Synthetic Biology, 2021, 6(1): ysab001

doi: 10.1093/synbio/ysab001
Advance Access Publication Date: 22 February 2021
Software

SynBiopython: an open-source software library for *Synthetic Biology*

Jing Wui Yeoh[1], Neil Swainston[2,*], Peter Vegh[3], Valentin Zulkower[3], Pablo Carbonell [4,5], Maciej B. Holowko [6], Gopal Peddinti[7], and Chueh Loo Poh[1,*]

[1]NUS Synthetic Biology for Clinical and Technological Innovation (SynCTI), Life Sciences Institute, National University of Singapore, Singapore, Singapore, [2]Institute of Systems, Molecular and Integrative Biology, University of Liverpool, Liverpool, UK, [3]Edinburgh Genome Foundry, University of Edinburgh, Edinburgh, UK, [4]Instituto Universitario de Automática e Informática Industrial, Universitat Politècnica de València, Valencia, Spain, [5]Manchester Synthetic Biology Research Centre for Fine and Speciality Chemicals (SYNBIOCHEM), Manchester Institute of Biotechnology, The University of Manchester, Manchester, UK, [6]CSIRO Synthetic Biology Future Science Platform, Canberra, ACT, Australia and [7]VTT Technical Research Center of Finland, Espoo, Finland

*Corresponding authors: E-mail: poh.chuehloo@nus.edu.sg

Abstract

Advances in hardware automation in synthetic biology laboratories are not yet fully matched by those of their software counterparts. Such automated laboratories, now commonly called biofoundries, require software solutions that would help with many specialized tasks such as batch DNA design, sample and data tracking, and data analysis, among others. Typically, many of the challenges facing biofoundries are shared, yet there is frequent wheel-reinvention where many labs develop similar software solutions in parallel. In this article, we present the first attempt at creating a standardized, open-source Python package. A number of tools will be integrated and developed that we envisage will become the obvious starting point for software development projects within biofoundries globally. Specifically, we describe the current state of available software, present usage scenarios and case studies for common problems, and finally describe plans for future development. SynBiopython is publicly available at the following address: http://synbiopython.org.

Key words: Software; Synthetic Biology; Biofoundries; Open-source; Automation

Introduction

With synthetic biology developing at an increasing pace, there are now a large number of tools covering the Design-Build-Test-Learn (DBTL) cycle available to researchers, originating from both academic and commercial sources. For instance at the Design stage, computer-aided metabolic engineering tools such as Cameo (1) and RetroPath2.0 (2); or tools in the transition from Design to Build for sequence optimization such as DnaChisel (3) and PartsGenie (4); tools at the Build stage such as CloneFlow for planning ligase cycling reaction DNA assemblies (5); tools at the Test stage, such as mzmine for mass spectrometry data processing (6), tools from Test to Learn and Design such as BioModel Selection System (BMSS) that performs automated

Submitted: 6 December 2020; **Accepted:** 28 December 2020

BioModel selection (7), or tools facilitating the transition from Learn to Design such as the cobrapy library for genome-scale metabolic modeling (8). Increasing automation in synthetic biology laboratories [the consensus term for such an automated lab used for synthetic biology research and development is 'biofoundry' (see also ref. to GBA article, https://biofoundries.org/)] is posing another set of problems. While many laboratories may not require sophisticated software for data collection, sample tracking or batch genetic construct design, such software is essential for heavily automated labs. The main reason for this is the significant number of samples being processed daily (in the order of 10^2 to 10^4). Without appropriate software, manually generated mistakes can become increasingly prevalent and, given the volume of samples processed, such mistakes can become very costly in terms of both time and money.

Selecting appropriate software solutions for an automated laboratory can be difficult. The solutions are scattered and there are no definite guidelines or universally agreed state-of-the-art. As a result, many groups have created software solutions in-house, which typically involves directly hiring developers. This approach, however, leads to the multiplication of efforts, and since these solutions are usually developed with that specific lab in mind, it is often difficult to reuse these solutions in a different lab, even if the code is open-sourced. Commercial solutions, on the other hand, are usually developed with big operations in mind and do not scale well to smaller operations. Additionally, such solutions are often expensive and such costs are hard to justify for a relatively small, albeit automated, lab. Furthermore, many tools, whether academically or commercially developed, are typically end-to-end applications. Such solutions provide a predefined set of functionalities, which are difficult for other developers to unpick in order to reuse individual components in their own software.

In the 2000s, the bioinformatics community was in a similar situation and created Biopython (9) as a library of primitives. The advantages of such libraries include, (i) increased reliability, due to community testing; (ii) increased reusability and interoperability between the different modules of the project; and (iii) increased community uptake, due to easier discovery of features that are organized under a single umbrella project. The approach has been very successful, with over 2000 manuscript citations and 3500 Github projects using Biopython. However, since Biopython is primarily focused toward classical bioinformatics, with an emphasis on sequence analysis, the Global Biofoundries Alliance (GBA) (10) software group identified a need for a library specific to the requirements of the synthetic biology community. These requirements include tools assisting in DNA design and assembly, software for automation and robotic equipment. A project specific to synthetic biology provides better visibility and also encourages contributions from developers in this field.

The Software Working Group of Global Biofoundries Alliance, therefore, introduces a new package, named *SynBiopython*, to support aspects of development efforts that are common to many DNA design and assembly projects. Python is recognized as being ubiquitous in biofoundry software development efforts and is, therefore, a natural choice for such a consolidated, collaborative effort.

In introducing this work, it is recognized that there remains a large amount of development work to be performed, requiring the introduction of a multitude of new modules, for the package to be considered 'full-suite'. This article, therefore, acts as a 'call to arms' on the synthetic biology community, introducing the concept of reusable libraries, exemplifying its use through the development of specific, community-developed modules and specifying the governance requirements to manage the growth of the resource over time.

The initial modules demonstrated here include standard file parsers and tool interoperability, an automation library and support for codon usage tables. The presented tools were chosen from a number of tools that were originally written in different biofoundries that are part of the GBA. The decision was made to work on these modules first to meet the following general objectives of SynBiopython: (i) collation and development of synthetic biology-oriented code and tools in Python; (ii) support for both novice and advanced developers of synthetic biology software; and (iii) prevention of duplicated efforts. There are also a number of specific aims that the Authors would like SynBiopython to meet: (i) standardization of read/write operations and other procedures and automation related tools to allow ease of access and interaction; (ii) simplification of parsing of different synthetic biology-related file formats; and (iii) development of more intuitive APIs and wrapper functions on top of more complex code, hiding underlying details.

Results

Here, we describe the three modules that show how future modules in SynBiopython should be written and used. Each module is provided together with a case study and some example code for easier understanding. First, we describe Genbabel, a tool that enables translation between file formats relevant to synthetic biology. Next, we discuss the Automation Library, a module that can be used to create instruction files for automated equipment. Finally, we show how the Codon Usage Tables tool can be used to optimize DNA sequences.

Standard file parsers and tool interoperability: Genbabel

Driving interoperability between tools via a common standard is a deep-rooted effort in synthetic biology. Several standard file formats such as GenBank (11), FASTA, Synthetic Biology Open Language (SBOL) (12), Systems Biology Markup Language (SBML) (13), Simulation Experiment Description Markup Language (SED-ML) (14) and Computational Modeling in Biology Network (COMBINE) archives (15) have been proposed at different information levels to overcome the reproducibility challenge and to serve as a common integrated knowledge base for data sharing. Despite these standards having been adopted in many of the developed tools, there is no one-size-fits-all tool that supports the parsing of these common standard files in the field of synthetic biology. To mitigate these issues, SynBiopython introduces a universal environment, named Genbabel, which serves as a repository of standard file parsers built upon existing libraries and applications to enable easy generation and conversion of different standard files as mentioned above, including formats for DNA/protein sequences, genetic circuits, and model simulations. This aims to reduce redundant or overlapping efforts and to improve reusability which are essential to accelerate the progress of the field.

At the lower level, GenBank and FASTA files are the most ubiquitous standard formats used to encode DNA and protein sequence data (11). To capture the structural information at a higher level, the SBOL and SBOL Visual compliant diagrams have been introduced and applied in many software platforms (12). To enable the transferability of different data standards, built upon existing online platform and packages (16–18), a standard file parser has been developed in Genbabel to support the conversion between SBOL files and the aforementioned sequence data formats including General Feature Format (GFF3), and the rendering of highly customizable genetic circuits and their associated regulations.

Aligning with the use of the model-driven approach in forward and reverse cell engineering, the advent of SBML enables the representation of computational models in a declarative form to ease the exchange of quantitative descriptions (13). SBML is widely used for modeling and simulation for chassis optimization through Python-based tools for flux analysis and knock-out/knock-in optimization such as COBRApy (8) or cameo (1). Genome-scale metabolic models for the most common industrial hosts are available at public databases, such as BioModels (19) and BiGG (20) and can be downloaded in SBML. The support of modeling in general and SBML in particular is therefore of increasing interest to the synthetic biology community. The Genbabel module was thus extended to provide the capability of generating SBML files and other formats related to modeling, such as SED-ML (14) and COMBINE archives in Open Modeling EXchange (OMEX) format (15), which are hinged on several developed packages (21–23).

The incorporation of Genbabel module in the SynBiopython package seeks to provide a universal parser environment which supports the gathering of and interfacing with parsers spanning across gene sequence, circuit, and systems levels. Longer-term goals include the development of an improved interface, linking file parsing encoded in different formats from sequence, structure, model, simulation and analysis.

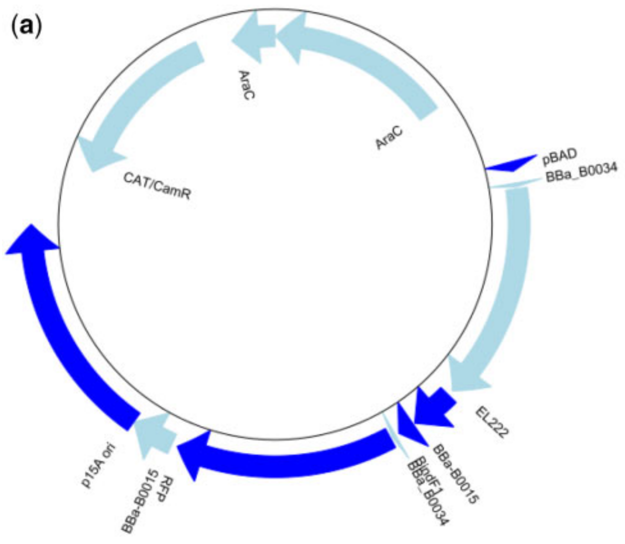
Case study: standard file generation and model generation

To demonstrate the capability of Genbabel, we present an example (Code Block 1) to demonstrate the conversion of a GenBank file, which encodes an AND logic gate genetic circuit generated from Benchling (Benchling Inc., San Francisco, USA) during the Design phase, to SBOL file using the GenSBOLconv submodule. The circular plasmid map can also be constructed based on the given GenBank file (Figure 1a). Meanwhile, the corresponding SBOL-compliant genetic circuit diagram can be generated using the SimpleDNAplot submodule (Figure 1b).

During the Design phase before the actual circuit construction, one can also utilize the SBMLgen submodule to generate the SBML file which encodes the kinetic model of the AND gate for simulation as demonstrated in Code Block 2. All the different elements such as the ODEs, variables, initial conditions, parameters names, values and units are to be provided in lists of strings as input arguments to the export_sbml function. Otherwise, in the Learn phase, using characterization data of the AND gate, an SBML file can also be generated via running the BMSS tool (7). With the available SBML file, the corresponding SED-ML file and COMBINE archive in OMEX format can subsequently be generated and executed using the SEDMLOMEXgen submodule with the AND gate simulation results shown in Figure 2. These formats ensure the reproducibility of the model implementation and simulation. A detailed example code implementation is provided in the SynBiopython GitHub repository.

```
import synbiopython.genbabel as stdgen

# Create an object from the class GenSBOLconv:
stdconv = stdgen.GenSBOLconv()

# Convert GenBank file to SBOL file:
uri_prefix = 'http://synbiohub.org/public/igem'
inputfile = 'pBAD_BLind_AND.gb'
output = 'SBOL2'
stdconv.run_sbolvalidator(inputfile, output, uri_prefix)

# Export the plasmid map of the GenBank file (see Figure 1a):
stdconv.export_plasmidmap(inputfile)

# create an object from the class SimpleDNAplot (see Figure 1b):
simplot = stdgen.SimpleDNAplot()
circuit = '-c.orange.AraC -p p.pBAD r.rbs34 c.blue.EL222 t p.BLind r.rbs34 c.red.RFP
t o.p15A'
regulations = 'c0->p1.Repression p1->p1.Derepression.red c1->p2.Activation p2-
>p2.Activation2.blue'
simplot.plot_circuit(circuit, regulations, "ANDgate_circuitplot.png")
```

Code Block 1. Demonstration of the features of GenSBOLconv and SimpleDNAplot submodules. Here, we exemplified the conversion of a GenBank file to SBOL file, which encodes an AND logic gate for a blue-light inducible system, using the run_sbolvalidator function from GenSBOLconv class. This function enables the interconversion of GenBank, Fasta, GFF3 and SBOL files. With the provided GenBank file, the linear and circular plasmid maps can be exported. Users can also employ the plot_circuit function from SimpleDNAplot class to generate the SBOL-compliant gene circuit diagram. The circuit configuration and the corresponding regulations were to be defined in the form of string following proper sequences separated by spaces. The alphabets p, r, c, t, o represent the promoter, ribosome binding site, coding sequence, terminator and origin with the negative sign denoting the reverse direction. Each of the parts consists of the part type followed by the color (optional) and part name (optional). The regulations were defined in the form of 'from part->to part' followed by the type of regulation and color (optional). The parts were numbered starting from 0 following the sequences defined in circuit from left to right.



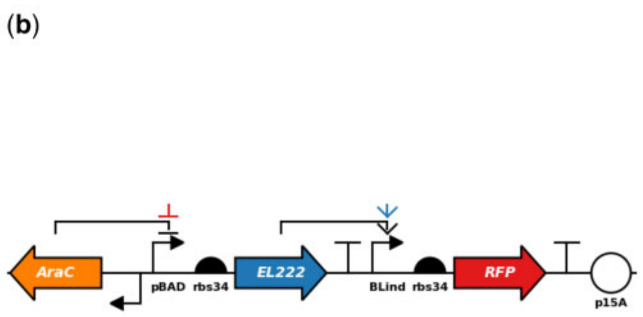


Figure 1. A case study of an AND logic gate genetic circuit generated using the Genbabel module. The AND gate consists of a blue-light inducible system using a photosensitive DNA-binding protein EL222 (24, 25). The system is turned on in the presence of both blue light and the arabinose inducer, to drive the expression of red fluorescent proteins (RFPs). (**a**) Circuit plasmid map generated using the GenSBOLconv submodule; and (**b**) SBOL Visual compliant gene circuit diagram generated using the SimpleDNAplot submodule of Genbabel module.

```
import synbiopython.genbabel as stdgen
.
. define odes, variables, initial values, parameters' names, values, units
.

# Generate and export the SBML file:
sbmlgen = stdgen.SBMLgen()
sbml_str = sbmlgen.export_sbml(odes, variables, init, param_name, param,
param_units, outputfile = "ANDgate_sbml.xml")

# Generate, and export the SEDML file and OMEX archive:
omexgen = stdgen.SEDMLOMEXgen()
antimony_str = omexgen.sbmltoantimony("ANDgate_sbml.xml")
phrasedml_str = """
      model1 = model "{}"
      sim1 = simulate uniform(0, 720, 1000)
      sim1.algorithm = rk4
      state_11 = run sim1 on model1
      model2 = model model1 with state1 = 1, state2 = 0
      state_10 = run sim1 on model2
      ...
    """
omex_str= omexgen.export_omex(antimony_str, phrasedml_str)
```

Code Block 2. Demonstration of the features of SBMLgen and SEDMLOMEXgen submodules. To generate a SBML file, the function export_sbml from the SBMLgen class is used to generate the ANDgate_sbml.xml file. Input arguments such as ODEs, variables, initial conditions, parameters' names, values and units have to be defined and provided into the function. To generate the COMBINE omex file, the previously generated SBML file is read and converted into an antimony string representation using function from SEDMLOMEXgen submodule. Users can then define the phrasedml string which encodes the descriptions for the simulation experiment. The antimony and the phrasedml strings are then supplied as input arguments to the export_omex function to generate the corresponding omex file.

Automation library

In the spirit of developing computational infrastructure across the DBTL cycle, the Build phase is supported through the introduction of the SynBiopython automation library. The goal of the library is to provide an easy-to-use, standardized solution for the creation of automated workflows for biofoundries. It is envisaged that the library will act as the first software suite that a user of a biofoundry will have contact with and by setting good practices it will reinforce them in the users.

There are currently a number of solutions that allow lab workflow automation, including Aquarium (26), Antha (Synthace Ltd., London, UK), TeselaGen BUILD (TeselaGen Biotechnology Inc., San Francisco, USA) and the Autoprotocol [Strateos Inc. (formerly Transcriptic), Menlo Park, USA]. However, these may not be suitable for a biofoundry operator. Some of these are still under development, or do not allow the development of protocols via scripts. Proprietary software can be costly and inhibits collaborative development and adapting the software to custom needs.

The GBA recognizes automation to be a major bottleneck in the development of the biofoundry technology. For the most part, each lab uses their own collection of open-source, in-house and commercial automation software which makes collaborations and comparative studies very difficult. Many of the routine tasks that are performed in biofoundries involve liquid manipulation, including dilutions, normalizations, transfers between plates and rearraying. Creating reliable protocols and picklists for such operations is a time-consuming effort and, without proper software support, very error prone.

The introduction of the lab automation module within the SynBiopython package aims to address these issues. This module, adapted from Plateo by the Edinburgh Genome Foundry, enables generation of picklists and protocols for commonly used machines, focusing on liquid handlers (e.g. Labcyte Echo). A long-term goal is the integration of this module with a number of open-source libraries and APIs (e.g. Biopython, Benchling, Teselagen, other common LIMS or DNA synthesis providers) to enhance its data-tracking capability.

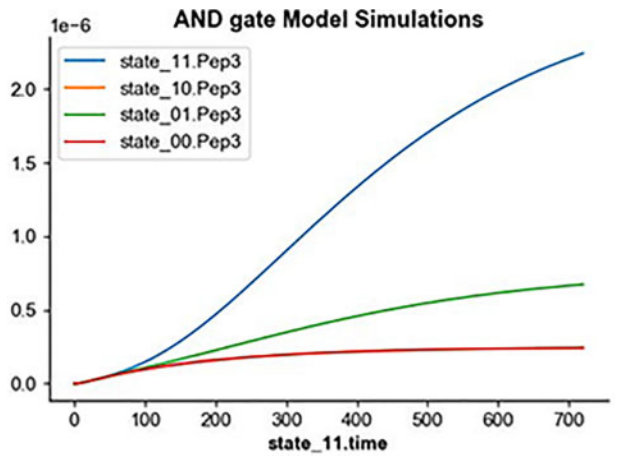


Figure 2. Model simulation results for the AND logic gate with four state inputs (00, 01, 10, 11) generated from the SBML and SED-ML files contained in the COMBINE archive OMEX format. The simulation is performed using the execute_inlineomex function from the SEDMLOMEXgen submodule.

Case study: generating a picklist

The lab automation module includes a number of classes: the plate class (e.g. a microplate) which contains objects of the well class (which stores information about the contents of a given well in the plate), the transfer class that stores information about transfers to be performed between wells and finally the

```
import synbiopython.lab_automation as lab

source = lab.Plate96(name="96-well plate")
source.wells["A1"].add_content({"Compound_1": 1}, volume=5 * 10 ** (-6))
source.wells["C2"].add_content({"Compound_X": 1}, volume=15 * 10 ** (-6))

destination = lab.Plate384(name="384-well plate")

transfer_1 = lab.Transfer(
    source.wells["C2"], destination.wells["G8"], 3 * 10 ** (-6))

transfer_2 = lab.Transfer(
    source.wells["A1"], destination.wells["I12"], 2 * 10 ** (-6))

# A picklist represents a list of well-to-well transfers:
picklist = lab.PickList()
picklist.add_transfer(transfer=transfer_1)
picklist.add_transfer(transfer=transfer_2)

print(picklist.to_plain_string())

# Transfer 3.00E-06L from 96-well plate C2 into 384-well plate G8
# Transfer 2.00E-06L from 96-well plate A1 into 384-well plate I12
# Perform the transfers in the picklist:
picklist.simulate()
```

Code Block 3. Demonstration of the lab automation module. First, a 96 well source plate object is created, followed by two lines which fill the wells of that plate with content of given volume (plate is created empty). Next, a 384 well destination plate is created, then two transfers from source wells to destination wells are defined. Finally, a picklist is created, and the transfers are added to it. The user can then choose to simulate the picklist to see if the transfers are resolved correctly.

```
from collections import Counter

from synbiopython.codon import table, taxonomy_utils, utils

# Use codon module to map between organism names and taxonomy ids:
name = taxonomy_utils.get_organism_name(4932)
tax_id = taxonomy_utils.get_tax_id("Saccharomyces cerevisiae")

print("Name:", name)
print("Taxonomy id:", tax_id)

# Codon usage tables can be retrieved through either name or taxonomy id:
name_table = table.get_table(name)
tax_id_table = table.get_table(tax_id)

# These will be the same, irrespective of the method of retrieval:
assert name_table == tax_id_table

# The table is a simple dictionary of amino acids to codons,
# which are themselves a codon to usage frequency dictionary:
l_codons = name_table["L"]
print(l_codons)

# Utility methods are available for random sampling of codons, and for performing
# simple codon optimisations:

# Sample:
sampled = [utils.sample(name_table, "L") for _ in range(10000)]

codons = Counter(sampled)

for cdn, count in codons.items():
    print(cdn, count / len(sampled), l_codons[cdn])

# Codon optimise:
aa_seq = 'ACDEFGHIKLMNPQRSTVWY'
print(utils.optimise(name_table, aa_seq))
```

Code Block 4: Demonstration of the features of the Codon Usage module, codon. The taxonomy_utils module supports mapping between organism names and taxonomy ids. The names and the taxonomy ids can be used to retrieve the codon usage table which is a simple dictionary of amino acids to codons, and the codons are themselves a dictionary of a codon to usage frequency.

picklist class, which contains a list of transfers to be made within a single plate or between different plates.

The example in Code Block 3 shows how a picklist can be generated. The picklist object can be initiated with a predetermined list of transfers to be performed or the transfers can be directly defined using the add_transfer method. After defining all the transfers, the picklist can be then translated to a form accepted by a relevant liquid handler (which will be a future feature) and finally executed. More detailed code demonstrating these features is available in the examples directory of the code repository.

Codon usage tables

A typical task in the Design step of a biofoundry workflow is the optimization of the coding sequence of a given amino acid sequence for recombinant expression in a host of interest. As codon usage differs across organisms, such codon optimization is reliant on codon usage tables, which specify a given organism's frequency of use of each degenerate codon. While codon usage tables are publicly available (27), there remains as yet no standardized means for the programmatic access and manipulation.

SynBiopython consequently includes a module for support of codon usage tables and codon optimization. This module is based on previous work from the Manchester Centre for Synthetic Biology (SYNBIOCHEM) and the Edinburgh Genome Foundry. Following a simple interface, codon usage tables may be automatically accessed from the Kazusa Codon Usage Database and used in a number of codon optimization methods. The library complements the existing Biopython CodonTable module but includes codon frequency in addition to translation tables. Example code for the codon module is provided in Code Block 4.

Future work may include support for custom codon usage tables of novel or rare organisms, more sophisticated codon optimization algorithms, and support for Biopython sequences.

Future directions

It is hoped that future directions of development for the SynBiopython library will be driven by the needs of the community, and by interested volunteers who would happily provide useful modules that would be of general utility. Synthetic biology is an umbrella term, encompassing a number of subdisciplines, and the SynBiopython project aspires to support a range of tools and applications across these numerous subcommunities.

One such application of interest for the metabolic engineering community is to provide a straightforward scripting way for *in silico* prototyping of genetic constructs once inserted into the optimized chassis organisms. Such approach should be addressed effortlessly, as examples and tutorials exist in both cameo and COBRApy about adding biochemical species and reactions to genome-scale models in order to represent the genetic circuit of metabolic circuits and pathways.

To connect the dots, combinatorial genetic circuits represented in SBOL and designed through tools such as Cello (28) should generate annotated SBML models that can be seamlessly added into the genome-scale model of the chassis. Standard interconversion procedures exist between SBML and SBOL and have been implemented in the Java-based iBioSim tool (29). In general, the generation of an annotated SBML model from SBOL can be accomplished by using terms from ontologies (30). Ontologies are controlled vocabularies that can be associated with different elements in the SBOL. The Sequence Ontology (31) allows defining roles to the components such as promoters, coding sequences or terminators, while the Systems Biology Ontology (SBO) (32) allows the definition of biochemical species and reactions. As a first approach, SBOL should provide the minimal annotations required in order to be able to integrate the engineered circuit of the pathway into the SBML model and perform steady-state flux analysis simulations.

Furthermore, an extensive number of freely available online (web-based) and offline tools are available to expedite the different phases of the DBTL cycle of synthetic biology. Supporting interoperability between these tools will allow for more efficient development of computational pipelines and the reduction of redundant efforts (Figure 3). Such interoperability will pave the way toward a long-standing goal of synthetic biology: full lab automation assisted by streamlined computer-aided tools. Several useful tools that serve to automate design, modeling and optimization phases are compiled below, and depending upon the priorities of the community, these will be incrementally supported by future developments of SynBiopython. Such a platform for the support of third-party applications will be made highly extensible to allow more tools to be interfaced over time.

Potential tools for future incorporation are as follows, and readers are encouraged to contact the Authors with comments regarding their prioritization and to make further suggestions.

a. ‘Cello’ allows for the automatic design of genetic logic gates using a high-level language known as Verilog. Circuit performance can be predicted, factoring in growth and load (28).
b. The ‘RBS Calculator’ predicts translation initiation rates, based on the start codon of mRNA transcripts, and designs and optimizes synthetic ribosome binding site (RBS) sequences to achieve a desired translation rate (34).
c. The ‘Biomodel Selection System (BMSS)’ automatically derives or selects the best mathematical model based upon part/circuit characterization data (7).
d. sRNA design tools include ‘IntaRNA’ which is used to predict the mRNA target sites for a given sRNA or to predict the interactions between two RNA molecules. ‘CopraRNA’ is built upon IntaRNA and computes whole-genome sRNA target predictions for a set of given organisms (35).
e. gRNA design tools. ‘Cas-OFFinder’, ‘CHOPCHOP’ and ‘CRISPOR’ are free web-based tools which allow off-target site analysis, with some providing specificity scores and cleavage likelihood of a gene sequence (36).

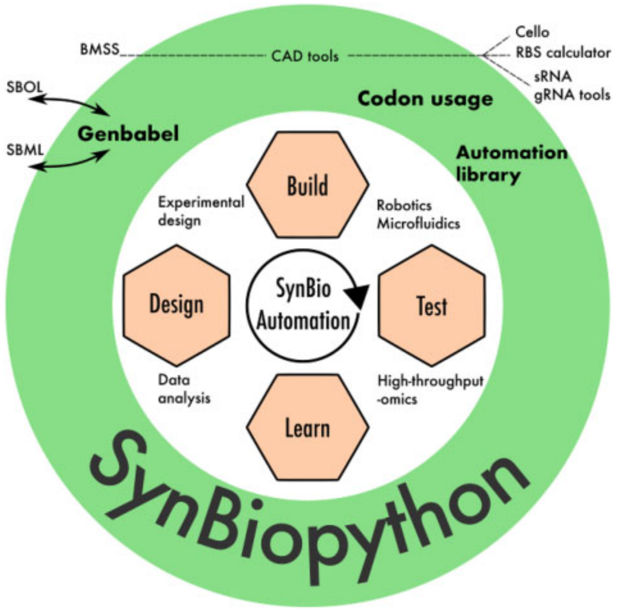


Figure 3. Current SynBiopython modules within the Design-Build-Test-Learn (DBTL) cycle of synthetic biology (33). Genbabel module provides the link between Design and Build by allowing the interconversion of sequence-based files into gene circuit representation format in SBOL, and the generation of SBML models and other modeling-related formats, which could then be interfaced through the CAD tools with external tools such as BMSS, Cello, RBS Calculator or sRNA, gRNA tools. The link between Build and Test is implemented through the Codon usage and the Automation library.

The overarching goal is to support the interoperability of file formats and software tools, from sequence design, through automation, data analysis and representation, and machine learning, allowing for the development of computational pipelines across the DBTL cycle, complementing the work conducted on the bench.

Conclusion

This work introduces SynBiopython, in which initial efforts in creating a standardized, open-source, Python library to be used in biofoundry-type facilities around the world are demonstrated. The library is modeled on the existing Biopython library, being divided into modules of different functionalities. To our knowledge, this is the first synthetic biology specific software package for standardizing development efforts across automated synbio facilities.

It is strongly envisaged that SynBiopython will be a community effort. As the global biofoundry community grows and more labs join the automation effort, the hope is to attract more developers and other stakeholders. A key goal is for members of the community to offer additional modules, used locally in their own labs but with perhaps wider utility, and thereby to help with the development and curation of the package. Such an approach has many mutual benefits, reducing duplication of efforts and thereby freeing up resources to focus on the development of more novel and innovative methods. It is clear that there are developers and users in the general synbio community with skills and interests that would benefit the development efforts of the SynBiopython package, and interested members are encouraged to mail info@synbiopython.org to discuss their potential involvement.

With an increasing number of contributors, a governance model will also be developed to help steer the future development of the package. Such governance matters include deciding on the scope of the package and which new modules to prioritize, and more technical matters including code standardization, automated testing and documentation requirements. All such decisions will be made with the consultation of the SynBiopython development community and more details can be found in the relevant file in the Github repository.

With the introduction of the SynBiopython package, a clear mechanism for the sharing and reusability of code being developed in individual biofoundries is proposed. Promoting such standardization and interoperability is not intended to stifle innovation, but rather to support the development of novel approaches through reducing effort spent on finding solutions to universal problems that are shared across many labs. Such developments are of benefit to all stakeholders in synthetic biology, from lab-based researchers, informaticians, research leaders and funders.

Funding

N.S. acknowledges funding from the Biotechnology and Biological Sciences Research Council (BBSRC) under grant 'GeneORator: a novel and high-throughput method for the synthetic biology-based improvement of any enzyme' (BB/S004955/1) and from the University of Liverpool. The Edinburgh Genome Foundry is supported by the UK Research Councils Synthetic Biology for Growth Programme and acknowledges funding from BBSRC grants (BB/M00029X/1, BB/M025640/1, BB/M025659/1, BB/M018040/1). SYNBIOCHEM is supported by the UK Biotechnology and Biological Sciences Research Council (BBSRC) and the Engineering and Physical Sciences Research Council (EPSRC) under grant: 'Centre for synthetic biology of fine and speciality chemicals (SYNBIOCHEM)' (BB/M017702/1). P.C. acknowledges support from the Universitat Politècnica de València Talento Programme. M.B.H. acknowledges support from CSIRO's Synthetic Biology Future Science Platform. C.L.P. acknowledges support from Singapore NRF Synthetic Biology Program (SBP-P5, SBP-P6) and the Synthetic Biology Initiative of the National University of Singapore (DPRT/943/09/14).

Conflict of interest statement. None declared.

Code availability

The code repository for SynBioPython is located on Github: https://github.com/Global-Biofoundries-Alliance/SynBioPython

References

1. Cardoso,J.G.R., Jensen,K., Lieven,C., Lærke Hansen,A.S., Galkina,S., Beber,M., Özdemir,E., Herrgård,M.J., Redestig,H., Sonnenschein,N. et al. (2018) Cameo: a Python library for computer aided metabolic engineering and optimization of cell factories. *ACS Synth. Biol.*, 7, 1163–1166.
2. Delépine,B., Duigou,T., Carbonell,P. and Faulon,J.-L. (2018) RetroPath2.0: a retrosynthesis workflow for metabolic engineers. *Metab. Eng.*, 45, 158–170.
3. Zulkower,V. and Rosser,S. (2020) DNA Chisel, a versatile sequence optimizer. *Bioinformatics*, 36, 4508–4509.
4. Swainston,N., Dunstan,M., Jervis,A.J., Robinson,C.J., Carbonell,P., Williams,A.R., Faulon,J.-L., Scrutton,N.S. and Kell,D.B. (2018) PartsGenie: an integrated tool for optimizing and sharing synthetic biology parts. *Bioinformatics*, 34, 2327–2329.
5. Chandran,S. (2017) Rapid assembly of DNA via ligase cycling reaction (LCR). *Methods Mol. Biol.*, 1472, 105–110.
6. Katajamaa,M., Miettinen,J. and Oresic,M. (2006) MZmine: toolbox for processing and visualization of mass spectrometry based molecular profile data. *Bioinformatics*, 22, 634–636.
7. Yeoh,J.W., Ng,K.B.I., Teh,A.Y., Zhang,J.Y., Chee,W.K.D. and Poh,C.L. (2019) An automated biomodel selection system (BMSS) for gene circuit designs. *ACS Synth. Biol.*, 8, 1484–1497.
8. Ebrahim,A., Lerman,J.A., Palsson,B.O. and Hyduke,D.R. (2013) COBRApy: COnstraints-based reconstruction and analysis for Python. *BMC Syst. Biol.*, 7, 74.
9. Cock,P.J.A., Antao,T., Chang,J.T., Chapman,B.A., Cox,C.J., Dalke,A., Friedberg,I., Hamelryck,T., Kauff,F., Wilczynski,B. et al. (2009) Biopython: freely available Python tools for computational molecular biology and bioinformatics. *Bioinformatics*, 25, 1422–1423.
10. Hillson,N., Caddick,M., Cai,Y. et al. (2019) Building a global alliance of biofoundries (vol 10, 2040, 2019). *Nat. Commun.*, 10, 1–4.
11. Benson,D.A., Boguski,M.S., Lipman,D.J., Ostell,J., Ouellette,B.F.F., Rapp,B.A. and Wheeler,D.L. (1999) GenBank. *Nucleic Acids Res.*, 27, 12–17.
12. Galdzicki,M., Clancy,K.P., Oberortner,E., Pocock,M., Quinn,J.Y., Rodriguez,C.A., Roehner,N., Wilson,M.L., Adam,L., Anderson,J.C. et al. (2014) The Synthetic Biology Open Language (SBOL) provides a community standard for communicating designs in synthetic biology. *Nat. Biotechnol.*, 32, 545–550.
13. Hucka,M., Finney,A., Sauro,H.M. et al. (2003) The systems biology markup language (SBML): a medium for representation and exchange of biochemical network models. *Bioinformatics*, 19, 524–531.
14. Waltemath,D., Adams,R., Bergmann,F.T., Hucka,M., Kolpakov,F., Miller,A.K., Moraru,I.I., Nickerson,D., Sahle,S., Snoep,J.L. et al. (2011) Reproducible computational biology experiments with SED-ML—the Simulation Experiment Description Markup Language. *BMC Syst. Biol.*, 5, 198.
15. Bergmann,F.T., Adams,R., Moodie,S., Cooper,J., Glont,M., Golebiewski,M., Hucka,M., Laibe,C., Miller,A.K., Nickerson,D.P. et al. (2014) COMBINE archive and OMEX format: one file to share all information to reproduce a modeling project. *BMC Bioinformatics*, 15, 369.
16. Zundel,Z., Samineni,M., Zhang,Z. and Myers,C.J. (2017) A validator and converter for the synthetic biology open language. *ACS Synth. Biol.*, 6, 1161–1168.
17. Der,B.S., Glassey,E., Bartley,B.A., Enghuus,C., Goodman,D.B., Gordon,D.B., Voigt,C.A. and Gorochowski,T.E. (2017)

DNAplotlib: programmable visualization of genetic designs and associated data. *ACS Synth. Biol.*, 6, 1115–1119.

18. Quinn,J.Y., Cox,R.S., Adler,A., Beal,J., Bhatia,S., Cai,Y., Chen,J., Clancy,K., Galdzicki,M., Hillson,N.J. et al. (2015) SBOL visual: a graphical language for genetic designs. *PLoS Biol.*, 13, e1002310.
19. Le Novere,N., Bornstein,B., Broicher,A. et al. (2006) BioModels Database: a free, centralized database of curated, published, quantitative kinetic models of biochemical and cellular systems. *Nucleic Acids Res.*, 34, D689–D691.
20. Schellenberger,J., Park,J.O., Conrad,T.M. and Palsson,B.Ø. (2010) BiGG: a Biochemical Genetic and Genomic knowledgebase of large scale metabolic reconstructions. *BMC Bioinformatics*, 11, 213.
21. Cannistra,C., Medley,K. and Sauro,H. (2015) SimpleSBML: a Python package for creating and editing SBML models. *BioRxiv; 030312*. doi: 10.1101/030312.
22. Bornstein,B.J., Keating,S.M., Jouraku,A. and Hucka,M. (2008) LibSBML: an API library for SBML. *Bioinformatics*, 24, 880–881.
23. Choi,K., Medley,J.K., König,M., Stocking,K., Smith,L., Gu,S. and Sauro,H.M. (2018) Tellurium: an extensible python-based modeling environment for systems and synthetic biology. *Biosystems*, 171, 74–79.
24. Jayaraman,P., Devarajan,K., Chua,T.K., Zhang,H., Gunawan,E. and Poh,C.L. (2016) Blue light-mediated transcriptional activation and repression of gene expression in bacteria. *Nucleic Acids Res.*, 44, 6994–7005.
25. Jayaraman,P., Yeoh,J.W., Zhang,J. and Poh,C.L. (2018) Programming the dynamic control of bacterial gene expression with a chimeric ligand- and light-based promoter system. *ACS Synth. Biol.*, 7, 2627–2639.
26. Keller B.,Vrana J.,Miller A., et al. Aquarium: the laboratory operating system (Version v2.5.0). Zenodo.
27. Nakamura,Y., Gojobori,T. and Ikemura,T. (2000) Codon usage tabulated from international DNA sequence databases: status for the year 2000. *Nucleic Acids Res.*, 28, 292–292.
28. Nielsen,A.A.K., Der,B.S., Shin,J., Vaidyanathan,P., Paralanov,V., Strychalski,E.A., Ross,D., Densmore,D. and Voigt,C.A. (2016) Genetic circuit design automation. *Science*, 352, aac7341.
29. Watanabe,L., Nguyen,T., Zhang,M., Zundel,Z., Zhang,Z., Madsen,C., Roehner,N. and Myers,C. (2019) iBioSim 3: a tool for model-based genetic circuit design. *ACS Synth. Biol.*, 8, 1560–1563.
30. Roehner,N., Zhang,Z., Nguyen,T. and Myers,C.J. (2015) Generating systems biology markup language models from the synthetic biology open language. *ACS Synth. Biol.*, 4, 873–879.
31. Eilbeck,K., Lewis,S.E., Mungall,C.J., Yandell,M., Stein,L., Durbin,R. and Ashburner,M. (2005) The Sequence Ontology: a tool for the unification of genome annotations. *Genome Biol.*, 6, R44.
32. Juty,N. (2010) Systems biology ontology: update. *Nature Precedings*. doi: 10.1038/npre.2010.5121.1.
33. Carbonell,P., Radivojevic,T. and García Martín,H. (2019) Opportunities at the intersection of synthetic biology, machine learning, and automation. *ACS Synth. Biol.*, 8, 1474–1477.
34. Salis,H.M. (2011) The ribosome binding site calculator. *Methods Enzymol.*, 498, 19–42.
35. Wright,P.R., Georg,J., Mann,M., Sorescu,D.A., Richter,A.S., Lott,S., Kleinkauf,R., Hess,W.R. and Backofen,R. (2014) CopraRNA and IntaRNA: predicting small RNA targets, networks and interaction domains. *Nucleic Acids Res.*, 42, W119–W123.
36. Wilson,L.O.W., O'Brien,A.R. and Bauer,D.C. (2018) The current state and future of CRISPR-Cas9 gRNA design tools. *Front. Pharmacol.*, 9, 749.