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 avoids the mental burdens of programming, especially considerations of control flow, data dependencies, and data structures. Recent 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].

Our experience is that there are three types of spreadsheet users.

- **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-loops and if-then statements, and they can use simple data structures such as lists and dataframes (which are like spreadsheets). However, they rarely encapsulate code into functions, preferring to copy code to get reuse.
- **Programmers** know about advanced data structures, modularization, and testing.

We find that the bulk of spreadsheet users who employ formulas are Calcers and then Scripters. Programmers are more likely to use a mix of formulas and macros (e.g., Visual Basic for Microsoft Excel or AppScript in Google Sheets).

Despite their appeal, the use of spreadsheet formulas has severe shortcomings.

‡ *University of Washington*
∗ *Corresponding author: joseph.hellerstein@gmail.com*

- poor scalability because executing formulas within the spreadsheet system has high overhead;
- great difficulty with reuse because there is no concept of encapsulation (and even different length data are problematic);
- great difficulty with transitioning from a spreadsheet to a program to facilitate integration into software systems;
- limited ability to handle complex data because there is no concept of structured data;
- poor readability because formulas must be expressions (not scripts) and any cell may have a formula; and
- limited ability to express calculations because formulas are restricted to using a few hundred or so functions provided by the spreadsheet system (or specially coded macros).

Academic computer science has recognized the growing importance of end-user programming (EUP) [BURN2009]. Even though spreadsheets are likely the most pervasiveness example of EUP, there is a virtual absence of academic literature about addressing the shortcomings of spreadsheets. Outside of academia there has been significant interest in innovating spreadsheets. Google Fusion Tables [Gonz2010] uses 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 increases the expressivity of formulas. A more radical approach is taken by Stencila [Stencila]. Stencila is not really a spreadsheet system; it is a document system that provides ways to execute code that can update tables (more along the lines of a Jupyter notebook [Pere2015]). Stencila supports a variety of languages including JavaScript, Python, and SQL. However, Stencila strays from features that spreadsheet users expect: (a) easily associating code with data values and (b) avoiding considerations of data dependencies in calculations.

Even with these innovations, serious deficiencies remain in spreadsheets. Specifically, the following requirements are poorly addresed.

- **Expressivity**: The expressivity of formulas is limited because formulas are restricted to being expressions, not scripts.
- **Reuse**: It is impossible to reuse spreadsheet formulas in other formulas or in software systems.

THDPA	V	INV_THDPA	INV_V	INTERCEPT	SLOPE	Vmax	KM
0.010	0.110	100.000	9.091	4.357	0.050	0.229	0.011
0.050	0.190	20.000	5.263				
0.120	0.210	8.333	4.762				
0.200	0.220	5.000	4.545				
0.500	0.210	2.000	4.762				
1.000	0.240	1.000	4.167				

THDPA	V	INV_THDPA	INV_V	INTERCEPT	SLOPE	Vmax	KM
0.010	0.110	=1/A2	=1/B2	=INTERCEPT(D2:D7,C2:C7)	=SLOPE(D2:D7,C2:C7)	=1/E2	=F2*G2
0.050	0.190	=1/A3	=1/B3				
0.120	0.210	=1/A4	=1/B4				
0.200	0.220	=1/A5	=1/B5				
0.500	0.210	=1/A6	=1/B6				
1.000	0.240	=1/A7	=1/B7				

Fig. 1: *Data view (top) and formulas view (bottom) for an Excel spreadsheet that calculates Michaelis-Menten Parameters.*

	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. 2: *Student grade data from two departments in the school of engineering.*

- **Complex Data**: It remains burdensome to deal with complex data relationships in spreadsheets, such as hierarchically structured data.
- **Performance**: Very little has been done to address the performance problems that occur as spreadsheets scale.

This paper introduces SciSheets [SciSheets], a new spreadsheet system with the goal of delivering the power of programming with the simplicity of spreadsheets. Our target users are technical professionals, such as scientists and financial engineers, who do complex calculations on structured data. To date, our focus has been on calculations, not features such as formatting.

SciSheets addresses the above requirements by introducing several novel features.

- Expressivity: *Formulas can be Python scripts, not just expressions.*
- Reuse and Performance: *Spreadsheets can be exported as standalone Python programs.* This provides for sharing and reuse since the exported codes can be used by other SciSheets spreadsheets or by python programs. This feature also improves scalability since calculations can be executed without the overhead of the spreadsheet system.
- Complex Data: *Tables can have nested columns (columns within columns).* This provides a conceptually simple way to express complex data relationships, such as n-to-m relationships.

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

2. Requirements

This section motivates the requirements of expressivity, reuse, and complex data through examples.

1. Expressivity and Reuse
   a. Background. Common processing of biochemical assays to compute key characteristics of enzymes
   b. Requirements
   a.) *Expressivity*: limited ability specify calculations as expressions b.) *Reuse*: Cannot reuse (robustly) formulas in other spreadsheets or in software systems
2. Complex Data
   a. Background. Multiple departments in the school of engineering, keeping records in slightly different ways.
   b. Requirements
      a) *Complex data*: Cannot easily manipulate complex data, such as nested tables. Examples include of manipulations: View data side-by-side, but still manage as separate tables in terms of insert/delete.

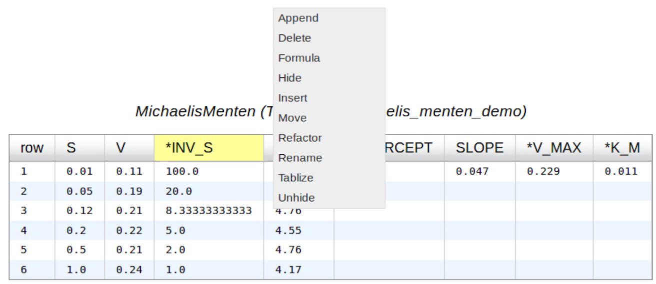


Fig. 3: *Column popup menu in a scisheet for the Michaelis-Menten calculation.*

3. Features

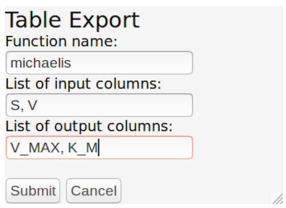
1. SciSheets UI structure
   a. Elements - sheet, tables, columns, rows, cells (Fig)
   b. Popup menus
   c. Execution model: prologue, formula evaluations, epilogue. (Dependency checking is not possible because users can employ "eval" statement.)

Fig. 4: *Formula for computing the inverse of the input value S.*

INTERCEPT

```
import scipy.stats as ss
SLOPE, INTERCEPT, _, _, _ = ss.linregress(INV_S, INV_V)
SLOPE = np.round(SLOPE, 3)
INTERCEPT = np.round(INTERCEPT, 3)
```

Fig. 5: Formula for computing the slope and intercept of a regression line for the Michaelis-Menten calculation. Note that One column assigns values to another column and that a script is used. label:fig-simpleformula



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. 6: Menu to export a table as a standalone python program.

2. Expressivity: Formulas can be scripts
3. Reuse: Code re-use through export
4. Complex data: managing multiple tables

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

Fig 12 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 latter means 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 ...

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.

	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. 7: A table with two subtables.

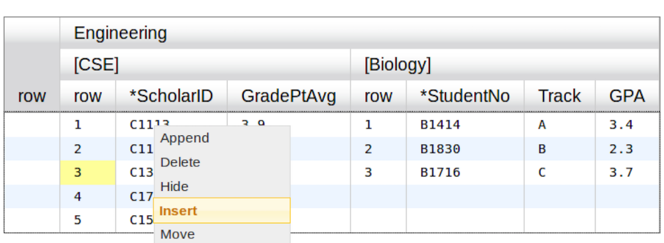


Fig. 8: Menu to insert a row in one subtable.

	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. 9: Result of inserting a row in one subtable.

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. 10: 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. 11: Column formula that is a script to process CSV files.

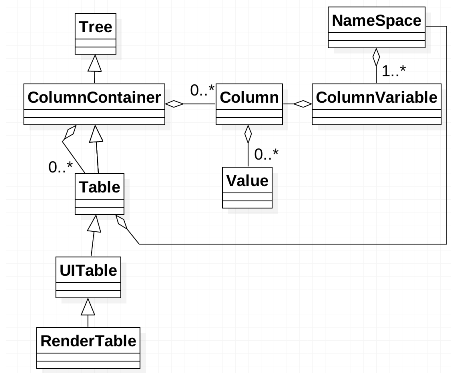


***Fig. 12:** SciSheets core classes.*

```
# 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)

    try:
      # Column INV_V
      s.controller.startBlock('INV_V')
      INV_V = np.round(1/V,2)
      s.controller.endBlock()
      INV_V = s.coerceValues('INV_V', INV_V)
    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()
```


5. Futures

- Hierarchical tables with local scopes provides another approach to reuse.
- Graphics
- Github integration
  - Why version control
  - Structure of the serialization file
  - User interface for version control

6. Conclusions

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.

[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.

Requirement	SciSheets Feature
• Expressivity	• python formulas • formula scripts
• Reuse	• program export • *hierarchical tables with local scopes*
• Complex Data	• hierarchical tables
• Performance	• progam export
• 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.*

[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.