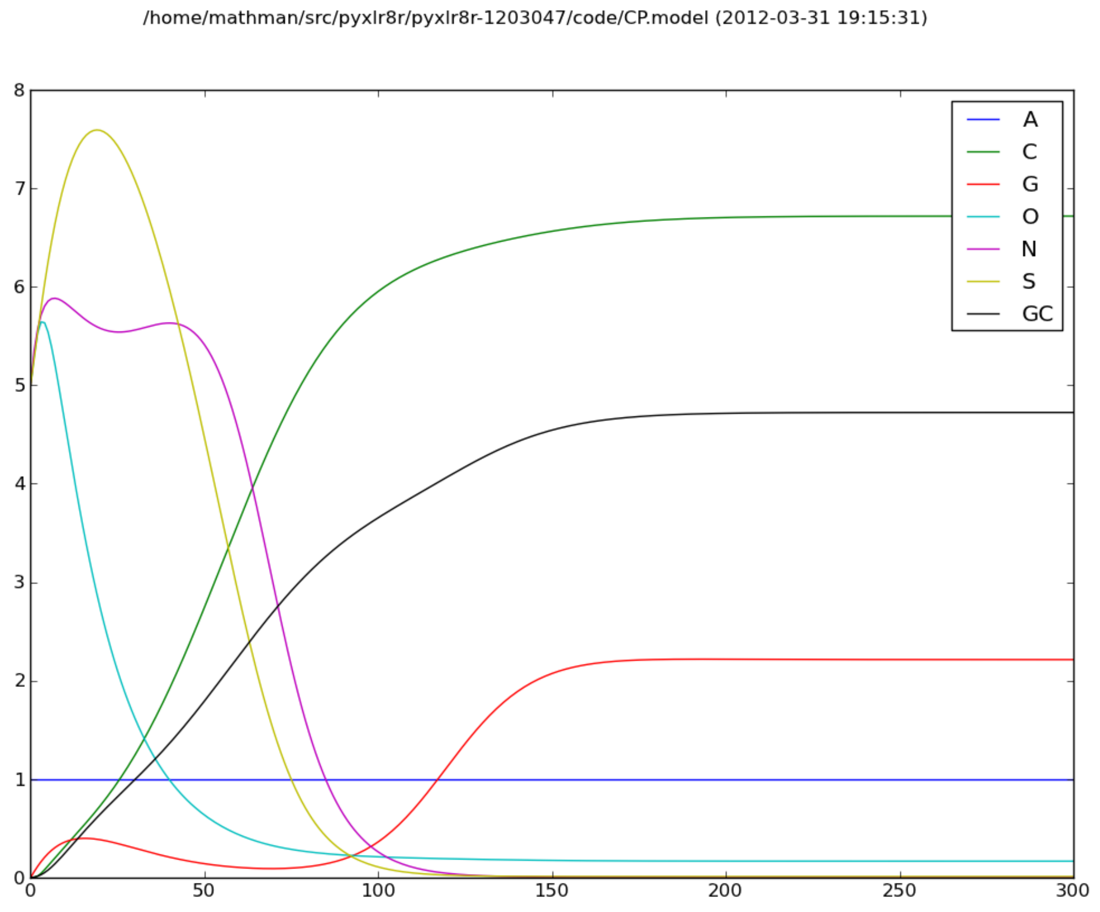
The repressilator can also be implementend using **`rational`** and **`USER`** reactions:

```
[[[Nil],[PY]] ==> Z, rational[[alpha], [K, 1],[1], [n,n]]]
[[[Nil],[PZ]] ==> X, rational[[alpha], [K, 1],[1], [n,n]]]
[[[Nil],[PX]] ==> Y, rational[[alpha], [K, 1],[1], [n,n]]]
[PY |-> Z, USER[alpha, -1, n, 0, "lambda x: 1/(K**n+x)"]]
[PZ |-> X, USER[alpha, -1, n, 0, "lambda x: 1/(K**n+x)"]]
[PX |-> Y, USER[alpha, -1, n, 0, "lambda x: 1/(K**n+x)"]]
```

5.3 Lineage Determination

This model demonstrates the use of the rational with product notation. The model is based on [2].

```
$REACTIONS
 [[[A,O*S,O*S*N],[A, O,O*S,O*S*N,C*O,GC]]==>O, rational[[a0,a1,a2,a3],
     [1,b0,b1,b2,b3,b4,b5], [],[]]]
 [O->Nil,gamma1]
#
 [[[O*S, O*S*N], [O, O*S, O*S*N]] ==>S, rational[[c0,c1,c2],
     [1,d0,d1,d2],[],[]]]
 [S->Nil,gamma2]
#
 [[[O*S,O*S*N],[O,O*S,O*S*N,O*G]] ==>N, rational[[e0,e1,e2],
    [1,f0,f1,f2,f3],[],[]]]
 [N->Nil,gamma3]
#
 [[[C],[C,C*O]]==>C, rational[[g0,g1],[1,h0,h1],[],[]]]
 [C->Nil,gamma4]
#
 [[[C,G],[C,G]]==>GC, rational[[i0,i1,i2],[1,j0,j1],[],[]]]
 [GC->Nil,gamma5]
#
 [[[O,G],[O,G,N]]==>G, rational[[p0,p1,p2],[1,q0,q1,q2],[],[]]]
 [G->Nil, gammag]
#
 [A->A,1]
$IC
 O=5
 N=5
 S=5
 G=0
 GC=0
 C=0
 A=1
$Rates
a0 = 0.001
 a1 = 1
 a2 = 0.005
 a3 = 0.025
 b0 = 1
 b1 = 0.001
 b2 = 0.005
 b3 = 0.025
 b4 = 10
 b5 = 10
 gamma1 = 0.1
 c0 = 0.001
 c1 = 0.005
 c2 = 0.025
 d0 = 0.001
 d1 = 0.005
 d2 = 0.025
 gamma2 = 0.1
 e0 = 0.001
 e1 = 0.1
 e2 = 0.1
 f0 = 0.001
 f1 = 0.1
 f2 = 0.1
 f3 = 10
 gamma3 = 0.1
 g0 = 0.001
 g1 = 2
 h0 = 2
 h1 = 5
 gamma4 = 0.1
 gamma5 = 0.1
 i0 = 0.001
 i1 = 0.1
 i2 = 0.1
 j0 = 0.1
 j1 = 0.1
 p0 = 0.1
 p1 = 1.
 p2 = 0.00025
 q0 = 1.
 q1 = 0.00025
 q2 = 15
 gammag = 0.1
$
```



5.4 NF-$\kappa\beta$ Signaling Model

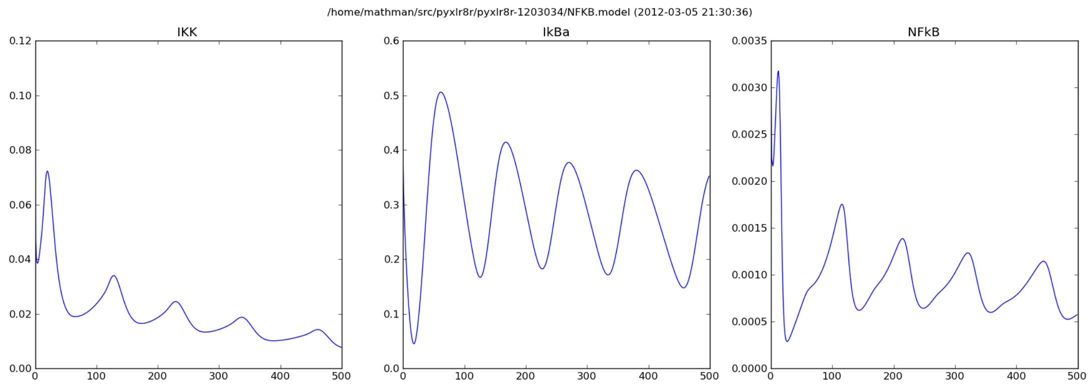
A pycellerator model of the the NF-$\kappa\beta$ signaling model of [8] is given by:

```
$Reactions
 [IkBa + NFkB <-> IkBa_NFkB, rates[a1, d1]]
 [IKK_IkBa + NFkB <-> IKK_IkBa_NFkB, rates[a2, d2]]
 [NFkB <-> NFkBn, rates[tr1, tr2]]
 [IkBa_NFkB -> NFkB, deg1]
 [IkBan + NFkBn <-> IkBan_NFkBn, rates[a1, d1]]
 [IkBa + IKK <-> IKK_IkBa, rates[a3, d3]]
 [IkBat -> Nil, deg3]
 [IkBa <-> IkBan, rates[tr3, tr4]]
 [IKK_IkBa_NFkB -> IKK + NFkB, k1]
 [IKK_IkBa -> IKK, k2]
 [IkBat |-> IkBa, Hill[v1, n1, K1]]
 [IkBa_NFkB <-> IkBan_NFkBn, rates[tr5, tr6]]
 [IkBa -> Nil, deg2]
 [NFkBn |-> IkBat, Hill[v2, n2, K2]]
 [Nil -> IkBat, trb]
 [IkBa_NFkB + IKK <-> IKK_IkBa_NFkB, rates[a4, d4]]
 [IKK -> Nil, adapt]
$IC
 NFkB = 0.003053
 IkBa = 0.372
 IkBa_NFkB = 0.09826
 NFkBn = .00047638
 IkBan = 0.11937
 IkBan_NFkBn = 0.001985
 IKK = 0.1
 IkBat = 0.008454217
 IKK_IkBa_NFkB = 0
 IKK_IkBa = 0
$Rates
 v1 = 2.448
 K1 = 10
 n1 = 1
 v2 = 1.02713
 K2 = 1
 n2 = 2
 a1 =  30
 a2= 30
 a3= 1.35
 a4= 11.1
 tr1= 5.4
 tr2= 0.0048
 tr3= 0.018
 tr4= 0.012
 tr5= 0
 tr6= 0.82944
 d1= 0.03
 d2= 0.03
 d3= 0.0075
 d4= 0.105
 deg1= 0.00135
 deg2= 0.00675
 deg3= 0.0168
 k1= 1.221
 k2= 0.2442
 trb=0.0000921375
 adapt= 0.0072
$
```

Running a simulation with this file as `NFKB.model` using the command

```
python pycellerator solve -in NFKB.model -plot IKK IkBa NFkB -run 500 .1
```

will produce the following plot:



The differential equations can be obtained from

```
python pycellerator.py  interpret -in NFKB.model
```

which produces the following output,

```
IkBa_NFkB' = IKK_IkBa_NFkB*d4 + IkBan_NFkBn*tr6 - IkBa_NFkB*d1 -
  IkBa_NFkB*deg1 - IkBa_NFkB*tr5 + IkBa*NFkB*a1 - IKK*IkBa_NFkB*a4
IkBan_NFkBn' = IkBa_NFkB*tr5 - IkBan_NFkBn*d1 - IkBan_NFkBn*tr6 +
  IkBan*NFkBn*a1
IKK_IkBa' = IKK_IkBa_NFkB*d2 - IKK_IkBa*d3 - IKK_IkBa*k2 +
  IKK*IkBa*a3 - IKK_IkBa*NFkB*a2
NFkBn' = IkBan_NFkBn*d1 + NFkB*tr1 - NFkBn*tr2 - IkBan*NFkBn*a1
IKK_IkBa_NFkB' = -IKK_IkBa_NFkB*d2 - IKK_IkBa_NFkB*d4 -
 IKK_IkBa_NFkB*k1 + IKK*IkBa_NFkB*a4 +
 IKK_IkBa*NFkB*a2
NFkB' = IKK_IkBa_NFkB*d2 + IKK_IkBa_NFkB*k1 + IkBa_NFkB*d1 +
 IkBa_NFkB*deg1 + NFkBn*tr2 - NFkB*tr1 -
 IKK_IkBa*NFkB*a2 - IkBa*NFkB*a1
IkBan' = IkBa*tr3 + IkBan_NFkBn*d1 - IkBan*tr4 - IkBan*NFkBn*a1
IkBa' = IKK_IkBa*d3 + IkBa_NFkB*d1 + IkBan*tr4 - IkBa*deg2 -
 IkBa*tr3 - IKK*IkBa*a3 - IkBa*NFkB*a1 +
 v1*IkBat**n1/(IkBat**n1 + K1**n1)
IkBat' = trb - IkBat*deg3 + v2*NFkBn**n2/(K2**n2 + NFkBn**n2)
IKK' = IKK_IkBa*d3 + IKK_IkBa*k2 + IKK_IkBa_NFkB*d4 +
 IKK_IkBa_NFkB*k1 - IKK*adapt - IKK*IkBa*a3 -
 IKK*IkBa_NFkB*a4
```

If the `code` option is requested, e.g.,

```
  python pycellerator.py interpret -in NFKB.dat -format CODE -out nfkb.py
```

The the file **`nfkb.py`** will look something like this:[3]

```
def f(y,t):
    y[0] = IkBa_NFkB
    y[1] = IkBan_NFkBn
    y[2] = IKK_IkBa
    y[3] = NFkBn
    y[4] = IKK_IkBa_NFkB
    y[5] = NFkB
    y[6] = IkBan
    y[7] = IkBa
    y[8] = IkBat
    y[9] = IKK
    yp[0] = IKK_IkBa_NFkB*d4 + IkBan_NFkBn*tr6 - IkBa_NFkB*d1 -
                IkBa_NFkB*deg1 - IkBa_NFkB*tr5 + IkBa*NFkB*a1 -
                IKK*IkBa_NFkB*a4
    yp[1] = IkBa_NFkB*tr5 - IkBan_NFkBn*d1 - IkBan_NFkBn*tr6 +
            IkBan*NFkBn*a1
    yp[2] = IKK_IkBa_NFkB*d2 - IKK_IkBa*d3 - IKK_IkBa*k2 +
            IKK*IkBa*a3
        - IKK_IkBa*NFkB*a2
    yp[3] = IkBan_NFkBn*d1 + NFkB*tr1 - NFkBn*tr2 -
            IkBan*NFkBn*a1
    yp[4] = -IKK_IkBa_NFkB*d2 - IKK_IkBa_NFkB*d4 - IKK_IkBa_NFkB*k1 +
            IKK*IkBa_NFkB*a4 + IKK_IkBa*NFkB*a2
    yp[5] = IKK_IkBa_NFkB*d2 + IKK_IkBa_NFkB*k1 + IkBa_NFkB*d1 +
            IkBa_NFkB*deg1 + NFkBn*tr2
                - NFkB*tr1 - IKK_IkBa*NFkB*a2 - IkBa*NFkB*a1
    yp[6] = IkBa*tr3 + IkBan_NFkBn*d1 - IkBan*tr4 - IkBan*NFkBn*a1
    yp[7] = IKK_IkBa*d3 + IkBa_NFkB*d1 + IkBan*tr4 - IkBa*deg2 -
            IkBa*tr3 - IKK*IkBa*a3
                - IkBa*NFkB*a1 + v1*IkBat**n1/(IkBat**n1 + K1**n1)
    yp[8] = trb - IkBat*deg3 + v2*NFkBn**n2/(K2**n2 + NFkBn**n2)
    yp[9] = IKK_IkBa*d3 + IKK_IkBa*k2 + IKK_IkBa_NFkB*d4 +
            IKK_IkBa_NFkB*k1 - IKK*adapt
                - IKK*IkBa*a3 - IKK*IkBa_NFkB*a4
    return (yp)
```

[3]While each line of code terminates with a newline character, individual lines of code are not line-wrapped and may be very line. The wrapping shown here was edited for illustration in this document, so in fact, it is likely that most lines of code will exceed the standard 80 character page width. This is why the usual python line wrap characters are not shown in the example.

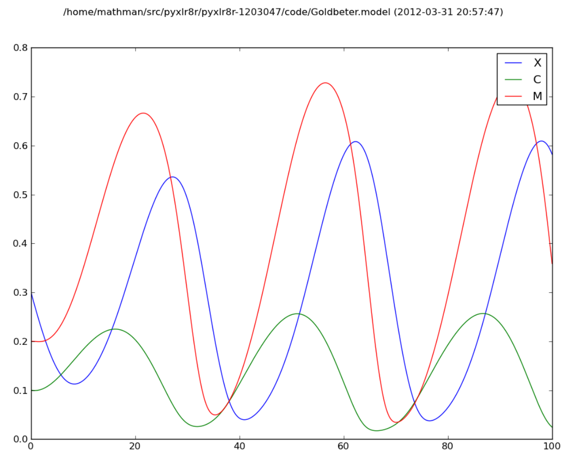
5.5 Minimal Mitotic Model

This is a standard minimal model of a mitotic oscillator[7]. It illustrates the use non-catalytic Hill function arrows as well as the use of equations as part of the rate constants. The model is given by:

```
# Goldbeter, A. A minimal cascade model for the mitotic
# oscillator involving cyclin and cdc2 kinase. Proc. Natl.
# Acad. Sci. USA 88:9107-1101 (1991).
$REACTIONS
 [C <-> Nil, rates[kd, vi]]
 [C |--> Nil, mod[X], Hill[vd, 1, Kd, 0,1 ]]
 [M |--> Nil, mod[Nil], Hill[v2, 1, K2, 0, 1]]
 [X |--> Nil, mod[Nil], Hill[v4, 1, K4, 0, 1]]
 [Nil -> X, " vm3 * M * (1-X)/(K3+1-X)"]
 [C |-> M, Hill[" vm1*(1-M)/(K1+1-M)", 1, Kc, 0, 1]]
$IC
 C = 0.1
 M = 0.2
 X = 0.3
$RATES
 vd = 0.1
 vi = 0.023
 v2 = 0.167
 v4 = 0.1
 vm1 = 0.5
 vm3 = 0.2
 kd = 0.00333
 K1 = 0.1
 K2 = 0.1
 K3 = 0.1
 K4 = 0.1
 Kc = 0.3
 Kd = 0.02
$
```

The syntax for running the model is

```
 $ python pycellerator.py solve -in Goldbeter.model  -plot -run 100 .1
   -sameplot
```



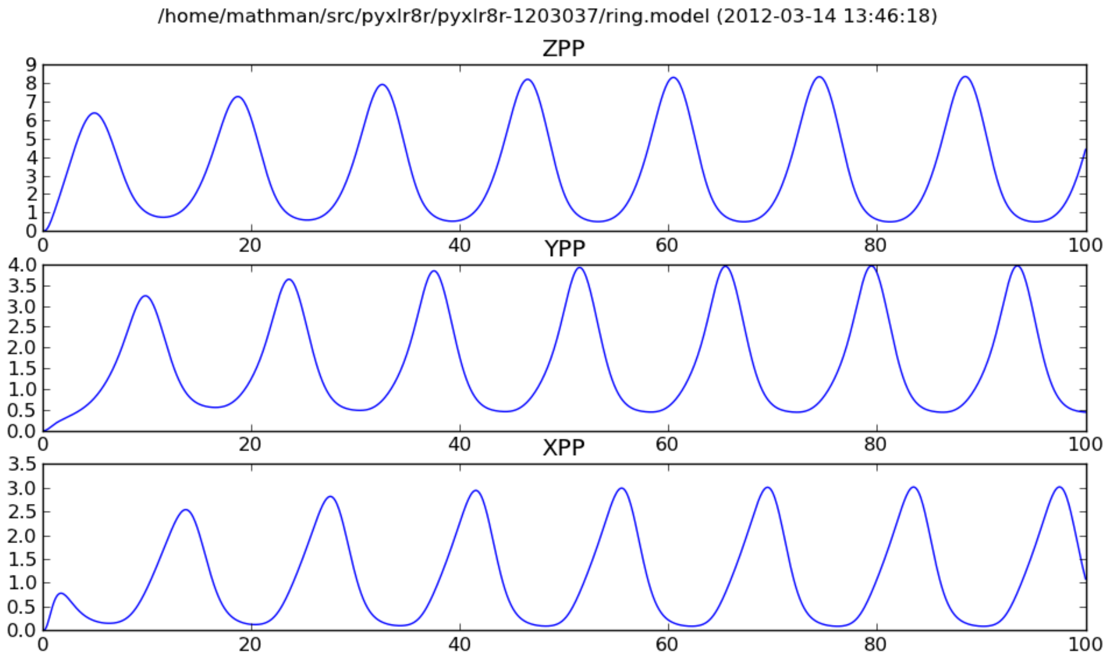
5.6 Ring Oscillator Using Mass Action Equations

The following model describes a ring oscillator using only mass-action equations:

```
$Reactions
 [X  <=> XP,   mod[Z, ZPP], rates[a, d, k, 0,  a, d, k]]
 [XP <=> XPP,  mod[Z, ZPP], rates[a, d, k, 0, a, d, k]]
 [Y  <=> YP,   mod[X, XPP], rates[a, d, k, 0, a, d, k]]
 [YP <=> YPP,  mod[X, XPP], rates[a, d, k, 0, a, d, k]]
 [Z  <=> ZP,   mod[Y, YPP], rates[a, d, k, 0, a, d, k]]
 [ZP <=> ZPP,  mod[Y, YPP], rates[a, d, k, 0, a, d, k]]
$IC
 X = 10.0
 Y = 15.0
 Z = 20.0
$Rates
 a = 1.0
 d = 1.0
 k = 1.0
$
```

Syntax:

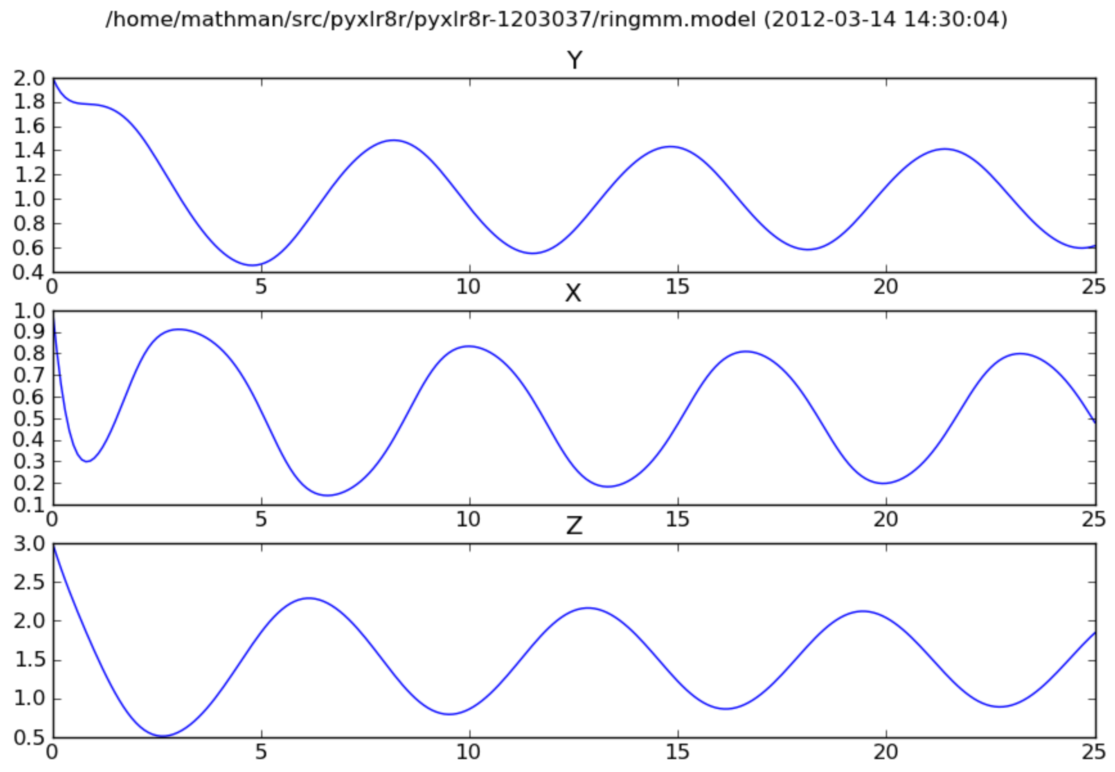
```
python pycellerator.py solver -in ringma.model -plot -run 100 .1
-plotcolumns 1
```



5.7 Ring Oscillator Using Michaelis-Menten-Henri Reactions

```
$Reactions
 [X :--> XP, mod[Z], MMH[K,v]]
 [XP:--> X,  mod[ZP],MMH[K,v]]
 [Y :--> YP, mod[X], MMH[K,v]]
 [YP:--> Y,  mod[XP],MMH[K,v]]
 [Z :--> ZP, mod[Y], MMH[K,v]]
 [ZP:--> Z,  mod[YP],MMH[K,v]]
$IC
 X = 1.0
 Y = 2.0
 Z = 3.0
$Rates
 v=1.0
 K=0.5
$
```

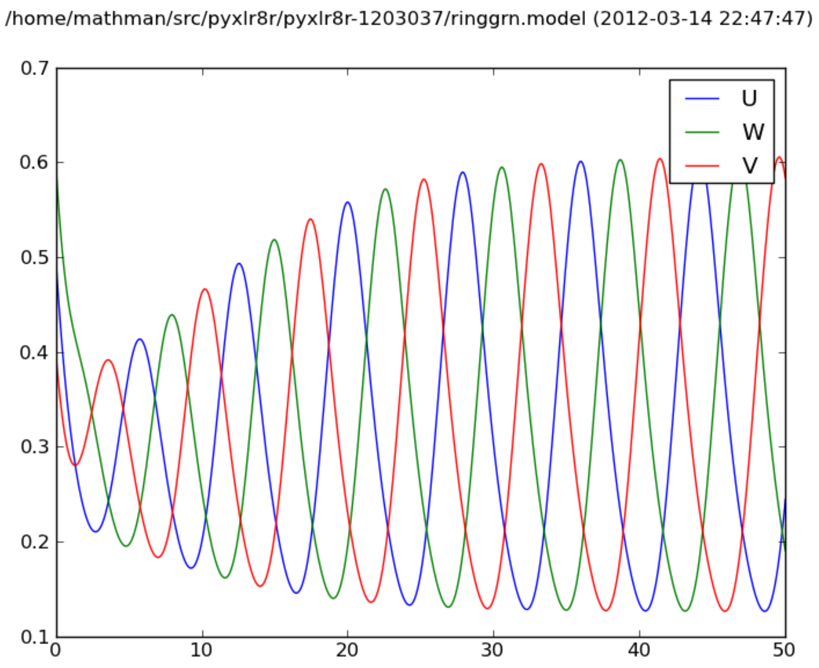
```
  python pycellerator solve -in ringmm.model -plot -run 25 .1 -plotcolumns 1
```



5.8 Ring Oscillator Using GRN Reactions

```
$Reactions
 [U |-> V, GRN[v, b, n, h]]
 [V |-> W, GRN[v, b, n, h]]
 [W |-> U, GRN[v, b, n, h]]
 [V -> Nil, k]
 [U -> Nil, k]
 [W -> Nil, k]
$IC
 U = .5
 V = .4
 W = 0.6
$Rates
 v = 1
 n = 2
 h = 1
 k = 0.5
 b = -10
$
```

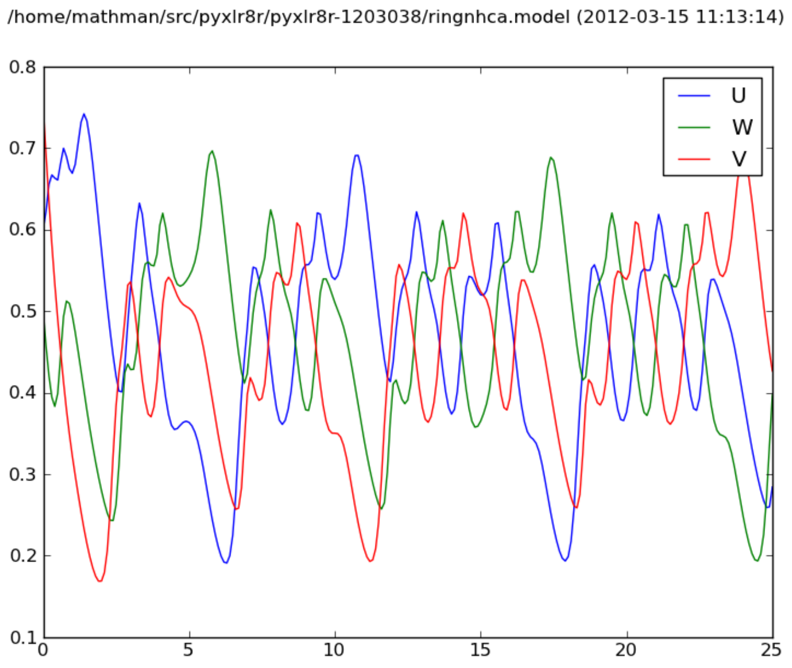
```
python pycelerator solve -in ringgrn.model -plot -run 50 .1 -sameplot
```



5.9 Ring Oscillator Using NHCA Reactions

```
$Reactions
 [U |-> V, NHCA[v,TP,TM,n,m,K]]
 [V |-> W, NHCA[v,TP,TM,n,m,K]]
 [W |-> U, NHCA[v,TP,TM,n,m,K]]
 [V -> Nil, k]
 [U -> Nil, k]
 [W -> Nil, k]
$IC
 U = .6
 V = .75
 W = .5
$Rates
 v = 1
 n = 2
 m = 2
 K = 10
 k = 1
 TP=.2
 TM=-5
$
```

```
 python pycellerator.py solve -in ringnhca.model -plot -run 25 .1 -sameplot
```

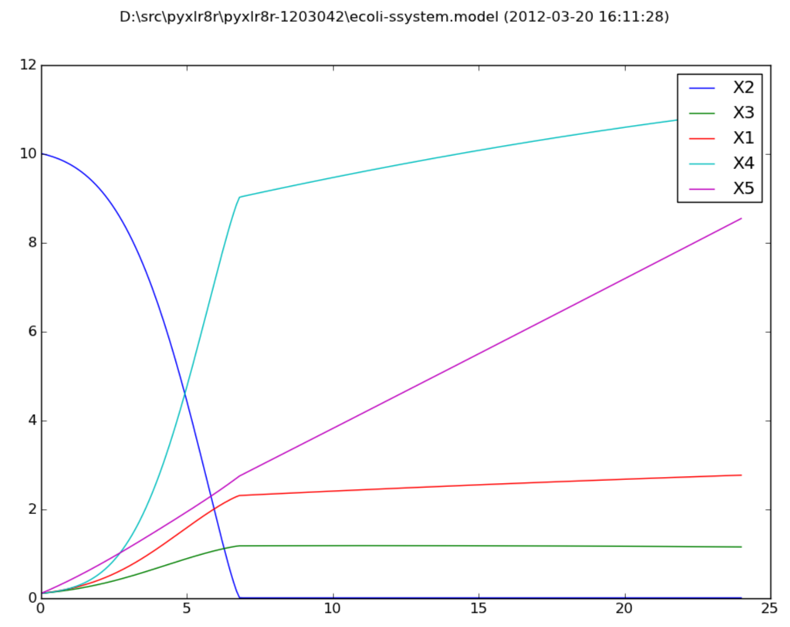


5.10 Recombinant *E coli* growth model

This model uses an S-System approach that is given by [11].

```
$reactions
[[X1,X2] |-> X1,    SSystem[1, a1, b1, [g11,g12], [h11,h12]]]
[[X1,X2] |-> X2,    SSystem[1, a2, b2, [0, 0], [h21, h22]]]
[[X1,X2,X3] |-> X3, SSystem[1, a3, b3, [g31,g32,0], [h31,h32,h33]]]
[[X1,X2,X4] |-> X4, SSystem[1, a4, b4, [g41,g42,0], [h41,h42,h44]]]
[[X1,X2,X5] |-> X5, SSystem[1, a5, b5, [g51,g52,0], [h51,h52,h55]]]
$ic
X1 = 0.1
X2 = 10
X3 = 0.1
X4 = 0.1
X5 = 0.1
$rates
a1 = 0.4973
a2 = 0.0817
a3 = 0.2858
a4 = 3.7124
a5 = 0.4562
b1 = 0.1648
b2 = 1.2484
b3 = 0.1285
b4 = 2.5318
b5 = 0.0335
g11 = 0.9099
g31 = 0.7366
g41 = 1.7076
g51 = 0.2292
g12 = 0.1301
g32 = 0.1311
g42 = 0.1252
g52 = 0.0277
h11 = 1.7514
h21 = 0.9325
h31 = 1.3535
h41 = 1.9875
h51 = 1.1978
h12 = 0.1292
h22 = 0.1927
h32 = 0.1175
h42 = 0.1210
h52 = 0.4462
h33 = -0.0110
h44 = -0.0100
h55 = -0.0426
$
```

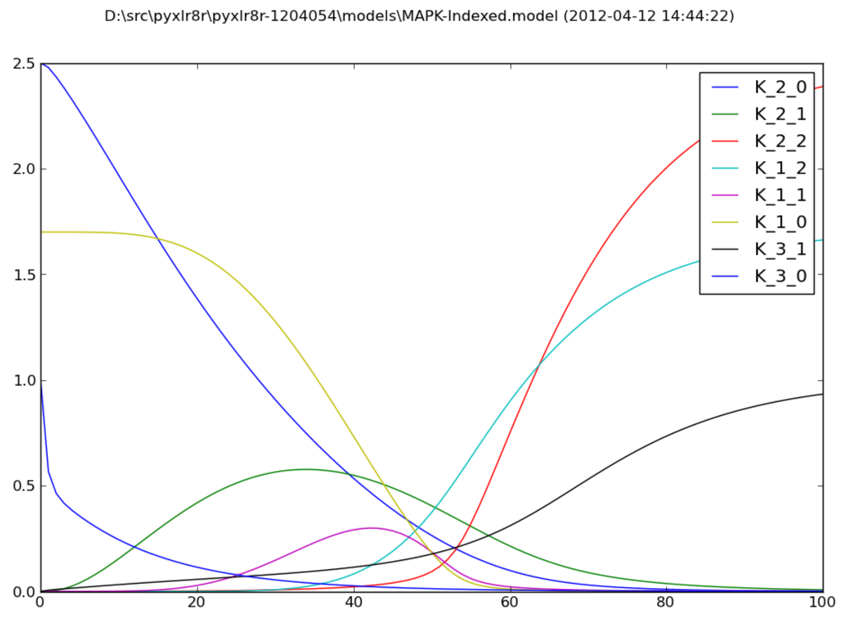
```
python pycellerator.py solve -in ecoli.model -plot -run 24 .1 -sameplot
```


D:\src\pyxlr8r\pyxlr8r-1203042\ecoli-ssystem.model (2012-03-20 16:11:28)
X2
X3
X1
X4
X5
0
2
4
6
8
10
12
0
5
10
15
20
25

5.11 MAPK Cascade with Indexed Stages

```
$Reactions
 [K(3,0) <=> K(3,1), mod[RAFK, RAFph],
    rates[a1,d1,k1,0,a2,d2,k2,0]]
 [K(2,0) <=> K(2,1) <=> K(2,2), mod[ K(3,1), MEKph ],
    rates[ a3, d3, k3, 0, a4, d4, k4, 0, a5, d5, k5, 0, a6, d6, k6, 0]]
 [K(1,0) <=> K(1,1) <=> K(1,2), mod[ K(2,2), MAPKph ],
    rates[ a7, d7, k7, 0, a8, d8, k8, 0, a9, d9, k9, 0, a10, d10, k10, 0]]
$IC
 K(3,0) = 1
 K(2,0) = 2.5
 K(1,0) = 1.7
 RAFK = 1
$Rates
 a1=1.
 a2=0.5
 a3=3.3
 a4=10.
 a5=3.3
 a6=10.
 a7=20.
 a8=5.
 a9=20.
 a10=5.
 d1=.4
 d2=.5
 d3=.42
 d4=.8
 d5=.4
 d6=.8
 d7=.6
 d8=.4
 d9=.6
 d10=.4
 k1=.1
 k2=.1
 k3=.1
 k4=.1
 k5=.1
 k6=.1
 k7=.1
 k8=.1
 k9=.1
 k10=.1
$
```

```
python pycellerator.py solve -in MAPK-Indexed.model -solve -plot K(1,0)
    K(1,1) K(1,2) K(2,0) K(2,1) K(2,2) K(3,0) K(3,1) -sameplot
```


D:\src\pyxlr8r\pyxlr8r-1204054\models\MAPK-Indexed.model (2012-04-12 14:44:22)
2.5
2.0
1.5
1.0
0.5
0.0
0
20
40
60
80
100
K_2_0
K_2_1
K_2_2
K_1_2
K_1_1
K_1_0
K_3_1
K_3_0

Appendix A

File Formats

A.1 Reaction Files

Reaction files are lists of cellerator reactions, with at most one reaction per line. A single reaction may be spread over multiple lines if the last character on the line is the backslash (\) character. Reaction syntax is as described in chapter 3. Blank lines are ignored, and the cross-hatch character (#) indicates the beginning of a comment.[1] Comments are terminated by an end-of-line. An example is given by the following:

```
# Oregonator Data File
# Reference: Field and Noyes, J Chem Phys 60: 1877 (1974)
#
[Br + BrO3 -> HBrO2 + HOBr, k1]
[Br + HBrO2 -> 2*HOBr, k2]
[BrO3 + HBrO2 -> 2*Ce + 2*HBrO2, k3]
[2*HBrO2-> BrO3 + HOBr, k4]
[Ce -> 0.5*Br, k5]
```

A.2 Model Files

A model file contains reactions, initial conditions, rate constants, function definitions (optional) and frozen variables (optional). Each of theses lists (except for the reactions) is optional. Each lists is indicated by an indicator **$Assignments**, **$Functions**, **$Reactions**, **$IC**, **$Rates** or **$Frozen** as illustrated in the following example. The order of the different sections does not matter, i.e., **Rates** may precede **IC**, etc. Comments and blank lines are ignored

The **$Assignments** section should be followed by a list of assignment statements of the form

```
variable = expression
```

where **variable** is any model variable and **expression** is any valid python expression, such as

```
A = B + C - 4D
C = pow(B, 4) +A
```

The **$Functions** section contains a list of functions that may be reference elsewhere in the model. Functions are defined with the syntax

[1]The crosshatch was chosen as the comment delimiter because it will be familiar to Python users.

```
fname(var1,var2,...) = expression
```

where **fname** is the name of the function; **var1,var2,...** are the arguments of the function, and **expression** is any valid python infix expression. For example,

```
 f(m, x)  =  m * (1-x)/(K3+1-x)
 g(m) = (1-m)/(K1+1-m)
```

In the first function definition, the function has two arguments, **m** and **x**, which must be passed to it; the function must be then referenced as, say **f(P,Q)** in a kinetic law, which would represent the expression **P*(1-Q)/(K3+1-Q)**. The reference to **K3** is a reference to a global parameter in the model. The second function has only one variable; a reference **g(U)** in a kinetic law would represent the expression **(1-U)/(K1+1-U)** where **K1** is also a global parameter of the model.

An example of the **$Functions** keyword is illustrated by this alternative version of the Goldbeter mitotic model (compare to the one given in section 5.5):

```
$REACTIONS
 [C <-> Nil, rates[kd, vi]]
 [C |--> Nil, mod[X], Hill[vd, 1, Kd, 0,1 ]]
 [M |--> Nil, mod[Nil], Hill[v2, 1, K2, 0, 1]]
 [X |--> Nil, mod[Nil], Hill[v4, 1, K4, 0, 1]]
 [Nil -> X, "vm3*f(M,X)"]
 [C |-> M, Hill["vm1*g(M)", 1, Kc, 0, 1]]
$IC
 C = 0.1
 M = 0.2
 X = 0.3
$FUNCTIONS
 f(m,x) =  m * (1-x)/(K3+1-x)
 g(m) = (1-m)/(K1+1-m)
$RATES
 vd = 0.1
 vi = 0.023
 v2 = 0.167
 v4 = 0.1
 vm1 = 0.5
 vm3 = 0.2
 kd = 0.00333
 K1 = 0.1
 K2 = 0.1
 K3 = 0.1
 K4 = 0.1
 Kc = 0.3
 Kd = 0.02
```

The **$Reactions** section contains a list of Reactions as described in Chapter 3, e.g.,

```
 [Br + BrO3 -> HBrO2 + HOBr, k1]
```

The **$Frozen** section contains a list of frozen variables, one variable per line. Frozen variables do not normally change their values in a reaction; however, they are allowed to change their values in (a) an assignment rule; or (b) a **using** reaction (which corresponds to an SBML rate rule).

An example of frozen variables is given in the Oregonator mode file.

```
# Oregonator Data File
# Reference: Field and Noyes, J Chem Phys 60: 1877 (1974)
#
$Reactions
 [Br + BrO3 -> HBrO2 + HOBr, k1]
 [Br + HBrO2 -> 2*HOBr, k2]
 [BrO3 + HBrO2 -> 2*Ce + 2*HBrO2, k3]
 [2*HBrO2-> BrO3 + HOBr, k4] # but BrO3 is frozen
 [Ce -> 0.5*Br, k5]
$IC
 HBrO2 = .001
 Br = .003
 Ce = .05
 BrO3 = .1
 HOBr = 0
$Rates
 k1 = 1.3
 k2 = 2.0E6
 k3 = 34
 k4 = 3000.0
 k5 = 0.02
$Frozen
 BrO3
```

The **`$IC`** section contains initial conditions, one per line, in the form **`variable = value`**

Appendix B

Software License

GNU GENERAL PUBLIC LICENSE[1]
Version 3, 29 June 2007

Preamble

The GNU General Public License is a free, copyleft license for software and other kinds of works.

The licenses for most software and other practical works are designed to take away your freedom to share and change the works. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change all versions of a program–to make sure it remains free software for all its users. We, the Free Software Foundation, use the GNU General Public License for most of our software; it applies also to any other work released this way by its authors. You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for them if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs, and that you know you can do these things.

To protect your rights, we need to prevent others from denying you these rights or asking you to surrender the rights. Therefore, you have certain responsibilities if you distribute copies of the software, or if you modify it: responsibilities to respect the freedom of others.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must pass on to the recipients the same freedoms that you received. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

Developers that use the GNU GPL protect your rights with two steps: (1) assert copyright on the software, and (2) offer you this License giving you legal permission to copy, distribute and/or modify it.

For the developers' and authors' protection, the GPL clearly explains that there is no warranty for this free software. For both users' and authors' sake, the GPL requires that modified versions be marked as changed, so that their problems will not be attributed erroneously to authors of previous versions.

Some devices are designed to deny users access to install or run modified versions of the software inside them, although the manufacturer can do so. This is fundamentally incompatible with the aim of protecting users' freedom to change the software. The systematic pattern of such abuse occurs in the area of products for individuals to use, which is precisely where it is most unacceptable. Therefore, we have designed this version of the GPL to prohibit the practice for those products. If such problems arise substantially in other domains, we stand ready to extend this provision to those domains in future versions of the GPL, as needed to protect the freedom of users.

[1] Copyright (C) 2007 Free Software Foundation, Inc. <http://fsf.org/> Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

Finally, every program is threatened constantly by software patents. States should not allow patents to restrict development and use of software on general-purpose computers, but in those that do, we wish to avoid the special danger that patents applied to a free program could make it effectively proprietary. To prevent this, the GPL assures that patents cannot be used to render the program non-free.

The precise terms and conditions for copying, distribution and modification follow.

Terms and Conditions

0. Definitions.

   "This License" refers to version 3 of the GNU General Public License.

   "Copyright" also means copyright-like laws that apply to other kinds of works, such as semiconductor masks.

   "The Program" refers to any copyrightable work licensed under this License. Each licensee is addressed as "you". "Licensees" and "recipients" may be individuals or organizations.

   To "modify" a work means to copy from or adapt all or part of the work in a fashion requiring copyright permission, other than the making of an exact copy. The resulting work is called a "modified version" of the earlier work or a work "based on" the earlier work.

   A "covered work" means either the unmodified Program or a work based on the Program.

   To "propagate" a work means to do anything with it that, without permission, would make you directly or secondarily liable for infringement under applicable copyright law, except executing it on a computer or modifying a private copy. Propagation includes copying, distribution (with or without modification), making available to the public, and in some countries other activities as well.

   To "convey" a work means any kind of propagation that enables other parties to make or receive copies. Mere interaction with a user through a computer network, with no transfer of a copy, is not conveying.

   An interactive user interface displays "Appropriate Legal Notices" to the extent that it includes a convenient and prominently visible feature that (1) displays an appropriate copyright notice, and (2) tells the user that there is no warranty for the work (except to the extent that warranties are provided), that licensees may convey the work under this License, and how to view a copy of this License. If the interface presents a list of user commands or options, such as a menu, a prominent item in the list meets this criterion.

1. Source Code.

   The "source code" for a work means the preferred form of the work for making modifications to it. "Object code" means any non-source form of a work.

   A "Standard Interface" means an interface that either is an official standard defined by a recognized standards body, or, in the case of interfaces specified for a particular programming language, one that is widely used among developers working in that language.

   The "System Libraries" of an executable work include anything, other than the work as a whole, that (a) is included in the normal form of packaging a Major Component, but which is not part of that Major Component, and (b) serves only to enable use of the work with that Major Component, or to implement a Standard Interface for which an implementation is available to the public in source code form. A "Major Component", in this context, means a major essential component (kernel, window system, and so on) of the specific operating system (if any) on which the executable work runs, or a compiler used to produce the work, or an object code interpreter used to run it.

The "Corresponding Source" for a work in object code form means all the source code needed to generate, install, and (for an executable work) run the object code and to modify the work, including scripts to control those activities. However, it does not include the work's System Libraries, or general-purpose tools or generally available free programs which are used unmodified in performing those activities but which are not part of the work. For example, Corresponding Source includes interface definition files associated with source files for the work, and the source code for shared libraries and dynamically linked subprograms that the work is specifically designed to require, such as by intimate data communication or control flow between those subprograms and other parts of the work.

The Corresponding Source need not include anything that users can regenerate automatically from other parts of the Corresponding Source.

The Corresponding Source for a work in source code form is that same work.

2. Basic Permissions.

   All rights granted under this License are granted for the term of copyright on the Program, and are irrevocable provided the stated conditions are met. This License explicitly affirms your unlimited permission to run the unmodified Program. The output from running a covered work is covered by this License only if the output, given its content, constitutes a covered work. This License acknowledges your rights of fair use or other equivalent, as provided by copyright law.

   You may make, run and propagate covered works that you do not convey, without conditions so long as your license otherwise remains in force. You may convey covered works to others for the sole purpose of having them make modifications exclusively for you, or provide you with facilities for running those works, provided that you comply with the terms of this License in conveying all material for which you do not control copyright. Those thus making or running the covered works for you must do so exclusively on your behalf, under your direction and control, on terms that prohibit them from making any copies of your copyrighted material outside their relationship with you.

   Conveying under any other circumstances is permitted solely under the conditions stated below. Sublicensing is not allowed; section 10 makes it unnecessary.

3. Protecting Users' Legal Rights From Anti-Circumvention Law.

   No covered work shall be deemed part of an effective technological measure under any applicable law fulfilling obligations under article 11 of the WIPO copyright treaty adopted on 20 December 1996, or similar laws prohibiting or restricting circumvention of such measures.

   When you convey a covered work, you waive any legal power to forbid circumvention of technological measures to the extent such circumvention is effected by exercising rights under this License with respect to the covered work, and you disclaim any intention to limit operation or modification of the work as a means of enforcing, against the work's users, your or third parties' legal rights to forbid circumvention of technological measures.

4. Conveying Verbatim Copies.

   You may convey verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice; keep intact all notices stating that this License and any non-permissive terms added in accord with section 7 apply to the code; keep intact all notices of the absence of any warranty; and give all recipients a copy of this License along with the Program.

   You may charge any price or no price for each copy that you convey, and you may offer support or warranty protection for a fee.

5. Conveying Modified Source Versions.

   You may convey a work based on the Program, or the modifications to produce it from the Program, in the form of source code under the terms of section 4, provided that you also meet all of these conditions:

   (a) The work must carry prominent notices stating that you modified it, and giving a relevant date.
   (b) The work must carry prominent notices stating that it is released under this License and any conditions added under section 7. This requirement modifies the requirement in section 4 to "keep intact all notices".
   (c) You must license the entire work, as a whole, under this License to anyone who comes into possession of a copy. This License will therefore apply, along with any applicable section 7 additional terms, to the whole of the work, and all its parts, regardless of how they are packaged. This License gives no permission to license the work in any other way, but it does not invalidate such permission if you have separately received it.
   (d) If the work has interactive user interfaces, each must display Appropriate Legal Notices; however, if the Program has interactive interfaces that do not display Appropriate Legal Notices, your work need not make them do so.

   A compilation of a covered work with other separate and independent works, which are not by their nature extensions of the covered work, and which are not combined with it such as to form a larger program, in or on a volume of a storage or distribution medium, is called an "aggregate" if the compilation and its resulting copyright are not used to limit the access or legal rights of the compilation's users beyond what the individual works permit. Inclusion of a covered work in an aggregate does not cause this License to apply to the other parts of the aggregate.

6. Conveying Non-Source Forms.

   You may convey a covered work in object code form under the terms of sections 4 and 5, provided that you also convey the machine-readable Corresponding Source under the terms of this License, in one of these ways:

   (a) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by the Corresponding Source fixed on a durable physical medium customarily used for software interchange.
   (b) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by a written offer, valid for at least three years and valid for as long as you offer spare parts or customer support for that product model, to give anyone who possesses the object code either (1) a copy of the Corresponding Source for all the software in the product that is covered by this License, on a durable physical medium customarily used for software interchange, for a price no more than your reasonable cost of physically performing this conveying of source, or (2) access to copy the Corresponding Source from a network server at no charge.
   (c) Convey individual copies of the object code with a copy of the written offer to provide the Corresponding Source. This alternative is allowed only occasionally and noncommercially, and only if you received the object code with such an offer, in accord with subsection 6b.
   (d) Convey the object code by offering access from a designated place (gratis or for a charge), and offer equivalent access to the Corresponding Source in the same way through the same place at no further charge. You need not require recipients to copy the Corresponding Source along with the object code. If the place to copy the object code is a network server, the Corresponding Source may be on a different server (operated by you or a third party) that supports equivalent copying facilities, provided you maintain clear directions next to the object code saying where to

find the Corresponding Source. Regardless of what server hosts the Corresponding Source, you remain obligated to ensure that it is available for as long as needed to satisfy these requirements.

(e) Convey the object code using peer-to-peer transmission, provided you inform other peers where the object code and Corresponding Source of the work are being offered to the general public at no charge under subsection 6d.

A separable portion of the object code, whose source code is excluded from the Corresponding Source as a System Library, need not be included in conveying the object code work.

A "User Product" is either (1) a "consumer product", which means any tangible personal property which is normally used for personal, family, or household purposes, or (2) anything designed or sold for incorporation into a dwelling. In determining whether a product is a consumer product, doubtful cases shall be resolved in favor of coverage. For a particular product received by a particular user, "normally used" refers to a typical or common use of that class of product, regardless of the status of the particular user or of the way in which the particular user actually uses, or expects or is expected to use, the product. A product is a consumer product regardless of whether the product has substantial commercial, industrial or non-consumer uses, unless such uses represent the only significant mode of use of the product.

"Installation Information" for a User Product means any methods, procedures, authorization keys, or other information required to install and execute modified versions of a covered work in that User Product from a modified version of its Corresponding Source. The information must suffice to ensure that the continued functioning of the modified object code is in no case prevented or interfered with solely because modification has been made.

If you convey an object code work under this section in, or with, or specifically for use in, a User Product, and the conveying occurs as part of a transaction in which the right of possession and use of the User Product is transferred to the recipient in perpetuity or for a fixed term (regardless of how the transaction is characterized), the Corresponding Source conveyed under this section must be accompanied by the Installation Information. But this requirement does not apply if neither you nor any third party retains the ability to install modified object code on the User Product (for example, the work has been installed in ROM).

The requirement to provide Installation Information does not include a requirement to continue to provide support service, warranty, or updates for a work that has been modified or installed by the recipient, or for the User Product in which it has been modified or installed. Access to a network may be denied when the modification itself materially and adversely affects the operation of the network or violates the rules and protocols for communication across the network.

Corresponding Source conveyed, and Installation Information provided, in accord with this section must be in a format that is publicly documented (and with an implementation available to the public in source code form), and must require no special password or key for unpacking, reading or copying.

7. Additional Terms.

"Additional permissions" are terms that supplement the terms of this License by making exceptions from one or more of its conditions. Additional permissions that are applicable to the entire Program shall be treated as though they were included in this License, to the extent that they are valid under applicable law. If additional permissions apply only to part of the Program, that part may be used separately under those permissions, but the entire Program remains governed by this License without regard to the additional permissions.

When you convey a copy of a covered work, you may at your option remove any additional permissions from that copy, or from any part of it. (Additional permissions may be written to require their own removal in certain cases when you modify the work.) You may place additional permissions on material, added by you to a covered work, for which you have or can give appropriate copyright permission.

Notwithstanding any other provision of this License, for material you add to a covered work, you may (if authorized by the copyright holders of that material) supplement the terms of this License with terms:

(a) Disclaiming warranty or limiting liability differently from the terms of sections 15 and 16 of this License; or

(b) Requiring preservation of specified reasonable legal notices or author attributions in that material or in the Appropriate Legal Notices displayed by works containing it; or

(c) Prohibiting misrepresentation of the origin of that material, or requiring that modified versions of such material be marked in reasonable ways as different from the original version; or

(d) Limiting the use for publicity purposes of names of licensors or authors of the material; or

(e) Declining to grant rights under trademark law for use of some trade names, trademarks, or service marks; or

(f) Requiring indemnification of licensors and authors of that material by anyone who conveys the material (or modified versions of it) with contractual assumptions of liability to the recipient, for any liability that these contractual assumptions directly impose on those licensors and authors.

All other non-permissive additional terms are considered "further restrictions" within the meaning of section 10. If the Program as you received it, or any part of it, contains a notice stating that it is governed by this License along with a term that is a further restriction, you may remove that term. If a license document contains a further restriction but permits relicensing or conveying under this License, you may add to a covered work material governed by the terms of that license document, provided that the further restriction does not survive such relicensing or conveying.

If you add terms to a covered work in accord with this section, you must place, in the relevant source files, a statement of the additional terms that apply to those files, or a notice indicating where to find the applicable terms.

Additional terms, permissive or non-permissive, may be stated in the form of a separately written license, or stated as exceptions; the above requirements apply either way.

8. Termination.

You may not propagate or modify a covered work except as expressly provided under this License. Any attempt otherwise to propagate or modify it is void, and will automatically terminate your rights under this License (including any patent licenses granted under the third paragraph of section 11).

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, you do not qualify to receive new licenses for the same material under section 10.

9. Acceptance Not Required for Having Copies.

   You are not required to accept this License in order to receive or run a copy of the Program. Ancillary propagation of a covered work occurring solely as a consequence of using peer-to-peer transmission to receive a copy likewise does not require acceptance. However, nothing other than this License grants you permission to propagate or modify any covered work. These actions infringe copyright if you do not accept this License. Therefore, by modifying or propagating a covered work, you indicate your acceptance of this License to do so.

10. Automatic Licensing of Downstream Recipients.

    Each time you convey a covered work, the recipient automatically receives a license from the original licensors, to run, modify and propagate that work, subject to this License. You are not responsible for enforcing compliance by third parties with this License.

    An "entity transaction" is a transaction transferring control of an organization, or substantially all assets of one, or subdividing an organization, or merging organizations. If propagation of a covered work results from an entity transaction, each party to that transaction who receives a copy of the work also receives whatever licenses to the work the party's predecessor in interest had or could give under the previous paragraph, plus a right to possession of the Corresponding Source of the work from the predecessor in interest, if the predecessor has it or can get it with reasonable efforts.

    You may not impose any further restrictions on the exercise of the rights granted or affirmed under this License. For example, you may not impose a license fee, royalty, or other charge for exercise of rights granted under this License, and you may not initiate litigation (including a cross-claim or counterclaim in a lawsuit) alleging that any patent claim is infringed by making, using, selling, offering for sale, or importing the Program or any portion of it.

11. Patents.

    A "contributor" is a copyright holder who authorizes use under this License of the Program or a work on which the Program is based. The work thus licensed is called the contributor's "contributor version".

    A contributor's "essential patent claims" are all patent claims owned or controlled by the contributor, whether already acquired or hereafter acquired, that would be infringed by some manner, permitted by this License, of making, using, or selling its contributor version, but do not include claims that would be infringed only as a consequence of further modification of the contributor version. For purposes of this definition, "control" includes the right to grant patent sublicenses in a manner consistent with the requirements of this License.

    Each contributor grants you a non-exclusive, worldwide, royalty-free patent license under the contributor's essential patent claims, to make, use, sell, offer for sale, import and otherwise run, modify and propagate the contents of its contributor version.

    In the following three paragraphs, a "patent license" is any express agreement or commitment, however denominated, not to enforce a patent (such as an express permission to practice a patent or covenant not to sue for patent infringement). To "grant" such a patent license to a party means to make such an agreement or commitment not to enforce a patent against the party.

    If you convey a covered work, knowingly relying on a patent license, and the Corresponding Source of the work is not available for anyone to copy, free of charge and under the terms of this License, through a publicly available network server or other readily accessible means, then you must either (1) cause the Corresponding Source to be so available, or (2) arrange to deprive yourself of the benefit of the patent license for this particular work, or (3) arrange, in a manner consistent with the requirements of this License, to extend the patent license to downstream recipients. "Knowingly relying" means you have actual knowledge that, but for the patent license, your conveying the covered

work in a country, or your recipient's use of the covered work in a country, would infringe one or more identifiable patents in that country that you have reason to believe are valid.

If, pursuant to or in connection with a single transaction or arrangement, you convey, or propagate by procuring conveyance of, a covered work, and grant a patent license to some of the parties receiving the covered work authorizing them to use, propagate, modify or convey a specific copy of the covered work, then the patent license you grant is automatically extended to all recipients of the covered work and works based on it.

A patent license is "discriminatory" if it does not include within the scope of its coverage, prohibits the exercise of, or is conditioned on the non-exercise of one or more of the rights that are specifically granted under this License. You may not convey a covered work if you are a party to an arrangement with a third party that is in the business of distributing software, under which you make payment to the third party based on the extent of your activity of conveying the work, and under which the third party grants, to any of the parties who would receive the covered work from you, a discriminatory patent license (a) in connection with copies of the covered work conveyed by you (or copies made from those copies), or (b) primarily for and in connection with specific products or compilations that contain the covered work, unless you entered into that arrangement, or that patent license was granted, prior to 28 March 2007.

Nothing in this License shall be construed as excluding or limiting any implied license or other defenses to infringement that may otherwise be available to you under applicable patent law.

12. No Surrender of Others' Freedom.

    If conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot convey a covered work so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not convey it at all. For example, if you agree to terms that obligate you to collect a royalty for further conveying from those to whom you convey the Program, the only way you could satisfy both those terms and this License would be to refrain entirely from conveying the Program.

13. Use with the GNU Affero General Public License.

    Notwithstanding any other provision of this License, you have permission to link or combine any covered work with a work licensed under version 3 of the GNU Affero General Public License into a single combined work, and to convey the resulting work. The terms of this License will continue to apply to the part which is the covered work, but the special requirements of the GNU Affero General Public License, section 13, concerning interaction through a network will apply to the combination as such.

14. Revised Versions of this License.

    The Free Software Foundation may publish revised and/or new versions of the GNU General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

    Each version is given a distinguishing version number. If the Program specifies that a certain numbered version of the GNU General Public License "or any later version" applies to it, you have the option of following the terms and conditions either of that numbered version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of the GNU General Public License, you may choose any version ever published by the Free Software Foundation.

    If the Program specifies that a proxy can decide which future versions of the GNU General Public License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Program.

Later license versions may give you additional or different permissions. However, no additional obligations are imposed on any author or copyright holder as a result of your choosing to follow a later version.

15. Disclaimer of Warranty.

    THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

16. Limitation of Liability.

    IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MODIFIES AND/OR CONVEYS THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

17. Interpretation of Sections 15 and 16.

    If the disclaimer of warranty and limitation of liability provided above cannot be given local legal effect according to their terms, reviewing courts shall apply local law that most closely approximates an absolute waiver of all civil liability in connection with the Program, unless a warranty or assumption of liability accompanies a copy of the Program in return for a fee.

End of Terms and Conditions

How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively state the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

```
<one line to give the program's name and a brief idea of what it does.>

Copyright (C) <textyear>  <name of author>

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

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

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

Also add information on how to contact you by electronic and paper mail.

If the program does terminal interaction, make it output a short notice like this when it starts in an interactive mode:

```
<program>  Copyright (C) <year>  <name of author>

This program comes with ABSOLUTELY NO WARRANTY; for details type `show w'.
This is free software, and you are welcome to redistribute it
under certain conditions; type `show c' for details.
```

The hypothetical commands `show w` and `show c` should show the appropriate parts of the General Public License. Of course, your program's commands might be different; for a GUI interface, you would use an "about box".

You should also get your employer (if you work as a programmer) or school, if any, to sign a "copyright disclaimer" for the program, if necessary. For more information on this, and how to apply and follow the GNU GPL, see http://www.gnu.org/licenses/.

The GNU General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Lesser General Public License instead of this License. But first, please read http://www.gnu.org/philosophy/why-not-lgpl.html.

Bibliography

[1] Belousouv BP. [A periodic reaction and its mechanism (Russian)]. [Compilation of Abstracts on Radiation Medicine], **147**:145 (1959)

[2] Chickarmane V, Peterson C. *A Computational Model for Understanding Stem Cell, Trophectoderm and Endoderm Lineage Determination.* **PLoS ONE 3**(10):e3478. doi:10.1371/journal.pone.0003478

[3] Cornish-Bowden A. *An automatic method for deriving steady-state rate equations* **Biochem J. 165**:55-59 (1977)

[4] Elowitz MB, Leibler S. *A synthetic oscillatory network of transcriptional regulators.* **Nature. 403**:335-338 (2000).

[5] Field RJ, Koros RM, Noyes RM *Oscillations in Chemical Systems II. Thorough analysis of temporal oscillations in the Ce-BrO3- - malonic acid system.* **J. Am. Chem. Soc.**:94(25), 8649-64 (1972).

[6] Field RJ, Noyes RM *Oscillations in Chemical Systems IV. Limit cycle behavior in a model of a real chemical reaction.* **J. Chem. Phys. 60**: 1877-84 (1974).

[7] Goldbeter A. *A minimal cascade model for the mitotic oscillator involving cyclin and cdc2 kinase.* **Proc. Natl. Acad. Sci. USA 88**:9107-1101 (1991).

[8] Hoffmann A, Levchenko A, Scott ML, Baltimore D. *THe IkappaB-NF-kappaB signaling module:temporal control and selective gene activation.* **Science. 298**: 1241 (2002).

[9] Hucka M, Hoops S, Keating SM, Le Novère N, Sahle S, Wilkinson DJ. *Systems Biology Markup Language (SBML) Level 2: Structures and Facilities for Model Definitions.* **Nature Precedings** (doi:10.1038/npre.2008.2715.1) http://precedings.nature.com/documents/2715/version/1

[10] King EL, Altman C. *A Schematic Method of Deriving the Rate Laws for Enzyme-Catalyzed Reactions.* **J. Phys. Chem. 60**: 1375-1378 (1956).

[11] Ko CL, Wang F-S, Chao Y-P, Chen T-W. *S-system approach to modeling recombinant Escherichia coli growth by hybrid differential evolution with data collocation* **Biochem. Eng. J.**, 28:10-16 (2006).

[12] Monod J, Wyman J, Changeux JP. *On the Nature of Allosteric Transitions: A Plausible Model.* **J. Mol. Bio. 12**:88-118 (1965).

[13] Najdi TS, Yang CR, Shapiro BE, Hatfield W, Mjolsness E. *Application of a Generalized MWC Model for the Mathematical Simulation of Metabolic Pathways Regulated by Allosteric Enzymes.* **J. Bioinf. Comp. Bio. 4**(2): 335-356 (2006).

[14] Savageau MA. *Biochemical systems analysis. II. The steady-state solutions for an n-pool system using a power-law approximation.* **J Theor Biol 25**:370-379 (1969).

[15] Savageau MA. *Biochemical systems analysis. 3. Dynamic solutions using a power-law approximation.* **J Theor Biol 26**:215-226 (1970).

[16] Shapiro BE, Levchenko A, Meyerowitz EM, Wold BJ, Mjolsness ED. *Cellerator: extendeing a computer algebra system to include biochemical arrows for signal transduction simulations.* **Bioinformatics, 19**(5):677-8 (2003). (doi: 10.1093/bioinformatics/btg042)

[17] Shapiro BE, Mjolsness ED. *Developmental Simulations with Cellerator.* Second International Conference on Systems Biology (ICSB-2001), Pasadena, CA, USA (2001).

[18] Yang CR, Shapiro BE, Mjolsness ED, Hatfield ED. *AN enzyme mechanism language for the mathematical modeling of metabolic pathways.* **Bioinformatics, 21**(6):774-780 (2005) (doi: 10.1093/bioinformatics/bti068)

[19] Yosiphon G. **Stochastic Parameterized Grammars : Formalization, Inference and Modeling Applications**, Ph.D. Thesis, Univ. Cal., Irvine (2009).

[20] Yosiphon G, Mjolsness E. *Towards the Inference of Stochastic Biochemical Network and Parameterized Grammar Models* in **Learning and Inference in Computational Systems Biology**, ed. N. Lawrence et. al. MIT Press (2010).

[21] Yosiphon G, Gokoffski KK, Lander AD, Calof AL, Mjolsness E. *Dynamical Grammar Modeling of Growth and Lamination of the Olfactory Epithelium.* **International Conference on Systems Biology (ICSB)** (2007).

[22] Zhabotinsky AM. [Periodic processes of malonic acid oxidation in a liquid phase. (Russian)] [Biofizika],**9**:306-311 (1964).