SciSheets: Delivering the Power of Programming With The Simplicity of Spreadsheets

Alicia Clark[§], Joseph Hellerstein[‡*]

Abstract—Short abstract.

Index Terms—software engineering

1. Introduction

Digital spreadsheets are the "killer app" that ushered in the PC revolution. This is largely because spreadsheets provide a conceptually simple way to do calculations that (a) closely associates data with the calculations that produce the data and (b) avoids the mental burdens of programming such as control flow, data dependencies, and data structures. Estimates suggest that over 800M professionals author spreadsheet formulas as part of their work [MODE2017], which is about 50 times the number of software developers world wide [Thib2013].

We categorize spreadsheet users as follows:

- **Calcers** want to evaluate equations. Spreadsheet formulas work well for Calcers since: (a) they can ignore data dependencies; (b) they can avoid flow control by using "copy" and "paste" for iteration; and (c) data structures are "visual" (e.g., rectangular blocks).
- **Scripters** feel comfortable with expressing calculations algorithmically using `for` and `if` statements; and they can use simple data structures such as lists and `pandas DataFrames` (which are like spreadsheets). However, they rarely encapsulate code into functions, preferring to `copy` and `paste` code to get reuse.
- **Programmers** know about sophisticated data structures, modularization, reuse, and testing.

Our experience is primarily with scientists, especially biologists and chemists. Most commonly, we encounter Calcers and Scripters. Only Programmers take advantage of spreadsheet macro capabilities (e.g., Visual Basic for Microsoft Excel or AppScript in Google Sheets).

Based on this experience, we find existing spreadsheets lack several key requirements. The first requirement is **Expressivity**. Existing spreadsheets only support formulas that are expressions, not scripts. This is significant limitation for Scripters who often want to express calculations as algorithms. It is also a burder for Calcers who want to write linear workflows to articulate a computational recipe, a kind a computational laboratory notebook. A second requirement not addressed in today's spreadsheets is **Reuse**. Specifically, it is impossible to reuse spreadsheet formulas in other spreadsheet formulas or in software systems. A third requirements is that spreadsheets handle **Complex Data**. For example, today's spreadsheets make it extremely difficult to handle hierarchically structured data and n-to-m relationships. A final requirement we consider is **Performance**. A common complaint is that spreadsheets scale poorly with the size of data and the number of formulas.

Academic computer science has recognized the growing importance of end-user programming (EUP) [BURN2009]. Even so, there is little academic literature on spreadsheets. [MCLU2006] discusses object oriented spreadsheets that introduces a sophisticated object model but fails to recognize the requirements of Calcers to have a simple way to evaluate equations. [JONE2003] describes a way that users can implement functions within a spreadsheet to get reuse, but the approach requires considerable user effort and does not address reuse of spreadsheet formulas in a larger software system. Outside of academia there has been significant interest in innovating spreadsheets. Google Fusion Tables [Gonz2010] and the "Tables" feature of Microsoft Excel ref?? use column formulas to avoid a common source of errors, the need to copy formulas as rows are added/deleted from a table. The Pyspread [PySpread] project uses Python as the formula language, which gives formulas access to thousands of Python packages. A more radical approach is taken by Stencila [Stencila], a document system that provides ways to execute code that updates tables (an approach that is in the same spirit as Jupyter Notebooks [Pere2015]). Stencila supports a variety of languages including JavaScript, Python, and SQL. However, Stencila lacks features that spreadsheet users expect: (a) closely associating data with the calculations that produce the data and (b) avoiding considerations of data dependencies in calculations.

This paper introduces SciSheets [SciSheets], a new spreadsheet system with the objective of delivering the power of programming with the simplicity of spreadsheets. The name SciSheets is a contraction of the phrase "Scientific Spreadsheet", a nod to the users who motivated the requirements that we address. That said, our target users are more broadly technical professionals who do complex calculations on struc-

§ eScience Institute, University of Washington
∗ Corresponding author: joseph.hellerstein@gmail.com
‡ Department of Mechanical Engineering, University of Washington

tured data. We use the term **scisheet** to refer to a SciSheets spreadsheet. We note in passing that our focus for scisheets is on calculations, not document processing features such as formatting and drawing figures.

SciSheets addresses the above requirements by introducing several novel features.

- *Formula Scripts* Scisheet formulas can be Python scripts, not just expressions. This addresses the expressivity requirement since calculations can be expressed as algorithms.
- *Program Export.* Scisheets can be exported as standalone Python programs. This addresses the reuse requirement since exported spreadsheets can be reused in SciSheets formulas and/or by external programs (e.g., written by Programmers). Further, performance is improved by the export feaure since calculations can execute without the overheads of the spreadsheet environment.
- *Subtables.* Tables can have columns that are themselves tables (columns within columns). This addresses the complex data requirement, such as representing n-to-m relationships.

The remainder of the paper is organized as follows. Section 2 describes the requirements that we consider, and section 3 details the SciSheets features that address these requirements. The design of SciSheets is discussed in Section 4, and section 5 discusses features planned for SciSheets. Our conclusions are presented in Section 6.

2. Requirements

This section present examples that motivate the requirements of expressivity, reuse, and complex data.

Our first example is drawn from biochemistry labs studying enzyme mediated chemical reactions. Commonly, the Michaelis-Menten ref?? model of enzyme activity is used in which there is a single chemical species, called the substrate, that interacts with the enzyme to produce a new chemical species (the product). Two properties of enzymes are of much interest: the maximum reaction rate, denoted by V_{MAX}, and the concentration K_M of substrate that achieves a reaction rate equal to half of V_{MAX}.

To perform the Michaelis-Menten analysis, laboratory data are collected for different values of the substrate concentration S and reaction rate V. Then, a calculation is done to obtain the parameters V_{MAX} and K_M using the following recipe.

1. Compute :math:`1/S` and :math:`1/V`, the inverses of S and V.
2. Compute the intercept and slope of the regression of :math:`1/V` on :math:`1/S`.
3. Calculate :math:`V_{MAX}` and :math:`K_M` from the intercept and slope.

	A	B	C	D	E	F	G	H
1	S	V	INV_S	INV_V	INTERCEPT	SLOPE	V_MAX	K_M
2	0.01	0.11	100.00	9.09	4.357	0.047	0.229	0.011
3	0.05	0.19	20.00	5.26				
4	0.12	0.21	8.33	4.76				
5	0.20	0.22	5.00	4.55				
6	0.50	0.21	2.00	4.76				
7	1.00	0.24	1.00	4.17				

Fig. 1: *Data view for an Excel spreadsheet that calculates Michaelis-Menten Parameters.*

	A	B	C	D	E	F	G	H
1	S	V	INV_S	INV_V	INTERCEPT	SLOPE	V_MAX	K_M
2	0.01	0.11	=1/A2	=1/B2	=INTERCEPT(D2:D7,C2:C7)	=SLOPE(D2:D7,C2:C7)	=1/E2	=F2*G2
3	0.05	0.19	=1/A3	=1/B3				
4	0.12	0.21	=1/A4	=1/B4				
5	0.20	0.22	=1/A5	=1/B5				
6	0.50	0.21	=1/A6	=1/B6				
7	1.00	0.24	=1/A7	=1/B7				

Fig. 2: *Formulas used in Fig. 1.*

Fig. 1 shows an Excel spreadsheet that implements this recipe with column names chosen to correspond to the variables in the recipe. Fig. 2 shows the formulas that perform these calculations. Readability can be improved by using column formulas (e.g., as in Fusion Tables). However, two problems remain. Calcers cannot make an *explicit* statement of the computational recipe; rather, it is implicit in the order of the columns. Even more serious, there is no way to reuse these formulas in other formulas (other than error-prone copy-and-paste), and there is no way to reuse in an external program.

We consider a second example to illustrate problems with handling non-trivial data relationships in spreadsheets. Fig. 3 displays data that a university might have for students in two departments in the School of Engineering. The data are organized into two tables (CSE and Biology) grouped under the School of Engineering, with separate columns for student identifiers and grades. These tables are adjacent to each other to facilitate the comparisons between students. However, the tables are independent of each other in that we should be able to insert, delete, and hide rows in one table without affecting the other table. Unfortunately, existing spreadsheet systems do not handle this well in that adding a row to one table affects all tables on that row in the sheet. Note that arranging the tables vertically does not help since now the problem becomes adding, deleting, or hiding columns. (We could arrange the tables in a diagonal, but this makes it difficult to make visual comparisons between tables becomes.)

3. Features

3.1 User Interface

1. Elements - sheet, tables, columns, rows, cells (Fig)
2. Row column - unique ID (name) for row
3. Common popup menus for sheet, table, column, row: insert, delete, hide/unhide, rename (for row,

	A	B	C	D	E	F
1	**Engineering - CSE**			**Engineering - Biology**		
2	ScholarID	GradePtAvg		StudentNo	Track	GPA
3	C1113	3.9		B1414	A	3.4
4	C1163	3.5		B1830	B	2.3
5	C1344	3.3		B1716	C	3.7
6	C1711	3.9				
7	C1579	2.8				

Fig. 3: *Student grade data from two departments in the school of engineering. CSE and Biology are separate tables that are grouped together for convenience of analysis. However, it is difficult to manage them separate, such as insert, delete, and/or hide rows.*

Append
Delete
Formula
Hide
Insert
Move
Refactor
Rename
Tablize
Unhide

MichaelisMenten ... haelis_menten_scipy)

row	S	V	*INV_S	*		CEPT	*SLOPE	*V_MAX	*K_M
1	0.01	0.11	100.0	9			0.047	0.229	0.011
2	0.05	0.19	20.0	5					
3	0.12	0.21	8.33	4.76					
4	0.2	0.22	5.0	4.55					
5	0.5	0.21	2.0	4.76					
6	1.0	0.24	1.0	4.17					

Fig. 4: *Column popup menu in a scisheet for the Michaelis-Menten calculation.*

INV_S

```
1/S
```

Fig. 5: *Formula for computing the inverse of the input value S.*

moves the row)
4. Cell - edit
5. Column: formula
6. Table: prologue, epilogue
7. scisheet: saveas, undo/redo, export

3.2 Formula Scripts

1. Column Variables. Column names are pandas array. Referred to as **Column Variables**. Means that vector operations are supported, natural for Calcer. Also, handles missing data.
2. Challenges with formula evaluation because of arbitrary code.
3. Workflow for table evaluation
   a. Prolog - initialize Column Variables from the table. If there is no exception, then control continues to formula evaluation. Otherwise an exception is raised.

INV_V

```
import scipy.stats as ss
INV_S = np.round(1/S, 2)
INV_V = np.round(1/V, 2)
SLOPE, INTERCEPT, _, _, _ = ss.linregress(INV_S, INV_V)
V_MAX = 1/INTERCEPT
K_M = SLOPE*V_MAX
```

Fig. 6: *Formula for the complete calculation of V_{MAX} and K_M. The formula is a simple script, allowing a Calcer to see exactly how the data in the scisheet are produced. Note that the formula assigns values to other columns.*

row	CSV_FILE	*K_M	V_MAX
1	Glu.csv	[5.179]	[0.568]
2	LL-DAP.csv	[0.929]	[23.81]
3	THDPA.csv	[0.011]	[0.229]

Fig. 7: *A scisheet that processes many CSV files.*

K_M

```
# Compute K_M and V_MAX for each CSV file
K_M = []
V_MAX = []
for csv_file in CSV_FILE:
  df = pd.read_csv(join(PATH, csv_file))
  s_val = df['S']
  v_val = df['V']
  v_max, k_m = michaelis(s_val, v_val)
  K_M.append(k_m)
  V_MAX.append(v_max)
```

Fig. 8: *Column formula that is a script to process CSV files.*

Table Export
Function name: michaelis
List of input columns: S, V
List of output columns: V_MAX, K_M
Submit Cancel

MichaelisMenten (Table File: michaelis_menten_demo)

row	S	V	*INV_S	*INV_V	*INTERCEPT	SLOPE	*V_MAX	*K_M
1	0.01	0.11	100.0	9.09	4.358	0.047	0.229	0.011
2	0.05	0.19	20.0	5.26				
3	0.12	0.21	8.33333333333	4.76				
4	0.2	0.22	5.0	4.55				
5	0.5	0.21	2.0	4.76				
6	1.0	0.24	1.0	4.17				

Fig. 9: *Menu to export a table as a standalone python program.*

   b. Formula evaluation loop. Evaluate each column formula until one of the following holds:
      a) All Column Variables have the same value in two successive iterations of the formula evaluation loop and there is no exception.
      b) A specified number of iterations has occurred. The number of iterations is equal to the number of formula columns. If there is no exception, then control continues to the Epilogue. Otherwise an exception is raised.
   c. Epilogue. Evaluate the Epilogue formula. If no exception occurs, update the column values.

3.3. Program Export

```
# Function definition
def michaelis(S, V):
  from scisheets.core import api as api
  s = api.APIPlugin('michaelis.scish')
  s.initialize()
  _table = s.getTable()
```

Prologue

```
#
  s.controller.startBlock('Prologue')
  # Begin Prologue
  import math as mt
  import numpy as np
  from os import listdir
  from os.path import isfile, join
  import pandas as pd
```

```
  import scipy as sp
  from numpy import nan  # Must follow sympy import
  # End Prologue
  s.controller.endBlock()

#
  # Loop initialization
  s.controller.initializeLoop()
  while not s.controller.isTerminateLoop():
    s.controller.startAnIteration()

#
    # Formula evaluation blocks
    try:
      # Column INV_S
      s.controller.startBlock('INV_S')
      INV_S = 1/S
      s.controller.endBlock()
      INV_S = s.coerceValues('INV_S', INV_S)
    except Exception as exc:
      s.controller.exceptionForBlock(exc)

#
    # Close of function
    s.controller.endAnIteration()

  if s.controller.getException() is not None:
    raise Exception(s.controller.formatError(
        is_absolute_linenumber=True))

  s.controller.startBlock('Epilogue')
  # Epilogue
  s.controller.endBlock()

  return V_MAX,K_M
```

Tests

```
from scisheets.core import api as api
from michaelis import michaelis
import unittest

#############################
# Tests
#############################
# pylint: disable=W0212,C0111,R0904
class Testmichaelis(unittest.TestCase):

  def setUp(self):
    from scisheets.core import api as api
    self.s = api.APIPlugin('michaelis.scish')
    self.s.initialize()
    _table = self.s.getTable()

  def testBasics(self):
    # Assign column values to program variables.
    S = self.s.getColumnValue('S')
    V = self.s.getColumnValue('V')
    V_MAX,K_M = michaelis(S,V)
    self.assertTrue(
        self.s.compareToColumnValues('V_MAX', V_MAX))
    self.assertTrue(
        self.s.compareToColumnValues('K_M', K_M))

if __name__ == '__main__':
  unittest.main()
```

3.4. Subtables

4. Design

To enable a zero-install deployment and leverage the rapid pace of UI innovation happening with web technologies, SciSheets is a client-server application in which the front end uses HTML and Javascript; tables are rendered using YUI DataTables ref??. The backend handles the bulk of the

	Engineering						
	[CSE]			[Biology]			
row	row	*ScholarID	GradePtAvg	row	*StudentNo	Track	GPA
	1	C1113	3.9	1	B1414	A	3.4
	2	C1163	3.5	2	B1830	B	2.3
	3	C1344	3.3	3	B1716	C	3.7
	4	C1711	3.9				
	5	C1579	2.8				

Fig. 10: A table with two subtables. Subtables CSE and Biology can be manipulated separately, providing a way to express n-to-m relationships.

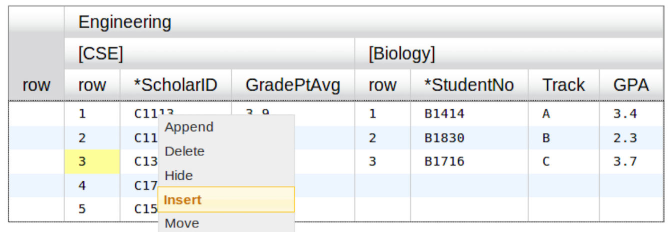

Fig. 11: Menu to insert a row in one subtable. The menu was accessed by left-clicking on the "3" cell in the column labelled "row" in the CSE subtable.

computing tasks (e.g., formula evaluation). We connect the frontend and backend using Django ref??.

Fig 13 displays the relationships between core classes used in the SciSheets backend.

The use casses create the following requirements: (a) SciSheets must perform calculations without prior knowledge of data dependencies between columns; and (b) column formulas may be arbitrary Python scripts. The implies that *SciSheets cannot use a static analysis to discover data dependencies between columns* (as is possible in a traditional spreadsheet). To see the issue here, note that a formula may contain an `eval` statement on a string variable whose value cannot be determined until runtime. Another example is that a formula may call an external function that changes values in columns.

A second implication follows from (b); this relates to debuggability. Specifically, since a formula may be a script consisting of many lines, syntax errors and exceptions must localize the problem to a line within the script. We refer to this as the **Script Debuggability** requirement.

We begin with our approach to handling data dependencies. Our solution is ...

	Engineering						
	[CSE]			[Biology]			
row	row	*ScholarID	GradePtAvg	row	*StudentNo	Track	GPA
	1	C1113	3.9	1	B1414	A	3.4
	2	C1163	3.5	2	B1830	B	2.3
	3			3	B1716	C	3.7
	4	C1344	3.3				
	5	C1711	3.9				
	6	C1579	2.8				

Fig. 12: Result of inserting a row in one subtable. Note that a row inserted in the CSE subtable without affecting the Biology substable.

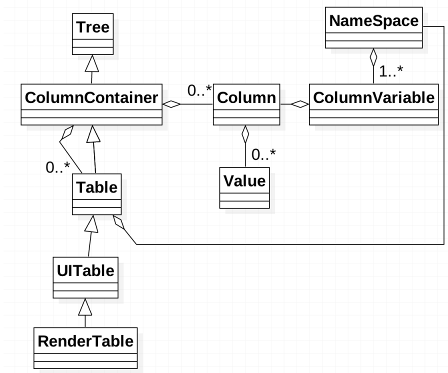

Fig. 13: *SciSheets core classes.*

- Use term "formula evaluation loop"
- Calculation workflow

Concern (2), localizing errors, seques into a broader discussion of how spreadsheets are executed. This must be done in a way so that the column formulas run in a standalone program.

Last, we consider performance. Our experience is that there are two common causes of poor performance in our current implementation of SciSheets. The first relates to data size since since, at present, SciSheets embeds data with the HTML document that is rendered by the browser. We expect to address this by implementing a feaure whereby data are downloaded on demand and cached locally.

The second cause of poor performance is having many iterations of the formula evaluation loop. If there is more than one formula column, then the best case is to evaluate each formula column twice. The first execution produces the desired result (which is possible if the formula columns are in order of their data dependencies); the second execution confirms that the result has converged. Some efficiencies can be gained by using the Prologue and Epilogue features for one-shot execution of high overhead operations (e.g., file I/O). Also, we are exploring the extent to which SciSheets can detect automatically when static dependency checking is possible so that formula evaluation is done only once.

Clearly, performance can be improved by reducing the number of formula columns since this reduces the maximum number of iterations of the formulation evaluation loop. SciSheets supports this strategy by permitting formulas to be scripts. This is a reasonable strategy for a Scripter, but it may work poorly for a Calcer who is unaware of data dependencies.

5. Future Directions

5.1 Subtables with Scoping

1. Approach to reuse

5.2 Plotting

- **Plotting** requirement.

5.3 Multiple Languages

5.4 Github Integration

- **Reproducibility** requirement.

Requirement	SciSheets Feature
• Expressivity	• Python formulas • Formula scripts
• Reuse	• Program export • *Subtables with Scoping*
• Complex Data	• Subtables
• Performance	• Progam export • Prologue, Epilogue • *Load data on demand* • *Conditional static dependency checking*
• Plotting	• *Embed bokeh components*
• Script Debuggablity	• Localized exceptions
• Reproducibility	• `github` *integration*

TABLE 1: *Summary of requirements and SciSheets features that address these requirements. Features in italics are planned but not yet implemented.*

- Why version control
- Structure of the serialization file
- User interface for version control

6. Conclusions

We developed SciSheets to address deficiencies in existing spreadsheet systems.

1. Discuss entries in table. For now, performance is not evaluated.

2. SciSheets seeks to improve the programming skills of its users. It is hoped that Calcers will start using scripts, and that Scripters will gain better insight into modularization and testing.

References

[BURN2009] Burnett, M. *What is end-user software engineering and why does it matter?*, Lecture Notes in Computer Science, 2009

[MODE2017] *MODELOFF - Financial Modeling World Championships*, http://www.modeloff.com/the-legend/.

[Thib2013] Thibodeau, Patrick. *India to overtake U.S. on number of developers by 2017*, COMPUTERWORLD, Jul 10, 2013.

[MCCU2006] McCutchen, M., Itzhaky, S., and Jackson, D.*Object spreadsheets: a new computational model for end-user development of data-centric web applications*, Proceedings of the 2016 ACM International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software, 2006.

[JONE2003] Jones, S., Blackwell, A., and Burnett, M. i *A user-centred approach to functions in excel*, SIGPLAN Notices, 2003.

[Gonz2010] *Google Fusion Tables: Web-Centered Data Management and Collaboration*, Hector Gonzalez et al., SIGMOD, 2010.

[PySpread] Manns, M. *PYSPREAD*, http://github.com/manns/pyspread.

[Stencila] *Stencila*, https://stenci.la/.

[Pere2015] Perez, Fernando and Branger, Brian. *Project Jupyter: Computational Narratives as the Engine of Collaborative Data Science*, http://archive.ipython.org/JupyterGrantNarrative-2015.pdf.

[SciSheet] *SciSheets*, https://github.com/ScienceStacks/SciSheets.