HHS Public Access

Author manuscript

Biosystems. Author manuscript; available in PMC 2019 September 01.

Published in final edited form as:

Biosystems. 2018 September ; 171: 74–79. doi:10.1016/j.biosystems.2018.07.006.

Tellurium: An Extensible Python-based Modeling Environment for Systems and Synthetic Biology

Kiri Choi[a], **J Kyle Medley**[a], **Matthias König**[b], **Kaylene Stocking**[a], **Lucian Smith**[a], **Stanley Gu**[a], and **Herbert M. Sauro**[a,*]

[a]Department of Bioengineering, University of Washington, William H. Foege Building, Box 355061, Seattle, WA, USA, 98195

[b]Institute for Biology, Institute for Theoretical Biology, Humboldt University, Berlin, Germany

Abstract

Here we present Tellurium, a Python-based environment for model building, simulation, and analysis that facilitates reproducibility of models in systems and synthetic biology. Tellurium is a modular, cross-platform, and open-source simulation environment composed of multiple libraries, plugins, and specialized modules and methods. Tellurium is a self-contained modeling platform which comes with a fully configured Python distribution. Two interfaces are provided, one based on the Spyder IDE which has an accessible user interface akin to MATLAB and a second based on the Jupyter Notebook, which is a format that contains live code, equations, visualizations, and narrative text. Tellurium uses libRoadRunner as the default SBML simulation engine which supports deterministic simulations, stochastic simulations, and steady-state analyses. Tellurium also includes Antimony, a human-readable model definition language which can be converted to and from SBML. Other standard Python scientific libraries such as NumPy, SciPy, and matplotlib are included by default. Additionally, we include several user-friendly plugins and advanced modules for a wide-variety of applications, ranging from complex algorithms for bifurcation analysis to multidimensional parameter scanning. By combining multiple libraries, plugins, and modules into a single package, Tellurium provides a unified but extensible solution for biological modeling and analysis for both novices and experts. Availability: tellurium.analogmachine.org

Keywords

Simulation; SBML; Software; Systems Biology

*Corresponding author: kirichoi@uw.edu (Kiri Choi), medjk@comcast.net (J Kyle Medley), koenigmx@hu-berlin.de (Matthias Ko¨nig), viola.sox@gmail.com (Kaylene Stocking), lucianoelsmitho@gmail.com (Lucian Smith), stanleygu@gmail.com (Stanley Gu), hsauro@u.washington.edu (Herbert M. Sauro).

Authors contributions

HMS conceived the idea, helped with documentation and debugging; KC developed the tool, maintained Windows distribution, wrote the documentation, example code, website, and the article; KM developed the tool, maintained Mac and Linux distributions, wrote the documentation and website; MK developed the tool, wrote the documentation and the article; KS developed the tool and wrote the documentation; LS developed the Antimony and phraSED-ML Languages; GS developed the tool.

Background

Python has proven to be a very popular language for scientific computing and data science. The ease of learning and use, coupled with the open-source nature of the language has made it an ideal platform for scientific computations. The systems and synthetic biology community have shown support for Python through the development of a variety of simulation tools. These include PySCeS [22] with a focus on simulation via differential equations, structural analysis, and metabolic control analysis; SloppyCell [21], with a focus on model fitting and calculating the resulting uncertainties; pySB [20], with a focus on rule-based reaction models; or COBRApy [14], with a focus on constraint-based modeling. However, as can be observed from this brief overview, most tools are limited in their scope and focus on a specific set of functionalities. Additionally, the installation process of systems biology software can often be quite cumbersome, requiring users to follow multiple and often fragile steps for proper configuration. This can be problematic for both novices and experts in the field.

Another critical issue in systems and synthetic biology is ensuring exchangeability and reproducibility of models and simulation setups. Over the past few years, the community has developed a variety of standards to accurately capture models and simulation experiments. These standards include the Systems Biology Markup Language (SBML) [17], which encodes the model, Simulation Experiment Description Markup Language (SEDML) [30], which encodes the simulation setup, and the COMBINE archive [5], which is the collection of files that represent the full description of the model and associated simulation experiments. For synthetic biology, the community has developed the Synthetic Biology Open Language (SBOL) to describe synthetic designs [2]. Many of the existing tools support at least part of these standards. For example, PySCeS supports SBML and a large portion of SED-ML. SloppyCell also supports SBML, as does COBRApy. pySB offers some support for reading and writing SBML models. However, none of the Python tools described here supports the full set of standards discussed above.

Therefore, our goal in developing Tellurium was to design a general platform with broader scope by combining a large variety of third-party tools while supporting various standards to ensure reproducibility. Furthermore, the installation process should be as simple as possible to make our tool easily accessible.

The core philosophy behind Tellurium is to provide a high-performance platform accessible to both novices and experts. We bring together a wide variety of libraries and tools for researchers in systems and synthetic biology. Tellurium is distributed using one-click installers so the installation process is extremely simple. Tellurium provides a convenient one-stop solution for many of the needs of the community, which is especially helpful for novices who do not wish to deal with the complexities of manual configuration of the various tools we distribute. For systems biology modeling, Tellurium supports various modeling standards including SBML, SED-ML, the COMBINE archive, and SBOL. In addition, we distribute libRoadRunner [28] for simulation, AUTO2000 [13] for bifurcation analysis, and Antimony [27], phraSED-ML [12], as well as SimpleSBML [9] for streamlined model creation and modification. Along with the tools distributed with

Tellurium, we provide a simple method for users to install additional Python packages, making Tellurium highly extensible.

Implementation

Tellurium is implemented in a mixture of C, C++, and Python. The software can be roughly partitioned into three functional pillars: i) standards support; ii) modeling; and iii) general utilities (Figure 1).

Support for standards in systems and synthetic biology is included in Tellurium via the respective libraries such as libSBML [8], libSEDML [4], libCOMBINE [6], libSBOL, and basic support for CellML [16] via Antimony. Many of these libraries come from third-party developers and some have been augmented for Tellurium to make them easier to use. For example, SimpleSBML simplifies model building instead of requiring users to use low-level methods in libSBML. Tellurium provides extensive layers to libSBML and libCOMBINE to simplify the process of generating COMBINE archives. We use COMBINE archives to facilitate simulation reproducibility.

The second pillar includes the modeling and numerical support for model design and analysis. Tellurium comes with packages such as Antimony [27] and phraSED-ML [12] which translate model and simulation setup in SBML and SED-ML format to human-readable counterparts. The numerical support includes libRoadRunner which provides a variety of analyses including ordinary differential equation simulation, Gillespie-based stochastic simulation, metabolic control analysis, and structural analysis of networks via libStructural [3].

Another important function included in Tellurium is bifurcation analysis, crucial for understanding models with multiple steady states. This type of analysis can be difficult for a novice to perform, so a wrapper to AUTO2000 is provided which interfaces itself to libRoadRunner. By implementing it as a plugin for libRoadRunner, AUTO2000 can 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. Note that unlike other AUTO2000 implementations, our implementation does not require an external compiler because this task is handled by libRoadRunner.

Finally, to demonstrate the flexibility in a Python ecosystem, we also bundle COBRApy, which is one of the primary constraint-based modeling packages. In addition, common Python packages that are essential in scientific computing are bundled with Tellurium. These include, but are not limited to, SciPy and NumPy (for a large variety of numerical methods), SymPy (for symbolic manipulation), and plotting libraries such as matplotlib and seaborn. Supplementary Table S1 lists short descriptions of the packages discussed in this manuscript.

Tellurium is distributed with two interfaces: The first is Tellurium Spyder, which is based on Spyder IDE and provides a MATLAB-like environment for researchers who are already familiar with editor/console type programming. Spyder IDE is a Python-based development

environment that comes with powerful tools like profiler and static code analysis. Spyder IDE is ideal for modelers and developers who prefer generating and debugging raw Python scripts. For those who prefer notebook-like interfaces, we provide a Jupyter notebook-based version called Tellurium Notebook. Jupyter notebook differs from Spyder IDE as it creates documents containing live code, plots, narrative texts, and equations. Moreover, Jupyter notebooks are interactive, making it ideal for sharing and displaying the work with others. It is also possible to install Tellurium and its dependencies in an existing Python environment through pip. Examples of alternative hosts that have employed Tellurium include PyCharm and Sublime Text.

Applications

In this section, we illustrate several use cases of Tellurium. In particular, we demonstrate various tools included in Tellurium as well as its ability to integrate with other Python packages. We present examples of model building, simulation, and subsequent analysis tasks such as metabolic control analysis, bifurcation analysis, and parameter estimation. All scripts used in this section are available in the Supplementary Materials.

Model Building and Simulation

First, we start with a simple example demonstrating model building and simulation in Tellurium. Models in Tellurium can be defined using Antimony, a human-readable model definition language which directly translates to SBML. Antimony supports a large part of SBML specification including events and assignment rules and can be easily translated to and from SBML. Figure 2 illustrates a model of a simple linear chain involving five floating species and corresponding simulation result.

Metabolic Control Analysis

An important part of the modeling and model analysis process is sensitivity analysis, which provides information about the effect of system parameters and states on the results. A standard approach for sensitivity analysis is metabolic control analysis (MCA). Tellurium calculates the various elasticities and coefficients defined in MCA [19, 24] using libRoadRunner [28]. In addition, there is support for frequency dependent MCA in the form of Bode plots [18]. A number of utilities are provided to make it easier to visualize results from MCA studies. In particular, we provide utilities to help visualize flux control, concentration control, and elasticity profiles using heat maps. Figure 3 shows heatmaps of the distribution of flux and concentration control coefficients in a linear pathway of six reactions (Figure 2).

Bifurcation Analysis

Bifurcation analysis enables qualitative changes in model behavior to be studied as a function of a model parameter. Such qualitative changes include bistability and oscillatory behavior [1, 15]. Tellurium's bifurcation facility 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 bifurcation tool 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 4A illustrates a number of bifurcation changes in a model of the embryonic stem cell switch [11] with Tellurium. For models where the stoichiometry matrix does not have full rank, libRoadRunner creates the appropriately reduced model [7] thus permitting bifurcation analysis of protein signaling networks to be carried out [25, 23].

COMBINE Support

Tellurium supports importing and exporting COMBINE archives through the Antimony and phraSED-ML languages. To demonstrate this feature, we present a simple application using the same model and simulation setup as in the code illustrated in Figure 2. To generate a COMBINE archive, a model and a simulation setup are defined in the Antimony and phraSED-ML language. We then create an inline OMEX by joining the two string blocks.

```
import tempfile, os
import tellurium as te
te.setDefaultPlottingEngine('matplotlib')
antimony_str = '''
model myModel
J1: $Xo -> S1; k1*Xo - k2*S1;
J2: S1 -> S2; k3*S1 - k4*S2;
J3: S2 -> S3; k5*S2 - k6*S3;
J4: S3 -> S4; k7*S3 - k8*S4;
J5: S4 -> S5; k9*S4 - k10*S5;
J6: S5 -> ; k11*S5;
Xo = 1.0; S1 = 0.3; S2 = 0.1;
S3 = 0.2; S4 = 0.1; S5 = 0.2;
k1 = 3.92; k2 = 2.83; k3 = 0.8;
k4 = 0.24; k5 = 0.68; k6 = 0.35;
k7 = 0.82; k8 = 0.47; k9 = 0.37;
k10 = 0.22; k11 = 0.1;
end
'''
phrasedml_str = '''
model1 = model "myModel"
sim1 = simulate uniform(0, 30, 100)
task1 = run sim1 on model1
plot "Figure 1" time vs S1, S2, S3, S4, S5
'''
inline_omex = '\n'.join([antimony_str, phrasedml_str])
```

This setup can be directly executed in Python to check the output using executeInlineOmex, which will be identical to the plot shown in Figure 2.

```
te.executeInlineOmex(inline_omex)
```

Export this setup to a COMBINE archive using exportInlineOmex.

```
wDir = tempfile.mkdtemp(suffix="_omex")
te.exportInlineOmex(inline_omex, os.path.join(wDir, 'archive. omex'))
```

Now import and execute the COMBINE archive that was just exported to check that the output is the same as before.

```
te.executeCombineArchive(os.path.join(wDir, 'archive.omex'))
```

While it is possible to import COMBINE archives through Python functions, both Tellurium Spyder and Tellurium Notebooks provide simple GUI-based plug-ins to open a COMBINE archive as well. Detailed description on support for creating reproducible models in Tellurium will be discussed in a separate publication due to its broad scope.

Parameter Estimation

Parameter estimation is a common step in developing a model where the model is fitted to experimental data. Since Tellurium is based on Python, users can use the various optimization packages available in Python. More-over, Tellurium provides an environment where parameter estimation routines can be easily customized to deal with almost any fitting problem. To demonstrate Tellurium's abilities in parameter estimation, we used a model of the central carbon metabolism of *E. coli* originally published by Chassagnole *et al.* [10] and later reformulated to be used as a part of benchmark suite for parameter estimation by Villaverde *et al.* [29]. The model is composed of 18 species and 48 reactions with 116 parameters to fit. Experimental data was supplied by the original authors, which consists of 110 time-course data points spread over 9 different metabolites. The reason why we choose this particular model is because 1) we have reference results to compare against, 2) the model is based on measured experimental data, and 3) the model presents a challenging parameter estimation problem where the reported 'optimized' results still does not fit well. Therefore, in this application, our goal will be to get a fit comparable to that obtained by Villaverde *et al.* [29].

The model presents a relatively large number of parameters to fit and many standard local optimization methods fail. Instead, a global optimizer is used to find a proper set of parameters. Here, we use the differential evolution optimizer supplied by the SciPy package. Figure 4B shows the result of parameter estimation on Tellurium, which is similar to the fit reported in the benchmark [29] and better than that of the original paper [10]. To compare the fit, we use cumulative normalized root-mean-squared error ($\sum$NRMSE), as was done by Villaverde *et al.* [29]. Root mean squared error measures the average of differences between observed and predicted values (error), and is given by Equation 1.

$$\mathrm{RMSE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}\left(y_i - \hat{y}_i\right)^2} \quad (1)$$

Here, N is the total number of the sample, y_i is the observed value, and $\hat{y}_i$ is the predicted value. This value has been normalized against the observables by dividing by the difference between the maximum and minimum values of observables as shown in Equation 2.

$$\mathrm{NRMSE} = \frac{\mathrm{RMSE}}{\max(y_i)\text{-}\min(y_i)} \quad (2)$$

Our parameter optimization run results in $\sum\mathrm{NRMSE} \approx 2.29$ which is similar to the value reported by Villaverde *et al.* ($\sum\mathrm{NRMSE} \approx 2.49$) [29] and better than the original parameterization given by Chassagnole *et al.* ($\sum\mathrm{NRMSE} \approx 3.61$) [10]. A plot of the residuals is provided as Supplementary Figure S1. Fitted parameters obtained from Tellurium are available in the Supplementary Materials.

A single run of parameter estimation using differential evolution took about 4.5 hours on a single core of Intel i7 4770 machine running at 3.4 GHz with 8GB RAM. Approximate standard errors on the fitted parameters can be obtained from the Hessian. For more accurate estimates it would be possible to use Monte Carlo or Profile Likelihood methods [26]. For the scope of this paper, we omit this step, but in the future, we will be supporting massively parallelized workloads through commercial cloud services that a user might be subscribed to.

Conclusion

With Tellurium, we provide the systems and synthetic biology community with an extensible Python-based modeling environment. We have endeavored to build a platform for collaboration by basing Tellurium on extensible and open architectures such as Spyder IDE and Jupyter Notebook. Our tools are available under Open Source Initiative (OSI)-approved open source licenses. As a result, our users have the freedom to apply further customizations to Tellurium. Pervasive support for systems biology standards enables models created by Tellurium to be stored, reused, and modified reliably by other software tools.

Availability

Installers for Tellurium Spyder are available for Microsoft Windows and Mac OS X. Jupyter Notebook versions of Tellurium (Tellurium Notebook) are available for Windows, Mac OS X, Debian and RedHat Linux distros. The Tellurium package is also available through PyPI. Binaries, documentation, and full source code are available at http://tellurium.analogmachine.org and https://github.com/sys-bio/tellurium. Tellurium is licensed

under the Apache License Version 2.0. Scripts used for applications section are available in the Supplementary Materials.

Supplementary Material

Refer to Web version on PubMed Central for supplementary material.

Acknowledgements

This work was partially supported by the National Institute of General Medical Sciences of the National Institutes of Health under awards R01-GM081070, R01-GM123032 and the Federal Ministry of Education and Research (BMBF, Germany) within the research network Systems Medicine of the Liver (LiSyM) [grant number 031L0054]. The SBOL work was supported by the National Science Foundation grant number DBI-1355909. The content is solely the responsibility of the authors and does not necessarily represent the official views of the National Institutes of Health or National Science Foundation. We wish to thank Joseph Hellerstein for his help and guidance in fixing parameter estimation issues.

References

[1]. Angeli D, Ferrell JE, Sontag ED, 2004 Detection of multistability, bifurcations, and hysteresis in a large class of biological positive-feedback systems. Proc. Natl. Acad. Sci. USA 101 (7), 1822–1827. URL http://www.pnas.org/content/101/7/1822.abstract [PubMed: 14766974]

[2]. Beal J, Cox RS, Gruünberg R, McLaughlin J, Nguyen T, Bartley B, Bissell M, Choi K, Clancy K, Macklin C, et al., 2016 Synthetic biology open language (sbol) version 2.1. 0. Journal of Integrative Bioinformatics 13 (3), 30–132.

[3]. Bedaso Y, Bergmann FT, Choi K, Medley K, Sauro HM, 2018 A portable structural analysis library for reaction networks. Biosystems 169–170, 20 – 25. URL http://www.sciencedirect.com/science/article/pii/ S0303264718300145

[4]. Bergmann F, Nickerson D, Nol V, Medley K, Sep. 2017 fbergmann/libsedml: libsedml 0.4.3 URL 10.5281/zenodo.998943

[5]. Bergmann FT, Adams R, Moodie S, Cooper J, Glont M, Golebiewski M, Hucka M, Laibe C, Miller AK, Nickerson DP, Olivier BG, Rodriguez N, Sauro HM, Scharm M, Soiland-Reyes S, Waltemath D, Yvon F, Le Nov`ere N, 12 2014 Combine archive and omex format: one file to share all information to reproduce a modeling project. BMC bioinformatics 15, 369 URL http://www.ncbi.nlm.nih.gov/pubmed/25494900 [PubMed: 25494900]

[6]. Bergmann FT, Keating SM, Sep. 2016 libCombine: a C++ API library supporting the COMBINE Archive URL 10.5281/zenodo.154158

[7]. Bergmann FT, Vallabhajosyula RR, Sauro HM, 2006 Computational tools for modeling protein networks. Current Proteomics 3 (3), 181–197.

[8]. Bornstein B, Keating S, Jouraku A, Hucka M, 2008 LibSBML: an API Library for SBML. Bioinformatics 24 (6), 880. [PubMed: 18252737]

[9]. Cannistra C, Medley K, Sauro H, 2015 Simplesbml: A python package for creating and editing sbml models. bioRxiv, 030312.

[10]. Chassagnole C, Noisommit-Rizzi N, Schmid JW, Mauch K, Reuss M, 2002 Dynamic modeling of the central carbon metabolism of escherichia coli. Biotechnology and bioengineering 79 (1), 53–73. [PubMed: 17590932]

[11]. Chickarmane V, Troein C, Nuber UA, Sauro HM, Peterson C, 9 2006 Transcriptional dynamics of the embryonic stem cell switch. PLoS Comput. Biol 2 (9), e123 URL http://dx.plos.org/10.1371%2Fjournal.pcbi.0020123 [PubMed: 16978048]

[12]. Choi K, Smith LP, Medley JK, Sauro HM, 2016 phrased-ml: A paraphrased, human-readable adaptation of sed-ml. Journal of bioinformatics and computational biology 14 (06), 1650035. [PubMed: 27774871]

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

[14]. Ebrahim A, Lerman JA, Palsson BO, Hyduke DR, 2013 Cobrapy: Constraints-based reconstruction and analysis for python. BMC systems biology 7 (1), 74.

[15]. Ermentrout GB, Terman DH, 2010 Mathematical foundations of neuroscience. Interdisciplinary Applied Mathematics (Book 35) New York Springer, xvi.

[16]. Hedley WJ, Melanie NR, Bullivant DP, Nielson PF, 2001 A Short Introduction to CellML. Phil. Trans. Roy. Soc. London A 359, 1073–1089.

[17]. Hucka M, Finney A, Sauro HM, Bolouri H, 2001 Systems Biology Markup Language (SBML) Level 1: Structures and Facilities for Basic Model Definitions Available via the World Wide Web at http://www.cds.caltech.edu/erato.

[18]. Ingalls BP, 2004 A Frequency Domain Approach to Sensitivity Anal- ysis of Biochemical Systems. Journal of Physical Chemistry B 108, 1143–1152.

[19]. Kacser H, Burns JA, 1973 The Control of Flux. In: Davies DD (Ed.), Rate Control of Biological Processes Vol. 27 of Symp. Soc. Exp. Biol. Cambridge University Press, pp. 65–104.

[20]. Lopez CF, Muhlich JL, Bachman JA, Sorger PK, 2013 Programming biological models in python using pysb. Molecular systems biology 9 (1), 646. [PubMed: 23423320]

[21]. Myers CR, Gutenkunst RN, Sethna JP, 2007 Python Unleashed on Systems Biology. Computing in Science and Engg 9 (3), 34–37.

[22]. Olivier BG, Rohwer JM, Hofmeyr JH, 2005 Modelling cellular systems with PySCeS. Bioinformatics 21, 560–1. [PubMed: 15454409]

[23]. Sauro HM, 2014 Systems Biology: An Introduction to Pathway Modeling Ambrosius Publishing, Seattle.

[24]. Sauro HM, 2017 Control and regulation of pathways via negative feedback. Journal of The Royal Society Interface 14 (127).

[25]. Sauro HM, Ingalls B, 4 2004 Conservation analysis in biochemical networks: computational issues for software writers. Biophys Chem 109 (1), 1–15. [PubMed: 15059656]

[26]. Schaber J, Klipp E, 2011 Model-based inference of biochemical parameters and dynamic properties of microbial signal transduction networks. Current Opinion in Biotechnology 22 (1), 109–116. [PubMed: 20970318]

[27]. Smith LP, Bergmann FT, Chandran D, Sauro HM, 2009 Antimony: a modular model definition language. Bioinformatics 25 (18), 2452–2454. [PubMed: 19578039]

[28]. Somogyi ET, Bouteiller J-M, Glazier JA, K¨onig M, Medley JK, Swat MH, Sauro HM, 2015 libroadrunner: a high performance sbml simulation and analysis library. Bioinformatics 31 (20), 3315–3321. [PubMed: 26085503]

[29]. Villaverde AF, Henriques D, Smallbone K, Bongard S, Schmid J, Cicin-Sain D, Crombach A, Saez-Rodriguez J, Mauch K, Balsa- Canto E, et al., 2015 Biopredyn-bench: a suite of benchmark problems for dynamic modelling in systems biology. BMC systems biology 9 (1), 8. [PubMed: 25880925]

[30]. Waltemath D, Adams R, Bergmann FT, Hucka M, Kolpakov F, Miller AK, Moraru II, Nickerson D, Sahle S, Snoep JL, et al., 2011 Reproducible computational biology experiments with sed-ml-the simulation experiment description markup language. BMC systems biology 5 (1), 1. [PubMed: 21194489]

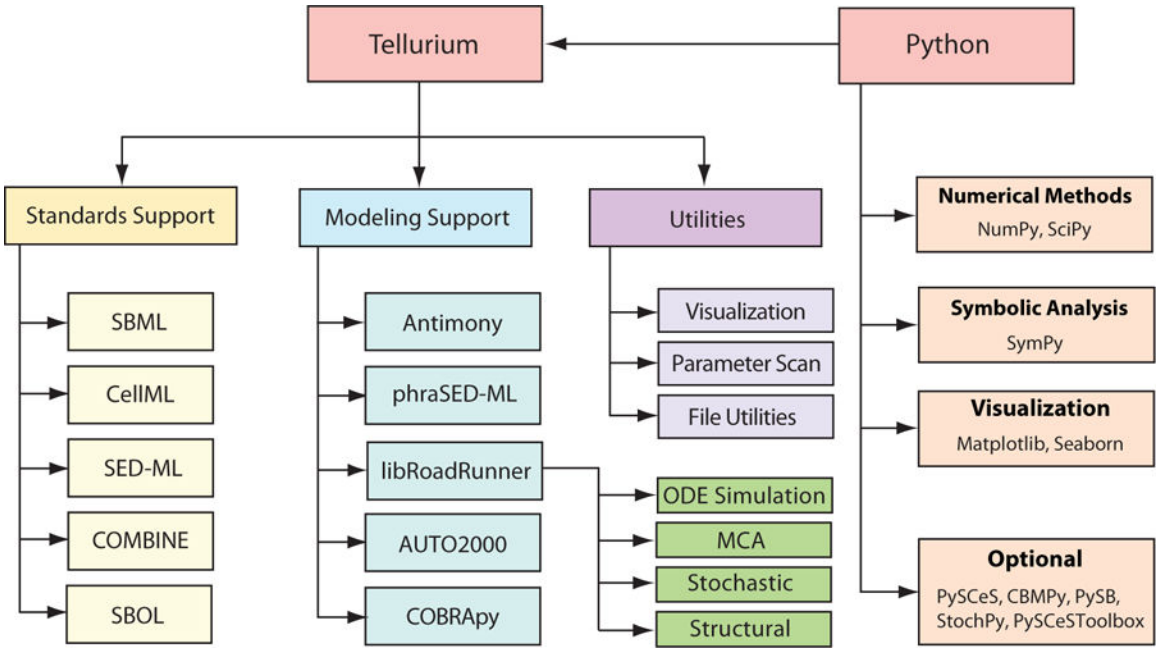


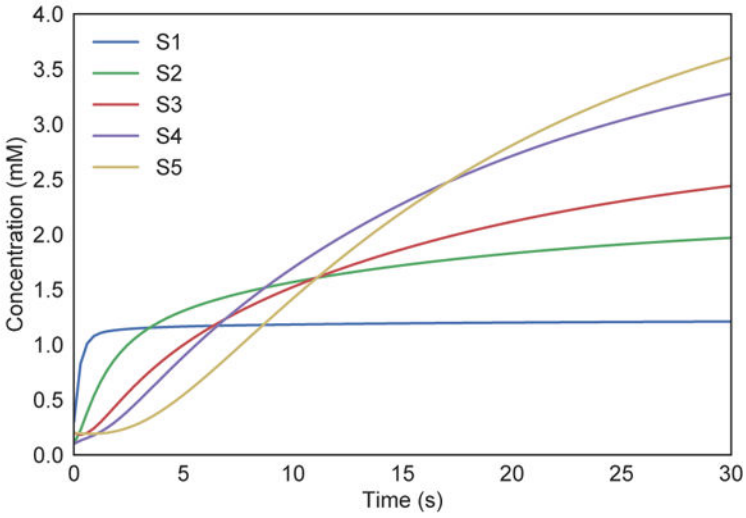
Figure 1:

Overview of Tellurium. Tellurium is composed of three distinct functional pillars including standards support, modeling support, and utilities. Several third-party Python packages come with Tellurium and additional packages can be installed if needed.

```
import tellurium as te

r = te.loada("""
J1: $Xo -> S1; k1*Xo - k2*S1;
J2: S1 -> S2; k3*S1 - k4*S2;
J3: S2 -> S3; k5*S2 - k6*S3;
J4: S3 -> S4; k7*S3 - k8*S4;
J5: S4 -> S5; k9*S4 - k10*S5;
J6: S5 -> ; k11*S5;

Xo = 1.0; S1 = 0.3; S2 = 0.1;
S3 = 0.2; S4 = 0.1; S5 = 0.2;
k1 = 3.92; k2 = 2.83; k3 = 0.8;
k4 = 0.24; k5 = 0.68; k6 = 0.35;
k7 = 0.82; k8 = 0.47; k9 = 0.37;
k10 = 0.22; k11 = 0.1;
""")
result = r.simulate(0, 30, 100)
r.plot()
```



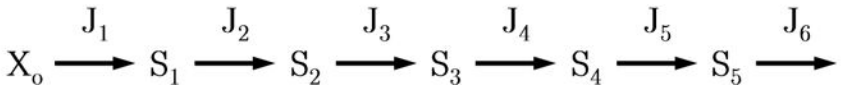


Figure 2:

A simple linear chain model with five floating species written in Antimony language and corresponding simulation result. The diagram below illustrates the model. Species X_o is a boundary species (fixed) and each reaction is modeled using reversible mass-action kinetics. J_i is used to label each reaction.

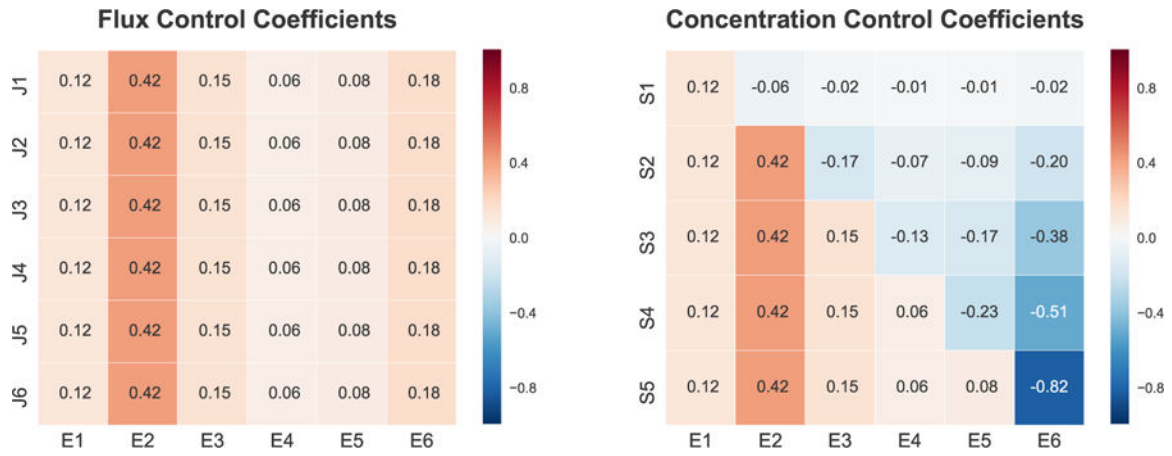


Figure 3:

Two heatmaps showing the flux and concentration control coefficients for a linear reaction chain of six reactions and five floating species illustrated in Figure 2. E_i is the enzyme level for reaction i, and J_i is the flux through reaction i. S_i is the substrate label. Red indicates positive values and blue indicates negative values. For example, reaction step six, E_6, has a strong negative influence on species S_5.

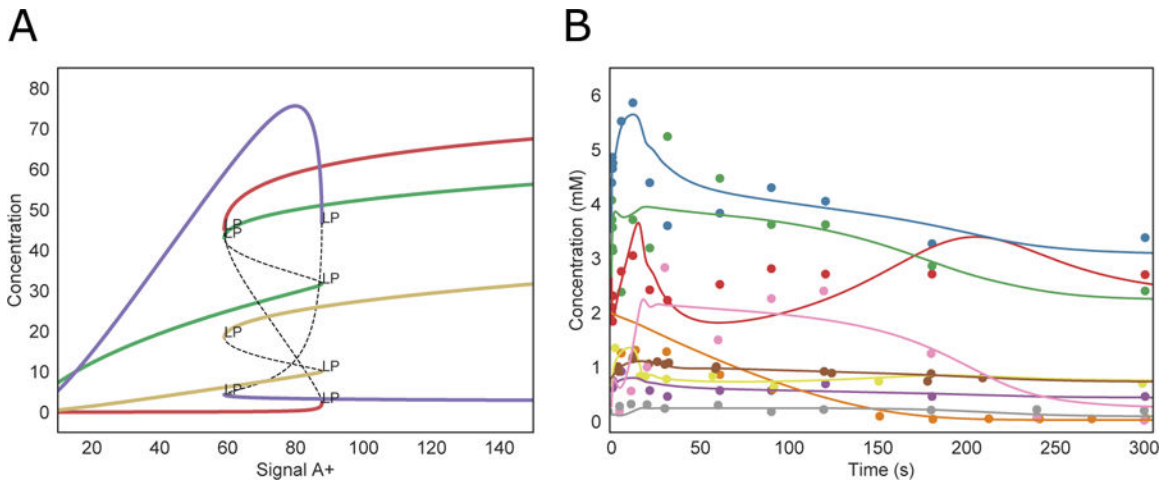


Figure 4:

Plots depicting applications of Tellurium. A) Bifurcation analysis applied to a model of an embryonic stem cell switch. The label LP represents a fold or turning point bifurcation. Blue, green, red, and yellow indicate transcription factors OCT4, SOX2, NANOG, and OCT4–SOX2 heterodimer respectively. Blue (OCT4) trace is covered by the green trace (SOX4). B) Central carbon metabolism model of *E. coli* fitted against experimental data of 9 metabolites. Lines represent simulated data using fitted parameters and dots represent the experimental data. Red, blue, green, purple, orange, yellow, brown, pink, and gray traces and dots corresponds to pep, g6p, pyr, f6p, glcex, g1p, pg, fdp, and gap, respectively.