Setting up and simulating ODE models

Supplementary Material to the Chapter in qbio textbook.

Herbert M. Sauro

Department of Bioengineering, , University of Washington, Seattle, 98195-5061, WA

1 Introduction

The use of simulation in cellular and molecular biology has a long, if uneven, tradition spanning almost seventy years [1]. With the advent of high-throughput technologies, interest in using more quantitative approaches to enhance our understanding of cellular dynamics has increased dramatically [2, 3, 4, 5]. As part of this trend there has been a significant rise in the availability of computer software to help us model, simulate and analyze dynamic models of cellular processes. In this chapter we will briefly survey the available simulation software for systems biology with a particular focus on python as a productive platform for model development and simulation. We will also provide illustrative examples and tips on debugging and improving models. We will not cover the theory of modeling of biochemical systems, instead we will refer the reader to the following published work [6, 7, 8, 9].

Using ordinary differential equations (ODEs) to simulate biochemical networks has a long history dating back at least to the 1940s [10]. The earliest simulations relied on converting ODEs into either mechanical or electrical analogs. It was only in the late 1950s, with the advent of digital computers and the development of specialized software tools [11] that the ability to simulate biochemical networks became more widely available. In the intervening years up to 1980, a handful of other software applications were developed [12, 13, 14]. Since the early 1990s, there has been a significant increase in interest in modeling biochemical processes and a wide range of tools is now available to the budding systems biologist. Many tools have been developed by practicing scientists and are therefore available for free (often open source) while others are commercial such as SimBiology from MathWorks – http://www.mathworks.com/.

Given the increase in interest in quantitative biology [15, 16], the number and variety of tools and approaches has likewise risen. In addition to the availability of simulation tools, the field has also seen the emergence of standard representations such as SBML [17] and CellML [18] what allows users to freely exchange models between different tools. This is particularly important for academic software which tends to have a short life span. The use of standard exchange formats prevents the loss of models as tools become orphaned or no longer functional.

2 The Need for Software

One question that arises is why develop specialized software to model biochemical networks? Given the availability of both generic commercial and freely available tools for numerical analysis one might ask if there is such a need. The first reason is that specialized tools reduce errors that occur when transcribing a reaction scheme (that is a biological representation) into the mathematical formalism ready for simulation. Deriving the mathematical equations by hand is often a frequent source of error especially in published papers but also in large models. The second reason is that developing software offers an opportunity to codify and develop new numerical algorithms or new theoretical approaches that are specific to problems found in systems biology. Such examples include metabolic control analysis [19, 20] or developing more efficient methods for network analysis [21]. Although it is possible to carry out many analyzes in tools such as Matlab or Mathematica, the process tends to be more tedious and error prone and certain kinds of analyses are unavailable to the user without the application of considerable effort. Finally the issue of reproducibility is becoming more important. The techniques we use to simulate biological systems should be transparent and freely available. This, to some degree, precludes commercial software where algorithms are proprietary and the software by its very nature is not freely available.

2.1 Standards

With the increase in the number of incompatible simulation tools since the year 2000, it was realized by two communities that some form of standardization for model exchange was necessary. The two standards that emerged, include CellML and SBML. CellML is primarily a notation for representing biological models in a strict mathematical form, as a result it is in principle completely general. SBML on the other hand uses a biologically inspired notation to represent cellular network from which a mathematical model can be generated. Each has its strengths and weaknesses, SBML has a simpler structure compared to CellML, as a result there is more software support for SBML. Most software tools at the present time support import and export of SBML. Both standards have very active communities with intracellular models being primarily the domain of SBML and physiological models for CellML. A more detailed review of standards in systems biology can be found at [22, 23, 24].

3 Software Tools

In the last ten years or so, numerous software applications have been written to support modeling of biochemical processes. Some applications work on single operating platforms, others work across multiple platforms; some are trivial to install, while others require considerable persuasion to make them work. There is also a wide variety of user interfaces such as graphical interfaces incorporating menus, buttons, lists etc., network design interfaces that allow users to draw a pathway on the screen and text based scripts that enable users to control and define a computer model precisely. Application capabilities tend to vary enormously, some tools attempt to be broad, offering many different capabilities, others choose to specialize in one particular area. Some of the tools are extensible, that is new functionality can be added by a user either through a scripting language or by writing small plugin extensions. Most have some form of reference documentation, including tutorials to help users get started. A visit to the software guide at the SBML.org web site will reveal over two hundred different tools for modeling or working with biochemical networks. The majority of these have been developed in the last fifteen years or so. However, although there appears to be a large number of offerings, many are not actively developed and have become orphaned software. Given the space limits and the wide range of tools, we refer the reader to existing reviews [25, 26, 27, 28, 29]. Readers can also examine the comparison web site maintained by the VCell group at http://ntcnp.org/twiki/bin/view/VCell/SBMLFeatureToolMatrix. The feature matrix at the VCell site is a user contributed table that lists the capabilities of each tool. Although there is functional overlap between the available tools, many offer specialized functionality. For example, BioNetGen and

PySB [30, 31, 32] are rule based simulators, CellDesigner [33], PathwayDesigner[1] and TinkerCell [34] are visual design tools, COPASI [35] specializes in parameter fitting, iBioSim [36] has specific support for synthetic biology, JSim [37] is focused more on physiological models, JigCell has specific support for model validation based on regression testing [38], PySCeS [39] offers particularly good support for control analysis [6, 40], MathSBML [41] is a Mathematica based tool, OpenCor `http://www.opencor.ws/` is geared towards CellML based models, and last but not least VCell [42] is specialized to run single cell spatial models.

Software Tool	GUI	Visual	Scripting	Web Site
BioNetGen			✓	`http://bionetgen.org/`
CellDesigner		✓		`www.celldesigner.org`
COPASI	✓		Addon, Python	`copasi.org`
iBioSim		✓		`www.async.ece.utah.edu/iBioSim`
JSim			MML	`www.physiome.org/jsim`
JigCell	✓			`jigcell.cs.vt.edu`
MathSBML			✓ Mathematica	`mathsbml.com/mathsbml`
PySCeS			✓ Python	`pysces.sourceforge.net`
PySB			✓ Python	`pysb.org`
PathwayDesigner		✓		`pathwaydesigner.org`
SBW	✓	✓	✓	`sbw.sourceforge.net`
Tellurium	✓		✓ Python	`tellurium.analogmachine.org`
TinkerCell		✓	✓ Python	`www.tinkercell.com`
VCell	✓	✓	✓ VCML	`www.nrcam.uchc.edu`

Table 1: List of SBML compliant Open Source Tools for ODE Modeling

SBML

Whereas CellML attempts to be highly comprehensive, SBML was designed to meet the immediate needs of the modeling community and is therefore more focused on a particular problem set. One result of this is that the standard is simpler compared to CellML although more recent revisions add new functionality so that the difference in complexity between CellML and SBML is becoming less significant. Like CellML, SBML is based on XML, however unlike CellML, it takes a different approach to representing cellular models. The way SBML represents models, closely maps the way existing modeling packages represent models. Whereas CellML represents models mathematically, SBML represent models as a list of chemical transformations. Since every process in a biological cell can ultimately be broken down into one or more chemical transformations this was a natural representation to use. However SBML does not have generalized elements such as components and connections, SBML employs specific elements to represent spatial compartments, molecular species and chemical transformations. In addition to these, SBML also has provision for rules which can be used to represent constraints, derived values and general math which for one reason or another cannot be transformed into a chemical scheme.

SBML, like any standard, evolves with time [43]. Major revisions of the standard are captured in levels, while minor modifications and clarifications are captured in versions. An example of a major

[1] `pathaydesigner.org`

change within the standard would be the use of MathML in level two of SBML, whereas level one encoded infix strings to denote reaction rates and rules. A minor change on the other hand would for example be the introduction of semantic annotations that can be added to SBML level two version three, whereas this was not possible in a supported fashion in earlier versions (see section 4.3.3). As of this writing, SBML level three is still in development. With level three the standard will develop in an extensible manner. This means there will be a set of core features that must be supported around which additional features, such as spatial modeling, can be included. There has been little effort to develop methods to inter-convert between SBML and CellML, however Schilstra from the UK developed a tool called CellML2SBML [44] that allows users to convert CellML based models into SBML. It is also possible to use tools such as VCell to inter-convert between these two standards.

CellML

CellML [18, 45] represents cellular models using a mathematical description. In addition, CellML represents entities using a component based approach where relationships between components are represented by connections. The literal translation of the mathematics however goes much further, in fact the representation that CellML uses is very reminiscent of the way an engineer might wire up an analog computer to solve the equations (though without specifying the integrators). As a result CellML is very general and in principal could probably represent any system that has a mathematical description. CellML is also very precise in that every item in a model is defined explicitly. However, the generality and explicit nature of CellML also results in increased complexity especially for software developers.

Another key aspect of CellML is its provision for metadata support. The metadata can be used to provide a context for a model, such as the author name, when it was created and what additional documents are available for it's description. CellML uses standard XML based metadata containers such as RDF and within RDF the Dublin Core. CellML metadata, such as BioPAX [46] (http://www.biopax.org), is how biological information can be introduced into a CellML model. The CellML team has amassed a very large suite (hundreds) of models(www. cellml.org/models/) which provides many real examples of CellML syntax.

SED-ML

A key issue in modeling is the ability to reproduce published models. In the past this has proved to be difficult due to errors or incomplete data published on the model. With the raise of modeling standards reproducibility of models is now more easier to achieve. As well as model standards such as SBML and CellML, there are also emerging standards on who models should be run. For example a modeling paper might have a number of graphs that illustrate specific simulation scenarios. Often the data used to generate the graphs is either descried in the graph legend or sometime in the form of Matlab or other types of program scripts. The most popular standard for describing simulation experiments is SED-ML (Simulation Experiment Description Markup Language). SED-ML is therefore an important step towards enabling complete reproducibility of cellular models. In order to describe a simulation experiment a number of aspects must be present, these include: 1) The model itself in some suitably coded form; 2) the set of parameter values which should be applied to the model; 3) the algorithm that should be used to simulate the model; and 4) how the simulation output should be formatted in order to produce the expected numerical result [Waltemath, 2011]. This information can be encoded in SED-ML currently as XML. SED-ML continues to be developed now permits the description of a time course simulation(s), steady-state parameter scans and more recently parameter optimization.

Given that a reproducible model requires a variety of different kinds of information, including the model, the experiment and data, an additional standard, the COMBINE archive was developed to package all necessary files into a single archive [47]. A typical COMBINE archive might include a SBML model, a SED-ML describing the simulation experiments, numerical algorithms used and data outputs as well as any experimental data that might be required for model fitting or comparison with the simulated data. Two tools currently support the complete work-flow for creating a COMBINE archive, these include PySCeS [39] and Tellurium [ref].

4 Further Details on Antimony Syntax

The main text of the qbio textbook briefly described some essential syntax notation for writing Antimony models. Here we will give additional details. Reactions can be named using the syntax `J1:`, for example: `J1:  A + B -> C`. This means the reaction has a name, `J1`. Named reaction are useful if we want to refer to the flux or reaction rate; kinetic rate laws come immediately after the reaction. Rate laws can be written using any of the usual mathematical operators (`+, -, *, etc`) or common functions (`sin, cos, log, etc`). Rate laws can include allosteric regulators, inhibitors etc. If only the stoichiometry matrix is required, it is not necessary to enter a full kinetic law, a simple `...  -> S1; v;` is sufficient. Here is an example of a reaction that is governed by a Michaelis-Menten rate law: `A -> B; Vm*A/(Km + A);` Note the semicolons. Here is a more complex example involving multiple reactions with different rate laws:

```
    MainFeed:    $X0 -> S1;  Vm*X0/(Km + X0 + S1/Ki);
    TopBranch:    S1 -> $X1; Vm1*S1^n/(Km1 + S1)^n;
    BottomBranch: S1 -> $X2; k1*S1;
```

Running Simulations Once a model has been described using Antimony, it can now be loaded into the libRoadRunner simulator [48]. To do this we import the `tellurium` package from which we can use the `loada` method. To load an Antimony model into libRoadRunner we call `te.loada` with the antimony model as the string argument. For example:

```
# Import the tellurium package as te
import tellurium as te
r = te.loada ('''
    const Xo, X1;   // Boundary species
    var S1, S2;     // Floating species

    J1:  X0 -> S1;  Vm1*X0/(Km + X0);
    J2:  S1 -> S2;  Vm2*S1/(Km1 + S1);
    J3:  S2 -> X1;  Vm3*S2/(Km2 + S2);
''')
```

Internally, libRoadRunner converts an Antimony model into a set of differential equations or propensity functions for a stochastic simulation [49]. For example the above model would be converted into the following two differential equations:

$$\frac{\texttt{dS}_1}{\texttt{dt}} = \texttt{Vm1*X0/(Km + X0) - Vm2*S1/(Km1 + S1)}$$

$$\frac{\texttt{dS}_2}{\texttt{dt}} = \texttt{Vm2*S1/(Km1 + S1) - Vm3*S2/(Km2 + S2)}$$

Note that there are no differential equations for X_o and X_1 in this model. This is because they are fixed and do not change in time. If the reactions have non-unity stoichiometry, this is taken into account when the differential equations are derived. To initialize the concentrations and parameters in a model we can add assignments after the network is declared, for example:

```
    S1 -> S2;  Vm*S1/(Km + S1);
    S1 = 3.4; S2 = 0.0;
    Vm = 12;  Km = 0.1;
```

In addition to loading Antimony models, libRoadRunner can also load an export standard SBML models. To simulate a model we use the libRoadRunner `simulate` method. In the following examples, `r` represents the roadrunner object. The following illustrates two ways to use the simulate method:

```
>>> result = r.simulate (0, 10, 100)
>>> result = r.simulate (0, 10, 100, ['time', 'S1'])
```

The first argument is the time start, followed by time end and the number of points to generate. The fourth argument is optional but can be used to specify what results should be returned as columns in the `result` matrix. If the fourth argument is absent then the first column will be time and subsequent columns the floating species concentrations. To plot the results of a simulation use `r.plot()`, see Figure 1.

```
import tellurium as te
rr = te.loada (’’’
    J1:  A -> B; k1*S1;
    k1 = 0.1; A = 10; B = 0;
’’’)
m = r.simulate (0, 50, 200)
r.plot()
```

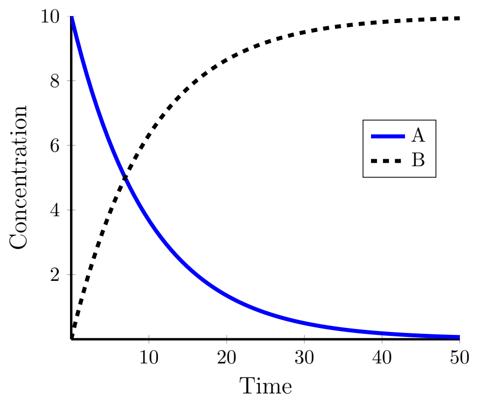

Figure 1: A simple model defined using Antimony and simulated using libRoadRunner.

Setting and Getting Values Often during a modeling experiment parameter values and initial condition need to be inspected or changed. For model parameters, reaction rates and species concentration, one can use the syntax `r.X`, where `X` is the name of a species, parameter or reaction names. For example:

```
r.k1 = 1.2; r.ATP = 1.5
print r.k1, r.ATP
```

The initial conditions represent the starting values of the species for a simulation. The easiest way to set the starting values is to use the method `reset`. This copies the current set of initial conditions into the model. There are a number of methods to change initial conditions, for example:

```
# Set the initial concentration of S1
r.model['init([S1])'] = 3.4
print r.model['init([S1])'
```

The `init([S1])` string is used to refer to the initial condition for `S1`. We can set both the amount or concentration of a species. For example to set the amount we simply leave out the square bracket.

Retrieving Reaction Rates or Fluxes The fluxes through the individual reactions can be obtained by either referencing the name of the reaction (e.g. `J1`), or via the short-cut command `rv` which returns the vector of all reaction rates. Note that indexing of vectors is from zero.

Resetting Models There are three ways to reset a model: 1) `reset`: Set the species concentrations to the current initial conditions; 2) `resetAll`: Sets the species concentrations to the current initial conditions and resets the values of the parameters to those that were originally loaded; and 3) `resetToOrigin`: resets the model back to what is was when the model was first loaded.

5 Debugging a Model

Building working and useful models takes experience. In this section will be describe some issues that can sometimes confound the beginner. Anyone who has written software will know that debugging is a major part of the effort. This is equally true in building a model, especially a complex one. There are various approaches one can use to track down misbehaving models.

No Steady State Many simulation tools allow the user to compute the steady state with a single command or click on a button marked steady state. The alternative is to run a time course simulation of sufficient time that the system settles to a steady state. Either way, the modeler is sometimes confronted with the situation where it appears there is no steady state. Often this is the result of a bad model. Failure to reach steady state can be manifest in a number of ways. When doing a time course simulation, the most common effect is that concentrations either go to infinity or go negative. When using the steady state function, a common error message that is returned is usually something like 'Jacobian Singular' or "Newton step failed. Damping limit exceeded' which may not make much sense to a beginner.

Another reason for failure when using a dedicated steady state function is due to numerical errors and is often the result of so-called conserved moieties in the model [50, 51]. Conserved moieties are parts of the model where the mass of a group of species is conserved. A typical example is a protein phosphorylation cycle where the amount of protein and phosphorylated protein is fixed. In these cases a simulator will often be required to adjust the algorithm to cope with the numerical difficulties [52, 21]. Tools such as COPASI [35], Tellurium/libRoadRunner [48], Jarnac [53] or SBW [54, 55] have been written specifically to deal with such cases.

Concentrations tending to infinity or negative It is not uncommon for beginners to build models where the concentrations of one or more species tend to infinity or go negative.

```
import tellurium as te
import roadrunner

r = te.loada("""
 $A -> B; vo;
 B + C -> D; k1*B*C - k2*D;

 A = 1;  C = 1.5;
 vo = 0.6; k1 = 0.34; k2 = 0.78;
""")
m = r.simulate (0, 100, 100)
r.plot()
```

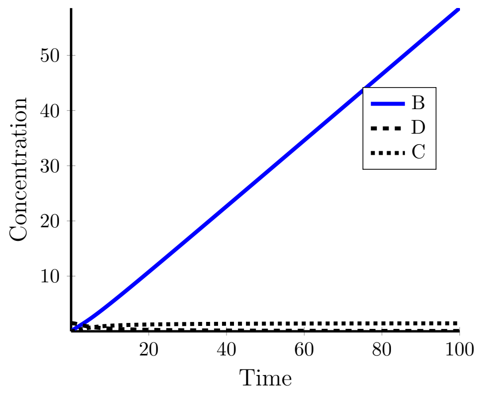

Figure 2: A bad model where one of the species, B, increases indefinitely.

```
import tellurium as te

rr = te.loada (’’’
     $Xo -> S1;  k1*Xo;
      S1 -> $X1; k2*S1;

     Xo = 10; k1 = 0.3; k2 = 0.15;

     at time > 40: S1 = S1*1.6
’’’)

m = rr.simulate (0, 80, 100)
rr.plot()
```

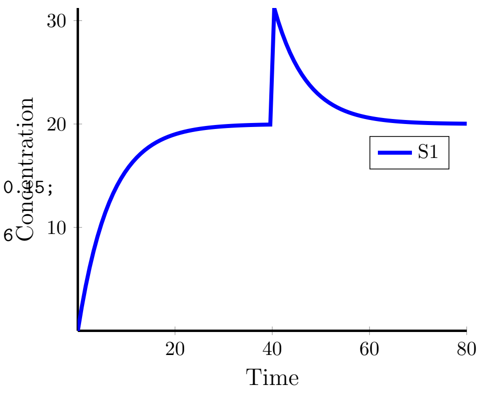

Figure 3: A model showing an event triggered at time 40

```
import tellurium as te
import roadrunner

r = te.loada("""
 # This reaction is commented out
 #$A -> B; vo;
 B + C -> D; k1*B*C - k2*D;

 A = 1;  C = 1.5;
 vo = 0.6; k1 = 0.34; k2 = 0.78;
""")
m = r.simulate (0, 100, 100)
r.plot()
```

Consider the model shown in Figure 2. Any attempt to compute the steady state of this model will fail. Carrying out a time course simulation is a useful way to debug such situations and in this case will reveal the output shown in Figure 2. From the simulation we can see that the concentration of species, B is increasing and in fact will continue increasing to infinity. The reason for this is not difficult to understand. B is being produced at a *fixed* rate `vo`. B in turn binds to a limited amount of C to produce D. Once C has been consumed B will start to climb indefinitely. One solution to this problem is to ensure that the fixed rate for creating B responds to B itself. The easiest way to do this is to make the first reaction reversible, that is make the rate law for the first reaction: `vo - k*B`. As B increases, the back reaction will eventually match the forward rate `vo` at which point B will stop increasing. This is a simple example but in a large model where we find concentrations that are ever increasing it might not be so obvious to locate the problem. In a large model, the key to debugging is to remove reactions until the problem goes away. In the simple example we could have commented out the first reaction and noticed that B no longer tends to infinity.

Another example is shown in Figure 4 where the species, `S1` goes to infinity and it is less obvious why this happens compared to the first example. The key is the second step where we have an irreversible reaction catalyzed by a saturable enzyme. The rate of inflow into the pathway is given by `k1*S1`. Since `S1` is a fixed species, the rate of flow into the pathway is also fixed. If we run the model with the Vmax of the enzyme set to a value of 3.0, the concentration of `S2` reaches a steady state. If however we reduced the Vmax to 2, `S2` increases to infinity.

```
import tellurium as te
import roadrunner

r = te.loada("""
     $S1 -> S2; k1*S1;
     S2 ->; Vm*S2/(Km + S2);

     k1 = 0.4; S1 = 5;
     Vm = 2; Km = 0.5;
""")

m = r.simulate (0, 10, 100)
r.plot()
```

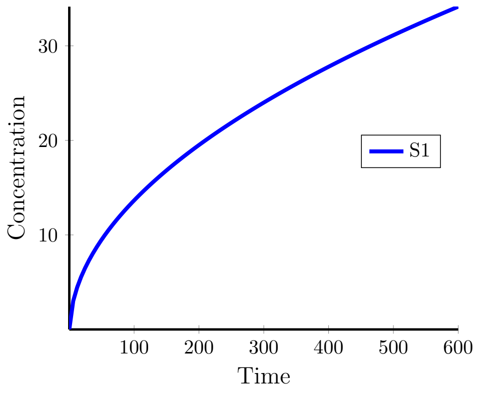

Figure 4: A bad model where one of the species, B, increases indefinitely.

A second issue that can be common with beginners is to observe negative concentrations in a model. There is a real temptation to make the simulator prevent concentrations from moving below zero. This however is the wrong strategy. If some species are going negative this is a indication that there is a problem with the model. Using the simulator to prevent concentrations from going negative simply masks the underlying problem. The example in Figure 5 illustrates the concentration `S1` going negative due to a fixed output rate that is unresponsive to changes in its reactant concentration.

Errors in species or parameter names Many computer languages including SBML are type sensitive, that is names such as 'Glucose' and 'glucose' are considered different species. This results is a very common error in model building where a parameter or a species has been misspelt. For example someone uses 'Km' in a rate law but accidently uses 'km' to initialize the value. Some tools will identify such errors or permit a user to predefine symbols. For example models written using Antimony can predefine all species names and parameters. When done so, any reactant or product names in reactions that have not be predeclared will result in an error reported to the user. If there are many parameters in a model it can be inconvenient to predeclare them. Instead, libRoadRunner will flag any parameters that have not been explicitly initialized and report this as an error to the user. As an example, the following script will result in a 'parameter not initialized error':

```
import tellurium as te
import roadrunner

r = te.loada("""
     $S1 -> S2; k*S1;
     S2 -> S3; vo;

     vo = 15; k = 0.4;
     S1 = 5; S2 = 50
""")

m = r.simulate (0, 10, 100)
r.plot()
```

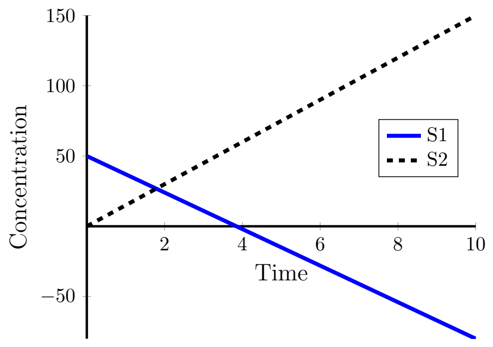

Figure 5: A simple model exhibiting unrealistic negative concentrations.

```
import tellurium as te

r = te.loada("""
     S1 -> S2; k1*S1;

     // This is an error
     k2 = 0.4;
""")
```

Commenting out sections of a model The most powerful tool to debug a model is to comment out sections until the model starts to behave in a more realistic manner. The problem can then be located in the commented section by making finer adjustments to the commented blocks until one or more parts of the model are found wanting.

Check for unusual behavior If commenting is not possible (for example in a GUI based tool) another useful trick is to look at the reaction rates and rates of change at time zero, that is v_i and dx/dt. This can identify problems with the rate laws and/or initialized values. For example reaction rates that appears to be very large or even marked as infinity or NAN (not a number) are a clear sign that the rate law is highly suspect. Looking at the rate of change will yield similar clues. If a reaction is identified in this way it can be isolated by commenting out all other sections and studying it in isolation. In tellurium, two shortcuts are provided to obtain the reaction rates and rates of change information and can provide a rapid snapshot of the state of the model. These include `rv()` and `dv()` respectively. Another simple problem is where a boundary species is drawn down to zero concentration and the pathway stops working. This is likely due to mistakenly categorizing the species as a floating species rather than a fixed one.

Double checking with other simulators If all else fails run the model on a different simulator to eliminate any issues with the simulator software itself. This will require export of the model in SBML or CellML format. This is also a good test for reproducibility of the model.

6 Bifurcation Analysis

Another useful feature of Tellurium is the included bifurcation analysis library. Bifurcation analysis enables qualitative changes in model behavior to be studied as a function of a model parameter.

Such qualitative changes can include bistabiliy and oscillatory behavior ([56, 57]). Tellurium supports bifurcation analysis through the AUTO2000 library ([58]). The feature is implemented as a plugin for libRoadRunner, which allows it to directly access the simulation engine and perform computations without the overhead of a cross-language API. This also means that the bifurcation tool can be used outside of Python and hosted by other tools.

Tellurium's bifurcation plugin is designed to automatically compute a bifurcation in parameter space and plot a bifurcation diagram without user intervention. The user specifies a model parameter as the basis for the analysis. The plugin will then automatically scan a user-specified range of parameter values. If at some point the system changes to an alternate stationary state, the bifurcation is recorded and scanning continues. Figure 6 illustrates a number of bifurcation changes in a model of the embryonic stem cell switch ([59]) and is generated using Tellurium. For models where the stoichiometry matrix is not full rank (e.g. signaling pathways), libRoadRunner creates the appropriately reduced model to avoid numerical errors ([60]).

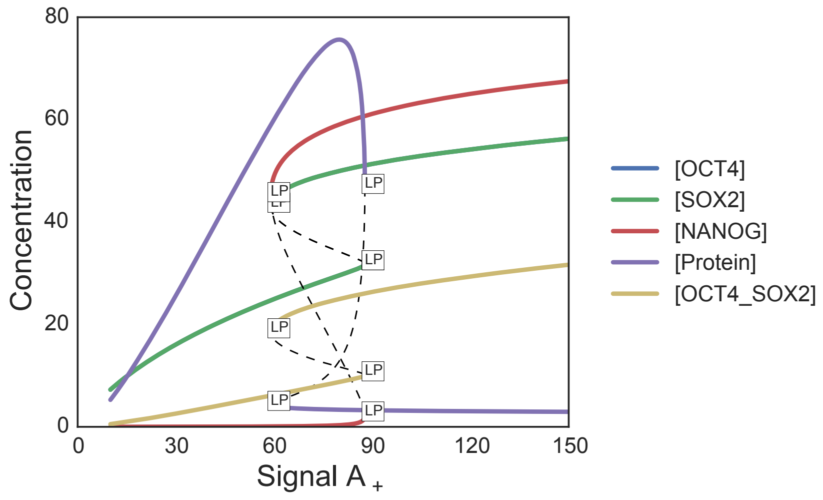

Figure 6: Bifurcation analysis applied to a model of an embryonic stem cell switch. The label LP represents a fold or turning point bifurcation.

7 Acknowledgements

This work was supported by the National Institute of General Medical Sciences of the National Institutes of Health under award numbers R01-GM081070 and through the generous support of the National science Foundation, Biological Infrastructure award #1355909 and Molecular and Cellular Bioscience award #1158573.

References

[1] S. Wright. 'Physiological and Evolutionary Theories of Dominance'. *The American Naturalist*, **68**:24–53, 1934.

[2] J. J. Tyson, K. Chen, B. Novak. 'Network Dynamics And Cell Physiology'. *Nature Reviews Molecular Cell Biology*, **2**:908–916, 2001.

[3] S. Neves, R. Iyengar. 'Modeling of signaling networks'. *BioEssays*, **24**(12):1110–1117, 2002.

[4] N. Geva-Zatorsky, N. Rosenfeld, S. Itzkovitz, R. Milo, A. Sigal, E. Dekel, T. Yarnitzky, Y. Liron, P. Polak, G. Lahav, U. Alon. 'Oscillations and variability in the p53 system'. *Mol Syst Biol*, **2**:2006–2006, 2006.

[5] B. N. Kholodenko. 'Cell-signalling dynamics in time and space'. *Nat Rev Mol Cell Biol*, **7**(3):165–176, 2006.

[6] D. Fell. *Understanding the Control of Metabolism.* Portland Press., London, 1997.

[7] R. Steuer. 'Computational approaches to the topology, stability and dynamics of metabolic networks.' *Phytochemistry*, **68**(16-18):2139–2151, 2007.

[8] B. Ingalls. *Mathematical Modeling in Systems Biology: An Introduction.* MIT Press, 2013. ISBN 9780262018883.

[9] H. M. Sauro. *Systems Biology: An Introduction to Pathway Modeling.* Ambrosius Publishing, 2014.

[10] B. Chance. 'The Kinetics of the Enzyme-Substrate Compound of Peroxidase'. *Journal of Biological Chemistry*, **151**(2):553–577, 1943.

[11] D. Garfinkel. 'A Machine-Independent Language for the Simulation of Complex Chemical and Biochemical Systems.' *Comput. Biomed. Res.*, **2**:31–44, 1968.

[12] J. Burns. 'Steady states of general multi-enzyme networks and their associated properties. Computational approaches.' *FEBS Lett*, **2 Suppl 1**:S30–S33, 1969.

[13] J. A. Burns. *Studies on Complex Enzyme Systems.* Ph.D. thesis, University of Edinburgh, 1971. http://www.sys-bio.org/jim-burns-thesis/.

[14] D. J. M. Park, B. E. Wright. 'METASIM, A General Purpose Metabolic Simulator for Studying Cellular Transformations.' *Comput. Progm. Biomed.*, **3**:10–26, 1973.

[15] E. Klipp, R. Herwig, A. Kowald, C. Wierling, H. Lehrach. *Systems Biology in Practice.* Wiley-VCH Verlag, Weinheim, 2005.

[16] U. Alon. *An Introduction to Systems Biology: Design Principles of Biological Circuits (Chapman & Hall/Crc Mathematical and Computational Biology Series).* Chapman & Hall/CRC, 2006. ISBN 1584886420.

[17] M. Hucka, A. Finney, H. M. Sauro, H. Bolouri, J. C. Doyle, H. Kitano, A. P. Arkin, B. J. Bornstein, D. Bray, A. Cornish-Bowden, A. A. Cuellar, S. Dronov, E. D. Gilles, M. Ginkel, V. Gor, I. I. Goryanin, W. J. Hedley, T. C. Hodgman, J. H. Hofmeyr, P. J. Hunter, N. S. Juty, J. L. Kasberger, A. Kremling, U. Kummer, N. Le NovÃÂre, L. M. Loew, D. Lucio, P. Mendes, E. D. Mjolsness, Y. Nakayama, M. R. Nelson, P. F. Nielsen, T. Sakurada, J. C. Schaff, B. E. Shapiro, T. S. Shimizu, H. D. Spence, J. Stelling, K. Takahashi, M. Tomita, J. Wagner, J. Wang. 'The Systems Biology Markup Language (SBML): A Medium for Representation and Exchange of Biochemical Network Models'. *Bioinformatics*, **19**:524–531, 2003.

[18] W. J. Hedley, N. R. Melanie, D. Bullivant, A. Cuellar, Y. Ge, M. Grehlinger, K. Jim, S. Lett, D. Nickerson, P. Nielsen, H. Yu. 'CellML Specification'. Available via the World Wide Web at `http://www.cellml.org.`, 2001.

[19] H. Kacser, J. A. Burns. 'The Control of Flux'. In D. D. Davies, editor, *Rate Control of Biological Processes*, volume 27 of *Symp. Soc. Exp. Biol.*, pages 65–104. Cambridge University Press, 1973.

[20] M. A. Savageau. 'The behaviour of intact biochemical control systems'. *Curr. Topics Cell. Reg.*, **6**:63–130, 1972.

[21] R. R. Vallabhajosyula, V. Chickarmane, H. M. Sauro. 'Conservation analysis of large biochemical networks'. *Bioinformatics*, **22**(3):346–353, 2006.

[22] H. Sauro, F. Bergmann. 'Standards and ontologies in computational systems biology'. *Essays Biochem*, **45**:211–222, 2008.

[23] G. S, S. HM. In A. Kriete, R. Eils, editors, *Computational Systems Biology: From Molecular Mechanisms to Disease*, chapter Standards, Platforms and Applications, pages 113–181. Academic press, 2013.

[24] A. Dräger, B. Ø. Palsson. 'Improving collaboration by standardization efforts in systems biology'. *Front Bioeng Biotechnol*, **2**:1–20, 2014.

[25] R. Alves, F. Antunes, A. Salvador. 'Tools for kinetic modeling of biochemical networks'. *Nat Biotechnol*, **24**(6):667–672, 2006.

[26] T. Manninen, E. Makiraatikka, A. Ylipaa, A. Pettinen, K. Leinonen, M. L. Linne. 'Discrete stochastic simulation of cell signaling: comparison of computational tools'. *Conf Proc IEEE Eng Med Biol Soc*, **1**:2013–2016, 2006.

[27] I. Vacheva, R. Eils. 'Computational Systems Biology Platforms (Computergestützte Systembiologieplattformen).' *it - Information Technology*, **48**(3):140–147, 2006.

[28] S. H.M, Bergmann. 'Software tools for systems biology'. In E. Liu, D. Lauffenburger, editors, *Systems biomedicine*, chapter 12, pages 289–313. Academic Press, 2010.

[29] R. Gostner, B. Baldacci, M. J. Morine, C. Priami. 'Graphical modeling tools for systems biology'. *ACM Computing Surveys (CSUR)*, **47**(2):16, 2015.

[30] M. L. Blinov, J. R. Faeder, B. Goldstein, W. S. Hlavacek. 'BioNetGen: software for rule-based modeling of signal transduction based on the interactions of molecular domains'. *Bioinformatics*, **20**(17):3289–3291, 2004.

[31] L. A. Harris, J. S. Hogg, J.-J. Tapia, J. A. Sekar, I. Korsunsky, A. Arora, D. Barua, R. P. Sheehan, J. R. Faeder. 'BioNetGen 2.2: advances in rule-based modeling'. *arXiv preprint arXiv:1507.03572*, 2015.

[32] C. F. Lopez, J. L. Muhlich, J. A. Bachman, P. K. Sorger. 'Programming biological models in Python using PySB'. *Molecular systems biology*, **9**(1):646, 2013.

[33] H. Kitano, A. Funahashi, Y. Matsuoka, K. Oda. 'Using process diagrams for the graphical representation of biological networks'. *Nat Biotechnol*, **23**(8):961–966, 2005.

[34] D. Chandran, F. Bergmann, H. Sauro, J. Zhao, P. Holme, A. Re, D. Davide Cora, M. Caselle, A. Mugler, A. Walczak, *et al.* 'TinkerCell: Modular CAD Tool for Synthetic Biology'. *Arxiv preprint arXiv:0907.3976*, 2009.

[35] S. Hoops, S. Sahle, R. Gauges, C. Lee, J. Pahle, N. Simus, M. Singhal, L. Xu, P. Mendes, U. Kummer. 'COPASI–a COmplex PAthway SImulator'. *Bioinformatics*, **22**(24):3067–3074, 2006.

[36] C. J. Myers, N. Barker, K. Jones, H. Kuwahara, C. Madsen, N.-P. D. Nguyen. 'iBioSim: a tool for the analysis and design of genetic circuits'. *Bioinformatics*, **25**(21):2848–2849, 2009.

[37] E. Butterworth, B. E. Jardine, G. M. Raymond, M. L. Neal, J. B. Bassingthwaighte. 'JSim, an open-source modeling system for data analysis'. *F1000Research*, **2**, 2014.

[38] M. Vass, N. Allen, C. A. Shaffer, N. Ramakrishnan, L. T. Watson, J. J. Tyson. 'the JigCell model builder and run manager'. *Bioinformatics*, **20**(18):3680–3681, 2004.

[39] B. G. Olivier, J. M. Rohwer, J. H. Hofmeyr. 'Modelling cellular systems with PySCeS.' *Bioinformatics*, **21**:560–1, 2005.

[40] H. Kacser, J. Burns, D. Fell. 'The control of flux'. *Biochemical Society Transactions*, **23**(2):341–366, 1995.

[41] B. E. Shapiro, M. Hucka, A. Finney, J. Doyle. 'MathSBML: a package for manipulating SBML-based biological models'. *Bioinformatics*, **20**(16):2829–2831, 2004.

[42] I. Moraru, J. C. Schaff, B. M. Slepchenko, L. M. Lowe. 'The Virtual Cell: An Integrated Modeling Environment for Experimental and Computational Cell Biology'. *Ann NY Acad Sci*, **971**(1):595–596, 2002.

[43] A. Finney, M. Hucka. 'Systems biology markup language: Level 2 and beyond'. *Biochemical Society Transactions*, **31**(6):1472–1473, 2003.

[44] M. J. Schilstra, L. Li, J. Matthews, A. Finney, M. Hucka, N. Le Novère. 'CellML2SBML: conversion of CellML into SBML'. *Bioinformatics*, **22**(8):1018–1020, 2006.

[45] C. M. Lloyd, M. D. Halstead, P. F. Nielsen. 'CellML: its future, present and past.' *Prog Biophys Mol Biol.*, **85**:433–50, 2004.

[46] J. S. Luciano, R. D. Stevens. 'e-Science and biological pathway semantics'. *BMC Bioinformatics*, **8 Suppl 3**:8 Suppl 3: S3, 2007.

[47] F. T. Bergmann, R. Adams, S. Moodie, J. Cooper, M. Glont, M. Golebiewski, M. Hucka, C. Laibe, A. K. Miller, D. P. Nickerson, *et al.* 'COMBINE archive and OMEX format: one file to share all information to reproduce a modeling project'. *BMC bioinformatics*, **15**(1):369, 2014.

[48] E. T. Somogyi, J.-M. Bouteiller, J. A. Glazier, M. Knig, J. K. Medley, M. H. Swat, H. M. Sauro. 'libRoadRunner: a high performance SBML simulation and analysis library.' *Bioinformatics*, **31**(20):3315–3321, 2015.

[49] H. M. Sauro. *Enzyme Kinetics for Systems Biology.* Ambrosius Publishing. 2nd Edition, 2012.

[50] J. G. Reich, E. E. Selkov. *Energy metabolism of the cell.* Academic Press, London, 1981.

[51] J.-H. Hofmeyr, H. Kacser, K. J. van der Merwe. 'Metabolic Control Analysis of Moiety-Conserved Cycles'. *Eur. J. Biochem.*, **155**:631–641, 1986.

[52] H. M. Sauro. 'Moiety-conserved cycles and metabolic control analysis: problems in sequestration and metabolic channelling'. *BioSystems*, **33**:15–28, 1994.

[53] H. M. Sauro. 'Jarnac: A System for Interactive Metabolic Analysis'. In J.-H. S. Hofmeyr, J. M. Rohwer, J. L. Snoep, editors, *Animating the Cellular Map: Proceedings of the 9th International Meeting on BioThermoKinetics.* Stellenbosch University Press, 2000. ISBN ISBN 0-7972-0776-7.

[54] F. T. Bergmann, R. R. Vallabhajosyula, H. M. Sauro. 'Computational Tools for Modeling Protein Networks'. *Current Proteomics*, **3**:181–197(17), October 2006.

[55] F. Bergmann, H. Sauro. 'SBW-a modular framework for systems biology'. pages 1637–1645, 2006.

[56] D. Angeli, J. E. Ferrell, E. D. Sontag. 'Detection of multistability, bifurcations, and hysteresis in a large class of biological positive-feedback systems'. *Proc. Natl. Acad. Sci. USA*, **101**(7):1822–1827, 2004.

[57] B. Ermentrout. *Simulating, Analyzing, and Animating Dynamical Systems: A Guide to Xppaut for Researchers and Students.* Society for Industrial Mathematics; 1st edition, 2002.

[58] E. J. Doedel. 'Auto: a program for the automatic bifurcation analysis of autonomous systems.' In *Proc. Manitoba Conf. Num. Math. Comput., 10th, Winnipeg, Canada.* 1981. [Congressus Numeratium, 30:265–284].

[59] V. Chickarmane, C. Troein, U. A. Nuber, H. M. Sauro, C. Peterson. 'Transcriptional dynamics of the embryonic stem cell switch'. *PLoS Comput Biol*, **2**(9), 2006.

[60] F. T. Bergmann, R. R. Vallabhajosyula, H. M. Sauro. 'Computational tools for modeling protein networks'. *Current Proteomics*, **3**(3):181–197, 2006.