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 one or more equations. Spreadsheet formulas work well for Calcers since: (a) they do not have to structure calculations based on data dependencies; (b) they can use "copy" and "paste" for iteration; and (c) the only data structures are rectangular blocks (more commonly a single cell), which is easily visualized.
- **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).

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

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

- 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 and improve scalability;
- 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 limited 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], and spreadsheets are likely the most pervasiveness example of EUP. However, even with the seriousness of shortcomings of spreadsheets, there is a virtual absence of academic literature about addressing these shortcomings. On the other hand, there has been significant commercial interest. 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 expressiveness of formulas. A more radical approach is taken by the Stencila system [Stencila], which provides a document structure that includes cells that execute formulas, including the display of of data tables; cells may execute statements from many languages including Python and R.

Sadly, even with the aforementioned innovations in spreadsheets, serious deficiencies remain.

1. The expressiveness of formulas is limited because formulas are restricted to being expressions, not scripts (although Stencila does provide a limited form of scripting).
2. None of the innovations ease the burden of dealing with complex data relationships, such as n-to-m relationships.

3. None of the innovations address code sharing and reuse between spreadsheet users or between spreadsheet users and software engineers.
4. 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 deficiencies enumerated above by introducing several novel features.

- *Formulas can be Python scripts, not just expressions.* This increases the expressiveness of formulas.
- *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.
- *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.

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

The remainder of this paper is organized as follows.

2. Use Cases

1. User profiles
   a. Calcers - no knowledge of data types or control flow. Doesn't think about data dependencies. Mental model is a calculator.
   b. Scripter - Writes scripts, saving them in a file. Can do if-then, for-loop, and list data types and pandas DataFrames.
   c. Programmer - Knows about functions and modules.

Our hope is to elevate the capabilities of the first two groups, introducing calculators to the power of scripting and scripting to the power of programming.

2. Michaelis-Menten
   a. Background. Common processing of biochemical assays to compute key characteristics of enzymes
   b. Use cases
      a) Writing formulas - script vs. expression
      b) Code reuse - None

3. Managing multiple tables

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.*

   a. Background. Multiple departments in the school of engineering, keeping records in slightly different ways.
   b. Use cases
      a) View data side-by-side, but still manage as separate tables in terms of insert/delete

3. Addressing the Use Cases

1. UI structure
   a. Elements - sheet, tables, columns, rows, cells (Fig)
   b. Popup menus

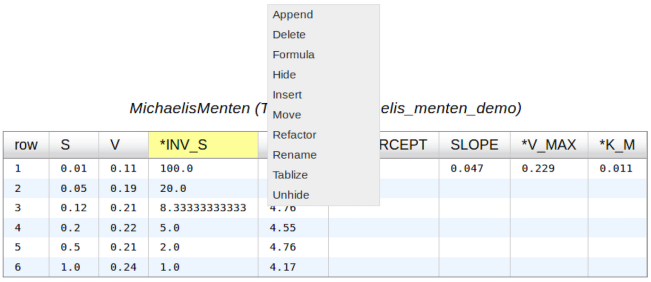


Fig. 3: *Column popup menu in a scisheet for the Michaelis-Menten calculation.*

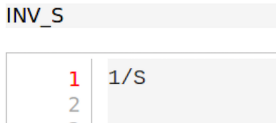
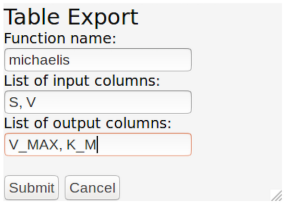


Fig. 4: *Formula for computing the inverse of the input value S.*

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.*

c. Execution model: prologue, formula evaluations, epilogue. (Dependency checking is not possible because users can employ "eval" statement.)

2. Code re-use through export

3. Formulas can be scripts

4. Managing multiple tables

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.*

	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. 9: *A table with two subtables.*

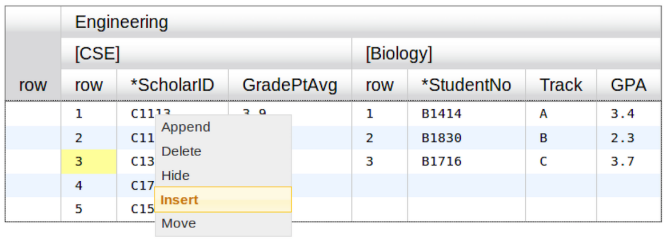


Fig. 10: *Menu to insert a row in one subtable.*

4. Design

1. Client-Server architecture
   a. Client (JS) - Simple UI handling (pop-ups, render table, convey user inputs via AJAX)
   b. Server (python) - table storage, formula evaluation
2. Software Dependencies - Django, JS packages
3. Class hierarchy
4. SciSheet export
5. Implications of requirements
   a. Requirements
   a.) User doesn't think about data dependencies between columns. b.) User can write arbitrary python scripts.
   b. Implications
   a.) Cannot do static dependency determination. Solution - execute until convergence. b.) Syntax and runtime errors must be isolated within the line in the column, not just to the column.

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

	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. 11: *Result of inserting a row in one subtable.*

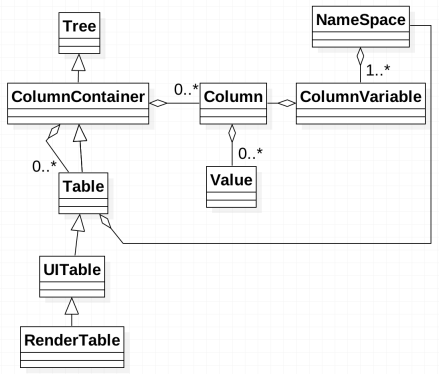


Fig. 12: *SciSheets core classes.*

```
  s.initialize()
  _table = s.getTable()
```

Prologue

```
# Prologue code
s.controller.startBlock('Prologue')
# 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
s.controller.endBlock()


# Formula evaluation loop
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. Logging and performance

5. Future Work

- Realizing the full power of hierarchies - reuse with "copy" action but with different technical semantics.
- Graphics
- Version control

6. Conclusions

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.

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

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

Problems	Solutions
• expressiveness	• python formulas • formulas can be scripts
• reuse	• export as a program • *copy with local scope*
• scalability	• export as a program
• reproducible	• *embedded version control*
• debuggable	• localized exception handling

TABLE 1: *Summary of the problems in current spreadsheets and how SciSheets features address these problems. Items in italics are not yet implemented.*