Structuring spreadsheets with *ObjTables* enables data quality control, reuse, and integration

Jonathan R. Karr\textsuperscript{1,2}*\textsuperscript{,} Wolfram Liebermeister\textsuperscript{3}, Arthur P. Goldberg\textsuperscript{1,2}, John A. P. Sekar\textsuperscript{1,2} \& Bilal Shaikh\textsuperscript{1,2}

June 6, 2020

\textsuperscript{1}Icahn Institute for Data Science and Genomic Technology, Icahn School of Medicine at Mount Sinai, New York, NY 10029, USA.
\textsuperscript{2}Department of Genetics and Genomic Sciences, Icahn School of Medicine at Mount Sinai, New York, NY 10029, USA.
\textsuperscript{3}Université Paris-Saclay, INRAE, MalAGE, 78350, Jouy-en-Josas, France.

*Corresponding author: Jonathan Karr (karr@mssm.edu)

Abstract

A central challenge in science is to understand how systems behaviors emerge from complex networks. This often requires aggregating, reusing, and integrating heterogeneous information. Supplementary spreadsheets to articles are a key data source. Spreadsheets are popular because they are easy to read and write. However, spreadsheets are often difficult to reanalyze because they capture data ad hoc without schemas that define the objects, relationships, and attributes that they represent. To help researchers reuse and compose spreadsheets, we developed *ObjTables*, a toolkit that makes spreadsheets human- and machine-readable by combining spreadsheets with schemas and an object-relational mapping system. *ObjTables* includes a format for schemas; markup for indicating the class and attribute represented by each spreadsheet and column; numerous data types for scientific information; and high-level software for using schemas to read, write, validate, compare, merge, revision, and analyze spreadsheets. By making spreadsheets easier to reuse, *ObjTables* could enable unprecedented secondary meta-analyses. By making it easy to build new formats and associated software for new types of data, *ObjTables* can also accelerate emerging scientific fields.

1. Introduction

A central challenge in science is to understand how systems behaviors emerge from complex networks. This often requires integrating multiple types of information from varied sources. For example, a key goal of systems biology is to create whole-cell models that predict phenotype from genotype\textsuperscript{1,2} by combining genomic, transcriptomic, proteomic, metabolomic, biochemical, single-cell and other data.

Integrating heterogeneous data requires obtaining high-quality, well-annotated data in clear formats and software for parsing and manipulating the data. One way to achieve this is to develop structured domain-specific formats, repositories, and software. For example, genetic data can be
represented by FASTA, FASTQ, and Binary Sequence Alignment Map files; shared via repositories such as GenBank and the Sequence Read Archive, and analyzed by software such as the Genome Analysis Toolkit.

Because it takes substantial effort to create domain-specific tools, scientists often share data via simpler, more flexible, and less structured channels. This includes journal articles; spreadsheets (collections of worksheets or tables also known as workbooks); file-sharing platforms such as Dryad, FAIRDOMHub, FigShare, and Zenodo; and code repositories such as GitHub.

Supplementary spreadsheets – as well as comma-separated values (CSV), tab-separated values (TSV), and other similar files – to journal articles are one of the most popular mediums for sharing scientific data because they are human-readable and easy to read and write with widely-available, user-friendly software such as Google Sheets, LibreOffice Calc, Microsoft Excel, and OpenOffice Calc. Spreadsheets are also ubiquitous in industry. For example, over 650 million people regularly use Excel.

However, it is often difficult to reuse and compose spreadsheets for several reasons. First, the structure of their worksheets and columns is frequently ad hoc. For example, different authors often represent similar data with different worksheets and columns, and authors rarely explicitly communicate the structures of their spreadsheets. Second, spreadsheets frequently lack sufficient metadata for proper interpretation. Third, few software tools provide high-level methods for working with spreadsheets, such as comparing their content and parsing them into data structures suitable for analysis with programming languages such as Python.

Spreadsheets often have ad hoc structures for several reasons: spreadsheets are frequently designed for human rather than machine readability, spreadsheets only natively support a few data types, and there is no widely accepted convention for encoding multi-dimensional data, such as a multi-omics dataset of metabolite concentrations, protein abundances, and reaction fluxes, into a collection of two-dimensional spreadsheets. In addition to making spreadsheets difficult to reuse, their ad hoc structures make them challenging to validate, which leads to frequent errors.

Schemas, or descriptions of the types of objects in a dataset, their possible relationships, and their attributes, could make spreadsheets easier to reuse. Schemas could help authors create high-quality spreadsheets by enabling software that can find errors such as invalid data types, invalid relationships, and invalid values of enumerated attributes. Such quality control would make spreadsheets more reliable. Schemas would also enable authors to communicate the structure of their spreadsheets, which would make them easier for others to understand, as well as make them easier to compare and merge with other spreadsheets that use the same schema. In addition, abstracting schemas from spreadsheets enables generic software for parsing spreadsheets into high-level data structures which, in turn, make it easy for other researchers to reuse spreadsheets for additional analyses.

Schemas have long been used with relational databases, such as SQLite, and object relational mapping (ORM) systems such as Active Record and SQLAlchemy. Due to the similarities between spreadsheets and databases, over the past decade, researchers have begun to explore enhancing spreadsheets with database-style schemas. For example, IDEOM outlines tables for metabolomics data; ISA-Tab outlines tables and columns for capturing experimental studies; MAGE-TAB outlines tables for capturing microarray data; and SBtab outlines tables for describing biochemical data and models. However, IDEOM, ISA-Tab, MAGE-TAB, and SBtab are limited
to specific domains. Table Schema\textsuperscript{21} supports custom schemas. However, Table Schema does not support several essential design patterns for human-readability that are commonly among real-world spreadsheets such as table of contents worksheets, spreadsheet and worksheet-level metadata, additional unstructured worksheets and columns, grouped columns, many-to-many relationships between rows, transposed tables, and embedded grammars. DataSpread\textsuperscript{22} links spreadsheets with schemas via relational databases. However, this hybrid approach is cumbersome because it requires users to run a database. Tyszkiewicz\textsuperscript{23} and Cunha et al.\textsuperscript{24} have explored encoding relational databases into spreadsheets. However, their approaches create cumbersome spreadsheets that mirror the tables of a relational database, which are difficult for humans to read, negating some of the benefits of spreadsheets. Furthermore, many of the above tools lack software for parsing spreadsheets into high-level data structures that make it easy to reuse datasets with tools such as NumPy, Pandas, scikit-learn,\textsuperscript{25} and SciPy in programming languages such as Python.

While spreadsheets are similar to relational databases, we believe that schemas need to be tailored for the unique features of spreadsheets. One salient difference between spreadsheets and databases is that spreadsheets often emphasize human readability, whereas databases typically emphasize machine computability. One common mechanism for making a spreadsheet human-readable is to use multiple levels of headers to group related columns. Grouping columns corresponds to a database query that joins two tables. A second common mechanism is to use grammars to encode information into strings in individual cells. For example, scientists often encode the participants in a chemical reaction, their stoichiometries, and its reversibility into a string (e.g., ‘A + B → 2 C’). Embedding grammars into cells corresponds to a database query that encodes one or more joined tables into a single string-valued column.

Despite its limitations, spreadsheets continue to be one of the most popular media for sharing data. Toward higher quality and more reusable spreadsheets, we developed \textit{ObjTables}, an open-source toolkit for structuring human-readable spreadsheets with schemas tailored for spreadsheets (Fig. 1). The toolkit melds the ease of use of spreadsheets with the rigor of schemas and the power of imperative programming. \textit{ObjTables} includes a simple format for describing schemas for human-readable spreadsheets; simple markup syntax for indicating the schema class and attribute represented by each worksheet and column and capturing metadata; an object relational mapping system for using schemas to systematically read and write spreadsheets into and out of objects; and software for using schemas to validate, compare, merge, split, revision, migrate, and manipulate spreadsheets. The \textit{ObjTables} software is available through four interfaces: a web application, a command-line program, a REST API, and a Python package.

\textit{ObjTables} emphasizes spreadsheets that are both human and machine-readable so that spreadsheets can be easily read and written with widely-available software such as Microsoft Excel and LibreOffice Calc, as well as systematically quality controlled and manipulated. \textit{ObjTables} achieves machine and human-readability through several features. \textit{ObjTables} uses schemas to model spreadsheets as instances (rows) of classes (worksheets) that have multiple relationships and attributes (columns), \textit{ObjTables} provides a markup syntax for indicating the class represented by each worksheet and the relationship or attribute represented by each column, and the \textit{ObjTables} software can systematically parse, validate, and analyze spreadsheets that use this \textit{ObjTables’} syntax for spreadsheets and format for schemas. \textit{ObjTables} supports grouped columns with bi-level headings (e.g., Fig. 2c, d) and embedded grammars. Grouped columns and grammars enable users to design normalized schemas and map multiple classes to a single worksheet. Additionally,
Figure 1. Overview of the ObjTables toolkit for structured, reusable, human-readable spreadsheets. The ObjTables formats and software tools help researchers build, integrate, and combine high-quality datasets by combining the ease of use of spreadsheets (a) with the rigor of schemas (c) and the computational power of imperative programming (e). This combination enables researchers to read and write datasets as spreadsheets with widely-available, user-friendly programs such as Microsoft Excel and LibreOffice Calc (a), use schemas (c) and the ObjTables software (b) to map spreadsheets into object-oriented data structures (e), and use these data structures and the ObjTables software to rigorously validate datasets, as well as systematically compare, merge, split, revision, migrate, analyze, and pretty-print datasets (d). The ObjTables software can also convert datasets to comma- and tab-separated values (CSV and TSV) files for revisioning with version control systems such as Git and to JavaScript Object Notation (JSON) for analysis in other programming languages.

the ObjTables software can make spreadsheets easier to read by highlighting, freezing, and protecting column headings; embedding descriptions of columns into notes on their headings; hiding unused rows and columns; and creating an additional table of contents worksheet that contains the name and description of each data worksheet.

In addition to reading, writing, and validating spreadsheets, the ObjTables software provides several advanced features that make it simpler to reuse spreadsheets. (a) The software can help researchers compare datasets encoded into the same schema reported by multiple investigators by identifying differences in their content. (b) The software can help researchers integrate information from multiple sources by merging spreadsheets that are encoded into the same schema. (c) The software can help researchers develop schemas and datasets iteratively by revisioning schemas and spreadsheets and migrating spreadsheets between versions of their schemas. (d) The software makes it easy to analyze spreadsheets with packages such as Pandas and SciPy by parsing spreadsheets into data structures suitable for programming languages such as Python. Together, these features can help teams of researchers develop and analyze spreadsheets collaboratively.

Here, we describe the ObjTables toolkit and demonstrate how it can facilitate data reuse and integration. First, we describe the toolkit, including the format for schemas, the markup syntax for spreadsheets, and the software tools. Second, we present examples that illustrate how the toolkit can help researchers quality control, manipulate, and integrate data. Due to the popularity of spreadsheets, we believe that ObjTables could accelerate the integration of data into holistic models. ObjTables can also accelerate emerging fields of science by making it easy to develop new domain-specific formats and associated software.
2. Results

2.1. Toolkit for structured, human-readable spreadsheets

The ObjTables toolkit includes several interrelated tools for creating, validating, and reusing structured, human-readable spreadsheets. Here, we outline the components of the toolkit including the format for structured, human-readable spreadsheets; the format for schemas and supported datatypes; software tools for validating, comparing, and pretty-printing spreadsheets; and a Python package for more advanced operations such as analyzing, merging, splitting, revisioning, and migrating spreadsheets. More examples, tutorials, and documentation are available at https://objtables.org.

2.1.1. Format for structured, human-readable spreadsheets

ObjTables builds upon the Open Office Spreadsheet XML format (XLSX; ECMA-376; ISO/IEC 29500). To make spreadsheets both machine and human-readable, ObjTables incorporates the additional layout conventions and markup syntax described below. As an example, Fig. 2 illustrates how a dataset of the transcripts and genes of an organism can be encoded into a spreadsheet.

Declaring that a spreadsheet is encoded in the ObjTables format. To communicate that a spreadsheet uses the ObjTables format, ObjTables includes an additional worksheet that contains a table of contents for the spreadsheet. The first row of the table of contents worksheet contains a single cell which begins with the markup !!!ObjTables. As described below, this row can also contain metadata about the dataset. The other rows of the table of contents worksheet describe and provide hyperlinks to the worksheets which represent the data. ObjTables indicates the table of contents worksheet with the title !!_Table of contents. As described below, the ObjTables software can automatically generate table of contents worksheets for users.

Encoding data into spreadsheets. To help users find information in spreadsheets, ObjTables encodes each principal class of object in a dataset into a separate worksheet. For example, a dataset of genes and their splice variants would be encoded into two worksheets. ObjTables uses two mechanisms to declare that a worksheet represents data. First, data worksheets have titles that begin with the markup !!!, followed by the name of the class represented by the worksheet. Second, the first row of each data worksheet begins with the markup !!!ObjTables type='Data' class='class-name'.

ObjTables encodes each principal object into a single row in the corresponding worksheet for its class. For example, each gene would be encoded into a row in the 'Genes' worksheet.

ObjTables encodes each attribute of each class into a separate column. For example, the id, symbol, chromosome, and 3’ and 5’ coordinates of genes would be represented by five columns. ObjTables encodes the attribute represented by each column into an additional heading row before the data rows. Each heading begins with the markup !!, followed by the name of the attribute represented by the column.

To help make datasets easy to read, ObjTables can encode related attributes into adjacent columns. ObjTables indicates column groups by including an additional heading that spans all of the associated columns in an extra row above the headings for the individual columns. These headings also begin with the markup !!, followed by the name of the group of attributes. For example, columns that represent the chromosome and 5’ and 3’ coordinates of genes could be located next to each
a. ‘!!_Table of contents’ worksheet

| Table            | Description                               | Number of objects |
|------------------|-------------------------------------------|-------------------|
| Schema           | Structure of the worksheets and columns   |                   |
| Genes            | Genes in the genome                       | 2                 |
| Transcript variants | Splice variants expressed from the genome | 4                 |

b. ‘!!_Schema’ worksheet

| Name         | Type      | Parent | Format                  | Verbose name |
|--------------|-----------|--------|-------------------------|--------------|
| Gene         | Class     | row    | string(\text{primary}=\text{true}, \text{unique}=\text{true}) | Id           |
| id           | Attribute | Gene   |                         |              |
| symbol       | Attribute | Gene   | string                  | Symbol       |
| location     | Attribute | Gene   | OneToOne(\text{Location}, \text{related}_name=\text{genes}) | Location     |
| Transcript   | Class     | row    | string(\text{primary}=\text{true}, \text{unique}=\text{true}) | Transcript   |
| id           | Attribute | Transcript |                         | Id           |
| gene         | Attribute | Transcript | ManyToOne(\text{Gene}, \text{related}_name=\text{transcripts}) | Gene         |
| location     | Attribute | Transcript | OneToOne(\text{Location}, \text{related}_name=\text{transcripts}) | Location     |
| Location     | Class     | multiple_cells | string(\text{primary}=\text{true}, \text{unique}=\text{true}) | Location     |
| chromosome   | Attribute | Location | string                 | Chromosome   |
| five_prime   | Attribute | Location | PositiveInteger(\text{primary}=\text{true}, \text{unique}=\text{true}) | 5'           |
| three_prime  | Attribute | Location | PositiveInteger        | 3'           |

c. ‘!!Genes’ worksheet

| Id             | Symbol | Chromosome | 5'        | 3'        |
|----------------|--------|------------|-----------|-----------|
| ENSG00000130203| APOE   | 19         | 44,905,791| 44,909,933|
| ENSG00000139618| BRCA2  | 13         | 32,315,086| 32,400,266|

d. ‘!!Transcript variants’ worksheet

| Id               | Gene               | Chromosome | 5'        | 3'        |
|------------------|--------------------|------------|-----------|-----------|
| ENST00000252486.9| ENSG00000130203    | 19         | 44,905,796| 44,909,933|
| ENST00000425718.1| ENSG00000130203    | 19         | 44,906,360| 44,908,954|
| ENST00000380152.7| ENSG00000139618    | 13         | 32,315,474| 32,400,266|
| ENST00000544455.5| ENSG00000139618    | 13         | 32,315,480| 32,399,668|

Figure 2. Example ObjTables-formatted spreadsheet for a dataset of human genes and their splice variants. The first worksheet (a) contains a table of contents for the spreadsheet. The second worksheet (b) describes the schema for the spreadsheet. The schema includes three classes (‘Gene,’ ‘Transcript,’ and ‘Location’) that interact via three relationships (the gene that codes for each splice variant and the location of each gene and transcript). The classes are encoded into two worksheets – one for genes and one for splice variants – by (a) using gene ids to represent the gene that codes for each splice variant and (b) embedding groups of columns for representing the location of each gene and splice variant into the worksheets for genes and transcripts. The data worksheets (c, d) describe the genes and splice variants in the dataset. Supplementary File 1 is an XLSX version of this example.
ObjTables utilizes three mechanisms to encode attributes that represent relationships between objects into spreadsheets. First, schemas can define a primary attribute for each class, and ObjTables can use their values to reference related objects. An object related via a one-to-one or many-to-one relationship can be encoded into the value of its primary attribute. Objects related via a one-to-many or many-to-many relationship can be encoded into a delimited list of the values of the related objects. To ensure these references can be resolved, schemas must also declare each primary attribute to be unique. Second, as described above, one-to-one and many-to-one relationships can be encoded into a group of adjacent columns. Third, ObjTables can use grammars to serialize related objects into a single string-valued column. For example, researchers could use the Nomenclature of Genes, Genetic Markers, Alleles, and Mutations in Mouse and Rat\textsuperscript{28} to encode specific alleles such as the spontaneous Mo allele of Atp7a into ‘Atp7a\textsuperscript{Mo}’ or use the Sequence Variant Nomenclature\textsuperscript{29} to encode sequence variants such as the deletion of thymidine at position g.19 of Homo sapiens dystrophin into ‘NG_012232.1:g.19delT’ rather than using additional worksheets to define alleles and sequence variants.

**Encoding metadata into spreadsheets.** ObjTables can encode multiple levels of metadata into spreadsheets. Metadata about an entire dataset can be encoded into the first cell of the table of contents worksheet as pairs of keys and values. For example, the author of a dataset can be captured by the syntax author=‘John Doe’. Similarly, metadata about a class can be encoded into the first cell of the corresponding worksheet. Metadata about an object can be encoded into an additional row above the row that represents the object. Metadata rows contain a single cell that contains a textual comment between the markup delimiters %/ and /%.

**Encoding schema documentation into spreadsheets.** To best leverage spreadsheet programs such as Microsoft Excel and LibreOffice Calc as editors for ObjTables datasets, to the extent permitted by the XLSX format, ObjTables encodes the type of each attribute and constraints on its values into validations of the corresponding column. For attributes that represent enumerations and one-to-one and many-to-one relationships, this provides users dropdown menus for selecting values. This validation can help users quickly find errors, such as an invalid value of an enumerated attribute. Due to the few validations supported by the XLSX format, ObjTables can only encode limited schema information into spreadsheets.

To help make spreadsheets easy to understand, ObjTables also embeds descriptions of each attribute into notes on their column headings. These notes serve as inline documentation for the schema for the dataset.

**Enhancing the human-readability of spreadsheets.** To make column headings easy to read, ObjTables bolds, shades, and freezes the header row(s) of each worksheet.

### 2.1.2. Format for schemas for structured, human-readable spreadsheets

ObjTables represents datasets as attribute graphs, or graphs of typed objects, where each object and its attributes are represented by a node and each relationship is represented by an edge. For example, the dataset of genes and transcripts in Fig. 2 is composed of instances of three classes (genes, transcripts, and locations) which are linked via three relationships (between transcripts and genes, genes and locations, and transcripts and genes) and which have several attributes (gene and transcript ids, gene symbols, and 3’ and 5’ coordinates).
**ObjTables** provides a simple tabular format for describing the classes that comprise a dataset, the relationships between the classes, the attributes of the classes, and how the classes, relationships, and attributes are encoded into worksheets, rows, and columns. Schema tables contain one row for each class, relationship, and attribute, and have four required and additional optional columns. Fig. 2b shows an example schema for datasets of genes and their splice variants.

**Classes.** The name of each class and the title of the corresponding worksheet are defined via the `!Name` column. Each name must begin with a letter and be composed of letters, numbers, and underscores. The `!Type` column indicates whether each row defines a class (value of `Class`) or relationship or attribute (value of `Attribute`). The `!Parent` column can indicate the superclass of each class. Subclasses inherit their parents' relationships and attributes. The `!Format` column indicates how each class is encoded into spreadsheets; the value `row` indicates that the class is encoded into its own worksheet, the value `multiple_cells` indicates that the class is encoded into groups of columns in the corresponding worksheets for its related classes, and the value `cell` indicates that the class is encoded into a single column in the corresponding worksheets for its related classes using a grammar. The optional `!Verbose name` column can define a more human-readable title for the corresponding worksheet or column heading of each class.

**Relationships and attributes.** The name of each relationship and attribute and the heading of the corresponding column is defined via the `!Name` column. Similar to classes, names must begin with letters and can only include letters, numbers, and underscores. The `!Type` column indicates whether each row defines a class (value of `Class`) or relationship or attribute (value of `Attribute`). The `!Parent` column indicates the parent class of each relationship and attribute. The `!Format` column indicates the type of each relationship and attribute and constraints on their values. The `!Format` column can also indicate the primary attribute of each class, which can be used to encode relationships between objects into spreadsheets. The optional `!Verbose name` column can define more human-readable column headings.

**ObjTables** support four types of relationships: one-to-one (indicated by the format `OneToOne`), one-to-many (`OneToMany`), many-to-one (`ManyToOne`), and many-to-many (`ManyToMany`). Relationship formats have two required arguments that indicate the related class.

To support scientific data, **ObjTables** provides a broad range of types of attributes. This includes attributes for Booleans; integers; floats; strings; dates; times; local files and URLs; emails; arrays; data frames; symbolic mathematical expressions; chemical structures and formulae; DNA, RNA, and protein sequences; and sequence features and motifs. To help researchers annotate scientific data, **ObjTables** also provides attributes for the identifiers of entries in databases, terms in ontologies, units, and uncertainties. For example, researchers could use the attribute type for identifiers to use ChEBI identifiers to describe the metabolites observed in a metabolomics experiment or use the attribute type for ontology terms to use Cell Ontology terms to describe the cell type observed in each experiment.

Researchers can specify constraints on the values of attributes through optional keyword arguments. For example, the integer attribute type supports two optional arguments, `min` and `max`, that can indicate the minimum and maximum valid value of an attribute. More information about the supported attributes and constraints is available at [https://objtables.org/docs](https://objtables.org/docs).
2.1.3. Tools for systematically controlling the quality of data in spreadsheets

To help researchers quality control spreadsheets, the ObjTables software can use schemas to systematically validate datasets. The software supports five levels of validation. First, ObjTables validates that a spreadsheet uses the ObjTables layout conventions and markup syntax. Second, ObjTables checks that the value of each attribute of each object is consistent with the constraints defined in the schema. For example, ObjTables can check that each metabolite has an integer-valued charge and check that each gene has positive 5' and 3' coordinates. Third, ObjTables checks that each relationship encoded using a primary attribute can be decoded. For example, ObjTables can check that the gene that codes for each transcript is defined. Fourth, ObjTables checks that the values of each primary attribute are unique. For example, ObjTables can check that each gene and transcript has a unique id. Fifth, researchers can use the ObjTables Python package to define more holistic validations of entire objects and datasets. For example, researchers can validate that chemical reactions are element-balanced, validate that a chemical reaction network is consistent with thermodynamics and the principle of detailed balance, or validate that a pedigree chart is acyclic.

2.1.4. Programmatically creating, querying, editing and analyzing spreadsheets

To help researchers work with spreadsheets programmatically, the ObjTables Python package can generate high-level data structures and methods for working with the datasets of a schema. (a) The Python package can generate Python classes for representing the datasets of a schema. (b) Researchers can use the methods of these classes to create instances of the classes, get and set their properties, link them to other objects, and find objects within datasets. The Python package can also import and export instances of these classes to and from spreadsheet files. (c) These classes make it easy to use Python to analyze datasets.

2.1.5. Comparing the content of spreadsheets

To help researchers compare datasets that are encoded in the same schema, the ObjTables software can use schemas to determine whether two datasets contain the same content equivalent and identify their differences. ObjTables determines whether two datasets are equivalent by encoding the datasets into attribute graphs, aligning their graph representations, and identifying the differences in the nodes, edges, and attributes of these representations. As described above, ObjTables encodes datasets into graphs by representing each object and its attributes as a node and representing each relationship as an edge. This approach ignores the order of the objects, relationships, and attributes within datasets (e.g., orders of worksheets, rows, and columns), which is often not semantically meaningful. For example, a researcher could use this to compare two reconstructions of the metabolic network of the same organism published by two different researchers.

2.1.6. Merging and splitting spreadsheets

To help researchers integrate data, the ObjTables software can automatically merge and split datasets that are encoded in the same schema. The software can merge datasets by representing datasets as graphs, aligning their nodes and edges, and taking the union of their edges. For example, a researcher could use this to merge separate datasets of intracellular metabolite concentrations, the reactants and products of metabolic reactions, and the kinetic rates of metabolic reactions into a single multi-dimensional dataset.

Conversely, the software can split a dataset by representing it as a graph, cutting a specified
set of edges, and collecting the resulting connected subgraphs. For example, this could help a researcher analyze a specific intracellular pathway within a large dataset of multiple pathways by extracting the information about that pathway from the dataset.

2.1.7. Revisioning and migrating spreadsheets

Complex datasets and their schemas are often developed over time as researchers gather more information and more types of data become available. ObjTables provides two features to help researchers develop datasets iteratively and collaboratively. First, the ObjTables software can help researchers track and manage historical versions of datasets by exporting datasets to CSV or TSV files, committing changes to a version control system such as Git, and merging or identifying conflicts between versions of datasets. Together, this can help a team of researchers work together to develop a dataset. Second, the ObjTables software can help researchers revise a schema and update datasets encoded into the schema by exporting the schema to CSV or TSV file, committing the schema to a version control system, and applying the changes to the schema (e.g., adding, removing, and renaming classes and attributes) to the datasets encoded into the schema. This feature can help researchers in emerging scientific fields develop schemas iteratively as new methodologies and information arise.

2.1.8. Converting spreadsheets to and from alternative formats

In addition to XLSX, the ObjTables software can encode and decode datasets into and out of the comma- and tab-separated values (CSV and TSV), JavaScript Object Notation (JSON), and YAML Ain’t Markup Language (YAML) formats. We recommend using XLSX for viewing, editing, and sharing datasets. We recommend using JSON for importing datasets into programming languages for further analysis. We recommend using CSV or TSV for revisioning datasets because they are the most compatible with version control systems such as Git.

2.1.9. Visualizing the structure of a spreadsheet

To help researchers understand data, the ObjTables software can generate UML diagrams for schemas.

2.1.10. User interfaces

The ObjTables toolkit provides four interfaces: a web application, a command-line program, a REST API, and a Python library. The web application, command-line program, and REST API provide the core features described above for validating, comparing, pretty-printing, and converting datasets and visualizing schemas. In addition to these core features, researchers can use the Python package to implement attributes for additional types; customize how objects and entire datasets are validated; merge and split datasets; revision schemas and datasets; migrate datasets between versions of their schemas; and programmatically construct, edit, query, and analyze datasets. More information about the Python package is available at https://objtables.org/docs.

2.2. Case studies

Through making it easier to structure spreadsheets, we believe that ObjTables can advance a wide range of research. As an example, we illustrate how ObjTables can be used to quality control and integrate information about the kinetics and thermodynamics of Escherichia coli metabolism into a comprehensive model. As a second example, we illustrate how we have used ObjTables to build
a format describing for whole-cell (WC) models. While these tasks could be conducted manually or with custom codes, ObjTables makes these tasks easier and more accessible to a wider range of investigators.

2.2.1. Toward an integrated kinetic-thermodynamic genome-scale model of *Escherichia coli* metabolism

Although metabolism is one of the best-characterized cellular subsystems, we still have limited abilities to predict metabolic phenotypes, such as growth, across genotypes and environments. One of the most promising methods for predicting metabolic phenotypes from genotypes is flux-balance analysis (FBA). However, FBA has limited abilities to make quantitatively accurate predictions due to the lack of quantitative flux constraints. Over the past two decades, researchers have explored a variety of strategies for improving FBA models by incorporating additional constraints based on information such as gene regulation, signal transduction, enzyme abundances, reaction kinetic parameters, and reaction thermodynamics. One potential way to create more accurate models is to combine these constraints. However, this is challenging because these constraints are typically reported via ad hoc spreadsheets due to the lack of a suitable standard.

Toward a more quantitatively predictive model of *E. coli* metabolism, we used ObjTables to merge kinetic and thermodynamic parameters of *E. coli* metabolic reactions published in ad hoc spreadsheets by Khodayari and Maranas and Gerosa et al., respectively and map the parameters onto the latest genome-scale model of *E. coli* metabolism. First, we designed schemas for the spreadsheets which include foreign key constraints between worksheets, grammars for reaction equations, and tab-separated tables embedded into cells in several columns. Second, we used the schemas and the ObjTables software to identify and correct errors in the spreadsheets such as swapped column headings, typos in enumerations, syntactically invalid reaction equations, invalid foreign key references, repeated rows, and misaligned columns. Third, we used the ObjTables software to automatically identify potentially semantically meaningful differences between the compartments, metabolites, and reactions represented by the Khodayari and Gerosa spreadsheets. Fourth, we manually reviewed the semantic meaning of the differences between the spreadsheets identified by ObjTables and manually aligned the datasets. For example, we aligned the identifiers of metabolites that we determined were semantically equivalent, and we aligned the canonical directions of reversible reactions that we determined were semantically equivalent. Once we aligned the datasets, it was trivial to join the datasets by merging pairs of semantically equivalent compartments, metabolites, and reactions. While we could have merged the datasets manually or with custom code, this would have taken significantly more effort. In particular, chemical formula and reaction equations cannot simply be compared by comparing their string representations because their string representations are not unique (e.g., the order of the atoms in a chemical formula has no semantic meaning). Rather, chemical formula and reaction formulae must be compared using algorithms that recognize their semantic meaning such as those implemented by ObjTables. In contrast, it only took a few seconds to use the ObjTables software to identify the semantically meaningful differences between the Khodayari and Gerosa datasets. Next, we examined the relationship between the kinetic parameters reported by Khodayari and the thermodynamic parameters reported by Gerosa (Fig. 3a–h). The lack of correlation between the kinetic and thermodynamic data confirmed that kinetic and thermodynamic parameters contain distinct information, and that kinetic and thermodynamic parameters could be used in combination to create a better constrained and more predictive model. Finally, we outlined an integrated kinetic-thermodynamic
model of *E. coli* metabolism by mapping the kinetic and thermodynamic parameters onto the latest genome-scale model of *E. coli* metabolism.41

This example illustrates how *ObjTables* can enhance the value of spreadsheets by making it easier to quality control, parse complex spreadsheets, compose, and analyze spreadsheets.

2.2.2. Format for composite, multi-algorithmic whole-cell (WC) models

The goal of whole-cell (WC) modeling1,2 is to develop models that can predict cellular phenotypes from their genotypes and environments. Achieving WC models will likely require a large collaborative effort. One of the most promising ways to build models collaboratively is to combine submodels of separate cellular pathways developed by different researchers.

To properly merge submodels, researchers must verify that the submodels capture the same biology with compatible assumptions, identify and fuse the common species, and remove any redundant reactions. Executing this at the scale required for WC modeling, requires structured semantic information about the chemical identity of each species and structured provenance information.
Figure 4. *ObjTables* can help researchers build formats for domain-specific data such as WC-Lang, a spreadsheet format for describing the mathematics, semantics, and provenance of whole-cell models. WC-Lang involves 23 worksheets, 38 classes, 157 relationships, 24 types of attributes, and 3 grammars. This diagram indicates the key classes of WC-Lang and their relationships. Red indicates the top-level class that represents models; green indicates classes that map to separate worksheets; blue indicates classes that map to individual columns or groups of columns of spreadsheets for their parent classes. A diagram for the complete schema is available at https://www.objtables.org/docs.

about the data sources and assumptions behind each submodel.

To facilitate collaboration, we have used *ObjTables* to develop WC-Lang (https://github.com/KarrLab/wc_lang), a format that enables researchers to (a) precisely describe submodels of individual pathways, their semantic meaning, and their provenance and (b) combine submodels into a single multi-algorithmic model. The schema for WC-Lang involves 23 worksheets, 38 classes, 157 relationships, 24 types of attributes, and three grammars (Fig. 4), demonstrating that *ObjTables* can manage large schemas.

We used *ObjTables* to implement WC-Lang for several reasons. (a) *ObjTables* enabled us to create a spreadsheet-based format, which we anticipate will enable modelers to quickly edit thousands of model elements, as well as make it easy for experimentalists to contribute to models. (b) *ObjTables*’ graph alignment methods make it easy to merge submodels with minimal code. (c) By providing high-level data structures and methods for manipulating models, *ObjTables* makes it easy to implement multi-algorithmic simulations.

3. Discussion

As discussed above, the *ObjTables* toolkit structures human-readable spreadsheets by combining them with schemas, an object relational mapping system, and software tools for using these schemas to parse, validate, and analyze spreadsheets. This enables researchers to leverage the simplicity of spreadsheets, the rigor of schemas, the high-level semantics of object-oriented data structures, and the power of imperative programming. Researchers can use user-friendly programs such as Microsoft Excel and LibreOffice Calc to view and edit spreadsheets and use schemas and the *ObjTables* software to validate, compare, merge, split, revision, migrate, and analyze spreadsheets.
To avoid barriers that could deter researchers from creating structured, reusable spreadsheets, ObjTables schemas and datasets can be read and written with any spreadsheet program such as Microsoft Excel or LibreOffice Calc, without any additional plugin, macro, or other software. We hope that the low barrier to using ObjTables will facilitate its adoption.

Despite its limitations, ObjTables builds upon on spreadsheets because they are one of the most common mediums for sharing data. In particular, spreadsheets are one of the most common formats for supplementary data to journal articles. Consequently, we believe that ObjTables has the potential to systemize a substantial fraction of our scientific data.

In the short-term, we expect that ObjTables will help increase the quality and transparency of scientific spreadsheets by helping authors validate their datasets and communicate their structures to others. In turn, this will make it easier for other researchers to reanalyze and extract additional insights from published datasets. Long-term, once there is a substantial number of structured spreadsheets, we anticipate that the ObjTables software tools make it easy for researchers to integrate datasets, which will enable more extensive and more comprehensive datasets that can help researchers gain new insights into complex systems through unprecedented meta-analyses. For example, we anticipate that ObjTables will help systems biologists assemble diverse information into global biochemical networks of entire cells. Additionally, we believe that ObjTables can accelerate emerging scientific fields by making it easier for emerging domains to develop new formats for new kinds of information.

### 3.1. Incorporating additional layout conventions and data types to better support specific fields of science

Realizing the full potential of ObjTables as a platform for exchanging, comparing, and composing data will likely require support for additional worksheet layouts and additional types of attributes and grammars that make it easy to encode additional types of data into spreadsheets. For example, multi-level headings could make it easier to represent to some types of data, attribute types and grammars for genetic variants and alleles would make it easier to represent genetic information, and attribute types and grammars for geographic and spatial coordinates would make it easier to represent geoscience information. We encourage users to share suggestions for additional layouts by posting issues to the ObjTables Git repository. To expand the capabilities of ObjTables for additional fields of science, we plan to help researchers implement additional types of attributes, encourage researchers to contribute additional types of attributes to the ObjTables Git repository, and periodically release new versions of ObjTables as we incorporate additional attributes.

### 3.2. Developing a registry of spreadsheet schemas to facilitate further standardization

Automatically comparing and composing data with ObjTables also requires authors to use the same classes to represent their data. To encourage communities to align on schemas for representing similar types of data, we hope to develop a registry of schemas. The registry would also make ObjTables easier to use by helping researchers reuse existing schemas rather than developing their own schemas. To start, we could implement the registry as a Git repository and accept submissions via Git pull requests.
3.3. Reusing existing spreadsheets with \textit{ObjTables}

\textit{ObjTables} enables researchers going forward to create reusable datasets. \textit{ObjTables} can also help researchers reuse and compose the large existing body of unstructured spreadsheets. To reuse an existing spreadsheet, a researcher must first review the spreadsheet and design a schema. Second, the researcher must restructure the spreadsheet to match the schema and annotate each worksheet and column. Once restructured, the researcher will be able to use the spreadsheet with the \textit{ObjTables} software.

3.4. Encouraging community adoption of \textit{ObjTables} as a meta-standard for structured spreadsheets

Ultimately, realizing the full potential of \textit{ObjTables} as a platform for comparing and composing data will require acceptance by the scientific community. We have begun to advertise \textit{ObjTables} by registering \textit{ObjTables} with the bio.tools registry of software for biology\cite{bib:biotools:objtables}, the BioCatalogue registry of web services\cite{bib:biocatalogue.service:4016}, the EMBRACE Data And Methods (EDAM) ontology of formats,\cite{bib:edam:objtables} the FAIRsharing registry of formats,\cite{bib:fairsharing:objtables} the SciCrunch registry of digital scientific resources (rrid:SCR_018652), and the TESS registry of biology tutorials\cite{bib:teds:objtables} and by sending announcements to community mailing lists. Currently, we encourage users to provide input through GitHub issues or pull requests. Long-term, we aspire to form a committee to govern \textit{ObjTables}. Long-term, we also aim to push the community to support reusable datasets by encouraging journals to require supplementary tables to be submitted in a structured format such as \textit{ObjTables}.

4. Methods

4.1. Implementation of the \textit{ObjTables} software

We implemented the \textit{ObjTables} software tools in Python. We implemented reading and writing CSV, TSV, XLSX, and YAML files with OpenPyXL, pyexcel, PyYAML, and XlsxWriter. We used Lark to implement support for grammars. We implemented the mathematics, science, chemistry, and biology attributes using the Biopython,\cite{bib:biopython} BpForms,\cite{bib:bpforms} BcForms,\cite{bib:bcforms} NumPy, Open Babel,\cite{bib:openbabel} Pint, Pronto, SymPy,\cite{bib:sympy} and Uncertainties packages. We implemented the revisioning and migration features using GitPython. We implemented the schema visualization feature using GraphViz. We implemented the web application, command-line program, and REST API using Zurb Foundation, cement, and Flask-RESTPlus, respectively.

4.2. Testing of the \textit{ObjTables} software

We used unittest to implement extensive unit tests of the \textit{ObjTables} software with over 98\% coverage. Furthermore, we used CircleCI and pytest to execute the tests each time we revised the \textit{ObjTables} source code. We assessed the line coverage of the tests using coverage and Coveralls.

4.3. Data availability

The spreadsheets for the case studies are available in CSV, TSV, JSON, XLSX, and YAML formats at https://objtables.org/docs.
4.4. Code availability

4.4.1. Software

ObjTables is released open-source under the MIT license. Additionally, a license to ChemAxon Marvin, which is freely available to academic researchers, is needed to execute some of the methods of the chemistry attributes.

The web application is available at https://objtables.org/app. The REST API is available at https://objtables.org/api. The command-line program and Python package are available at https://pypi.org/project/obj-tables. A Dockerfile for building a Docker image and the source code are available at https://github.com/KarrLab/obj_tables.

4.4.2. Examples, tutorials, and documentation

Documentation for the format for schemas, data types, and markup syntax for spreadsheets and examples are available at https://objtables.org. Documentation for the command-line program and REST API are available inline. Interactive tutorials and documentation for the Python package are available at http://sandbox.karrlab.org and http://docs.karrlab.org.

4.4.3. Version information

This manuscript describes version 1.0.0 of ObjTables.

Acknowledgements

We thank Yin Hoon Chew, Timo Lubitz, and Elad Noor for valuable input and feedback. This work was supported by National Institutes of Health grants R35GM119771 and P41EB023912 and National Science Foundation grant 1649014 to J.R.K and German Research Foundation grant Li 1676/2-2 to W.L.

Author information

Affiliations

Icahn Institute for Data Science and Genomic Technology and Department of Genetics and Genomic Sciences, Icahn School of Medicine at Mount Sinai, New York, NY 10029, USA
Jonathan R. Karr, Arthur P. Goldberg, John A. P. Sekar & Bilal Shaikh

Université Paris-Saclay, INRAE, MalAGE, 78350, Jouy-en-Josas, France
Wolfram Liebermeister

Contributions

J.R.K. conceived of the project, designed the formats, developed the software, developed the case studies, and wrote the manuscript. A.P.G. developed parts of the software, including the migration feature. W.L. designed the format for schemas and the markup syntax for spreadsheets. J.A.P.S. developed the biology attributes. B.S. developed the schema visualization software. All of the authors contributed to and approved this manuscript.
Corresponding author
Correspondence to Jonathan Karr.

Ethics declarations

Competing interests
The authors declare no competing interests.

Supplementary information

Supplementary File 1: XLSX version of the example spreadsheet for a dataset of human genes illustrated in Fig. 2.

References

1. Karr, J. R. et al. A whole-cell computational model predicts phenotype from genotype. *Cell* **150**, 389–401 (2012).

2. Goldberg, A. P. et al. Emerging whole-cell modeling principles and methods. *Curr. Opin. Biotechnol.* **51**, 97–102 (2018).

3. Lipman, D. J. & Pearson, W. R. Rapid and sensitive protein similarity searches. *Science* **227**, 1435–1441 (1985).

4. Cock, P. J., Fields, C. J., Goto, N., Heuer, M. L. & Rice, P. M. The Sanger FASTQ file format for sequences with quality scores, and the Solexa/Illumina FASTQ variants. *Nucleic Acids Res.* **38**, 1767–1771 (2010).

5. Li, H. et al. The sequence alignment/map format and SAMtools. *Bioinformatics* **25**, 2078–2079 (2009).

6. Sayers, E. W. et al. GenBank. *Nucleic Acids Res.* **47**, D94–D99 (2019).

7. Kodama, Y., Shumway, M. & Leinonen, R. The Sequence Read Archive: explosive growth of sequencing data. *Nucleic Acids Res.* **40**, D54–D56 (2012).

8. McKenna, A. et al. The Genome Analysis Toolkit: a MapReduce framework for analyzing next-generation DNA sequencing data. *Genome Res.* **20**, 1297–1303 (2010).

9. Vision, T. The Dryad Digital Repository: Published evolutionary data as part of the greater data ecosystem. *Nat. Preced.* 1–1 (2010).

10. Wolstencroft, K. et al. FAIRDOMHub: a repository and collaboration environment for sharing systems biology research. *Nucleic Acids Res.* **45**, D404–D407 (2017).

11. Croll, G. The importance and criticality of spreadsheets in the city of london. In *Proc. Eur. Spreadsheets Risk Interest Group Annu. Conf.* (2005).
12. Bryan, J. Spreadsheets. In *csv.conf* (2016).

13. Rajalingham, K. A revised classification of spreadsheet errors. In *Proc. Eur. Spreadsheets Risk Interest Group Annu. Conf.* (2005).

14. Caulkins, J. P., Morrison, E. L. & Weidemann, T. Spreadsheet errors and decision making: Evidence from field interviews. *J. Organ. End User Comput.* **19**, 1–23 (2007).

15. Powell, S. G., Baker, K. R. & Lawson, B. A critical review of the literature on spreadsheet errors. *Decision Support Systems* **46**, 128–138 (2008).

16. Ziemann, M., Eren, Y. & El-Osta, A. Gene name errors are widespread in the scientific literature. *Genome Biol.* **17**, 177 (2016).

17. Creek, D. J., Jankevics, A., Burgess, K. E., Breitling, R. & Barrett, M. P. IDEOM: an Excel interface for analysis of LC–MS-based metabolomics data. *Bioinformatics* **28**, 1048–1049 (2012).

18. Sansone, S.-A. *et al.* The first RSBI (ISA-TAB) workshop: “can a simple format work for complex studies?”. *OMICS* **12**, 143–149 (2008).

19. Rayner, T. F. *et al.* A simple spreadsheet-based, MIAME-supportive format for microarray data: MAGE-TAB. *BMC Bioinformatics* **7**, 489 (2006).

20. Lubitz, T. *et al.* SBtab: a flexible table format for data exchange in systems biology. *Bioinformatics* **32**, 2559–2561 (2016).

21. Fowler, D., Barratt, J. & Walsh, P. Frictionless data: Making research data quality visible. *Int. J. Digital Curation* **12**, 274–285 (2017).

22. Bendre, M. *et al.* Dataspread: Unifying databases and spreadsheets. *Proceedings VLDB Endowment* **8**, 2000–2003 (2015).

23. Tyszkiewicz, J. Spreadsheet as a relational database engine. In *Proc. ACM SIGMOD Int. Conf. Management Data*, 195–206 (2010).

24. Cunha, J., Saraiva, J. & Visser, J. From spreadsheets to relational databases and back. In *Proc. ACM SIGPLAN Workshop Partial Evaluation Program Manipulation*, 179–188 (2009).

25. Pedregosa, F. *et al.* scikit-learn: machine learning in Python. *J. Mach. Learn. Res.* **12**, 2825–2830 (2011).

26. Ecma International. Standard ECMA-376: Office Open XML file formats. https://www.ecma-international.org/publications/standards/Ecma-376.htm (2016).

27. International Organization for Standardization. ISO/IEC 29500-1:2016: Information technology – Document description and processing languages — Office Open XML file formats. https://www.iso.org/standard/71691.html (2016).

28. International Committee on Standardized Genetic Nomenclature for Mice. Guidelines for Nomenclature of Genes, Genetic Markers, Alleles, and Mutations in Mouse and Rat. http://www.informatics.jax.org/mgihome/nomen/gene.shtml (2018).
29. den Dunnen, J. T. et al. HGVS recommendations for the description of sequence variants: 2016 update. *Hum. Mutat.* **37**, 564–569 (2016).

30. Hastings, J. et al. ChEBI in 2016: Improved services and an expanding collection of metabolites. *Nucleic Acids Res.* **44**, D1214–D1219 (2016).

31. Diehl, A. D. et al. The Cell Ontology 2016: enhanced content, modularization, and ontology interoperability. *J. Biomed. Semantics* **7**, 44 (2016).

32. Ederer, M. & Gilles, E. D. Thermodynamic constraints in kinetic modeling: thermodynamic-kinetic modeling in comparison to other approaches. *Eng. Life Sci.* **8**, 467–476 (2008).

33. Orth, J. D., Thiele, I. & Palsson, B. Ø. What is flux balance analysis? *Nat. Biotechnol.* **28**, 245–248 (2010).

34. Covert, M. W., Schilling, C. H. & Palsson, B. Regulation of gene expression in flux balance models of metabolism. *J. Theor. Biol.* **213**, 73–88 (2001).

35. Covert, M. W., Xiao, N., Chen, T. J. & Karr, J. R. Integrating metabolic, transcriptional regulatory and signal transduction models in escherichia coli. *Bioinformatics* **24**, 2044–2050 (2008).

36. Adadi, R., Volkmer, B., Milo, R., Heinemann, M. & Shlomi, T. Prediction of microbial growth rate versus biomass yield by a metabolic network with kinetic parameters. *PLoS Comput. Biol.* **8** (2012).

37. Desouki, A. *Algorithms for improving the predictive power of flux balance analysis*. Ph.D. thesis, Universität Paderborn (2016).

38. Hoppe, A., Hoffmann, S. & Holzhütter, H.-G. Including metabolite concentrations into flux balance analysis: thermodynamic realizability as a constraint on flux distributions in metabolic networks. *BMC Syst. Biol.* **1**, 23 (2007).

39. Khodayari, A. & Maranas, C. D. A genome-scale escherichia coli kinetic metabolic model k-ecoli457 satisfying flux data for multiple mutant strains. *Nat. Commun.* **7**, 1–12 (2016).

40. Gerosa, L. et al. Pseudo-transition analysis identifies the key regulators of dynamic metabolic adaptations from steady-state data. *Cell Syst.* **1**, 270–282 (2015).

41. Orth, J. D. et al. A comprehensive genome-scale reconstruction of escherichia coli metabolism—2011. *Mol. Syst. Biol.* **7** (2011).

42. Ison, J. et al. The bio. tools registry of software tools and data resources for the life sciences. *Genome Biol.* **20**, 1–4 (2019).

43. Bhagat, J. et al. BioCatalogue: a universal catalogue of web services for the life sciences. *Nucleic Acids Res.* **38**, W689–W694 (2010).

44. Ison, J. et al. EDAM: an ontology of bioinformatics operations, types of data and identifiers, topics and formats. *Bioinformatics* **29**, 1325–1332 (2013).

45. Sansone, S.-A. et al. FAIRsharing as a community approach to standards, repositories and policies. *Nat. Biotechnol.* **37**, 358–367 (2019).
46. Beard, N. et al. Tess: a platform for discovering life-science training opportunities. *Bioinformatics* **36**, 3290–3291 (2020).

47. Cock, P. J. et al. Biopython: freely available Python tools for computational molecular biology and bioinformatics. *Bioinformatics* **25**, 1422–1423 (2009).

48. Lang, P. F. et al. BpForms and BcForms: tools for concretely describing non-canonical polymers and complexes to facilitate comprehensive biochemical networks. *Genome Biol.* (Accepted).

49. O’Boyle, N. M. et al. Open Babel: an open chemical toolbox. *J. Cheminform.* **3**, 33 (2011).

50. Meurer, A. et al. SymPy: symbolic computing in Python. *PeerJ Comput. Sci.* **3**, e103 (2017).