BioSystems 171 (2018) 74–79

Contents lists available at ScienceDirect

BioSystems

journal homepage: www.elsevier.com/locate/biosystems

Short Communication

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], Herbert M. Sauro[a,*]

[a] *Department of Bioengineering, University of Washington, William H. Foege Building, Box 355061, Seattle, WA 98195, USA*
[b] *Institute for Biology, Institute for Theoretical Biology, Humboldt University, Berlin, Germany*

ARTICLE INFO

Keywords:
Simulation
SBML
Software
Systems biology

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.

1. 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 (Olivier et al., 2005) with a focus on simulation via differential equations, structural analysis, and metabolic control analysis; SloppyCell (Myers et al., 2007), with a focus on model fitting and calculating the resulting uncertainties; pySB (Lopez et al., 2013), with a focus on rule-based reaction models; or COBRApy (Ebrahim et al., 2013), 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) (Hucka et al., 2001), which encodes the model, Simulation Experiment Description Markup Language (SED-ML) (Waltemath et al., 2011), which encodes the simulation setup, and the COMBINE archive (Bergmann et al., 2014), 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 (Beal et al., 2016). 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

* Corresponding author.
E-mail addresses: kirichoi@uw.edu (K. Choi), medjk@comcast.net (J.K. Medley), koenigmx@hu-berlin.de (M. König), viola.sox@gmail.com (K. Stocking), lucianoelsmitho@gmail.com (L. Smith), stanleygu@gmail.com (S. Gu), hsauro@u.washington.edu (H.M. Sauro).

https://doi.org/10.1016/j.biosystems.2018.07.006
Received 31 March 2018; Received in revised form 18 July 2018; Accepted 18 July 2018
Available online 25 July 2018

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 (Somogyi et al., 2015) for simulation, AUTO2000 (Doedel, 1981) for bifurcation analysis, and Antimony (Smith et al., 2009), phraSED-ML (Choi et al., 2016), as well as SimpleSBML (Cannistra et al., 2015) 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.

2. 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 (Fig. 1).

Support for standards in systems and synthetic biology is included in Tellurium via the respective libraries such as libSBML (Bornstein et al., 2008), libSEDML (Bergmann et al., 2017), libCOMBINE (Bergmann and Keating, 2016), libSBOL, and basic support for CellML (Hedley et al., 2001) 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 (Smith et al., 2009) and phraSED-ML (Choi et al., 2016) 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 (Bedaso et al., 2018).

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.

3. 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

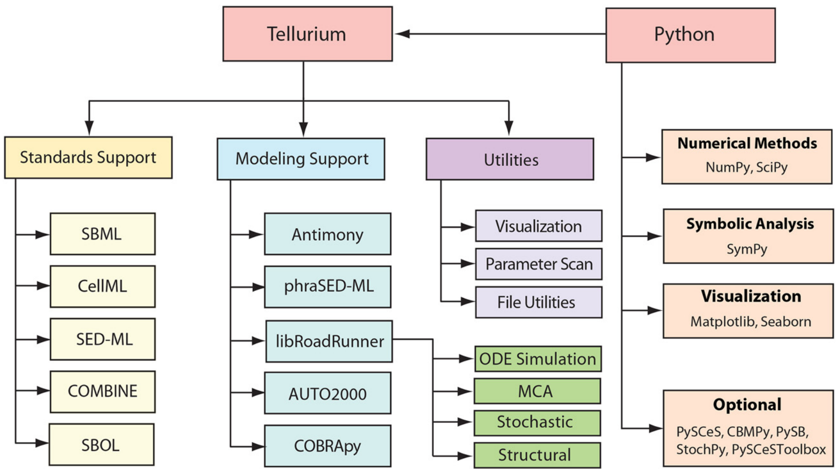

Fig. 1. Overview of Tellurium. Tellurium is composed of three distinct functional pillars including standards support, modeling support, and utilities. Several third-party Python packages come with Tellurium and additional packages can be installed if needed.