Spreadsheets in Education (eJSiE)

Volume 9 | Issue 1 Article 5

May 2016

Unlocking the Spreadsheet Utility for Calculus: A Pure Worksheet Solver for Differential Equations

Chahid K. Ghaddar
ExcelWorks LLC, cghaddar@excel-works.com

Unlocking the Spreadsheet Utility for Calculus: A Pure Worksheet Solver for Differential Equations

Abstract
This paper presents a unique solver for nonlinear initial-boundary value partial differential equations (PDE) that integrates with Microsoft Excel as a pure math function. The solver receives via input arguments formulas, variables, and parameters for the PDE, and is executed as a regular formula command in a range of cells. The solver, utilizing the method of lines, evaluates to a formatted tabular solution suitable for direct plotting of snapshot or transient views. Design of the solver is made possible by bypassing restrictions that block a worksheet function from receiving and evaluating formulas while preserving its purity. Three examples are presented to demonstrate the merits of this unconventional solver design which shields the tedious algorithmic implementation details from the user, and greatly simplifies solving a PDE using an intuitive math function without any dialogues, macros or VBA programming.

Keywords
partial differential equations, solvers, spreadsheet, calculus, numerical methods

I. INTRODUCTION

Partial differential equations (PDE) models are inescapable in science and engineering [1] as well as modern social sciences [2]. When PDE involve more than one spatial independent variable, the problem of geometry definition becomes an integral part of solving them. An entire industry has evolved supporting integrated advanced geometrical modeling and computational tools for solving PDE in product design. These sophisticated and expensive tools are typically geared to specific engineering applications. For simple regular geometries, several mathematical tools such as MATLAB® and Mathematica® offer powerful solvers for PDE, and are often used in the classroom. Students are expected to master the programming languages of these tools in order to utilize them effectively.

On the other hand, the ubiquitous spreadsheet application, primarily Excel, is widely used by professionals, and in the classroom as a math tool, thanks in part to its ease of use, rich intrinsic math functions, graphing, and extensibility. Attempts to solve PDE in a spreadsheet have been limited in scope to pedagogical linear models. These methods typically utilize finite difference approximations which map naturally to the native grid structure of the spreadsheet. Visual Basic for Application (VBA) programs are developed in conjunction with macros and user input forms to carry out the solution procedure [3], [4]. Such methods serve primarily as teaching aid on structured numerical solution procedures for linear PDE but are otherwise impractical for solving general nonlinear PDE. They do however, illustrate how far the common spreadsheet has been exploited perhaps beyond what its original designers have anticipated.

In this article we introduce a practical spreadsheet solver for a general system of nonlinear transient boundary value differential equations in one spatial dimension which can be presented in the following form:

$$\frac{\partial u_i}{\partial t} = f_i(t, x, \boldsymbol{u}, \boldsymbol{u_x}, \boldsymbol{u_{xx}}), \;\; i = 1, .., n \tag{1}$$

subject to proper initial and boundary conditions. The restriction to one spatial dimension avoids the problem of geometry definition which is, admittedly, not suited for the spreadsheet application. In (1) we have also restricted the order of the equations to 2nd order. This restriction can be easily relaxed to include higher orders although it is generally sufficient for modeling a practical range of problems.

The solver is designed exclusively for Excel as a pure worksheet math function. Definition of the problem formulas, as given in (1), are passed as arguments, and the solver is evaluated as a standard formula in an allocated range. The solver populates the range with a formatted solution which can be readily plotted. The solver pure functional design which receives the PDE formulas, variables and parameters via regular arguments, requires no dialogues, and does not modify its inputs or any data

in the spreadsheet, is strikingly different from the conventional spreadsheet command utilized for evaluating formulas, and the several hundred intrinsic math functions which can only operate on constant numeric input. The design is made possible by means of an innovative method [5] which allows a math function to accept formulas as a new type of argument while preserving its purity. The method works by capturing the definition of an input argument formula using the spreadsheet Advanced Programming Interface (API), and constructing a relational graph of nodes representing the formula inter-dependence on nested formulas, variable cells, and recursive calls. A graph evaluator which exploits the spreadsheet API, is employed to evaluate the relational nodes of the graph in an order of their dependence, and aggregating their values to obtain the value of the formula all without modifying any data in the spreadsheet.

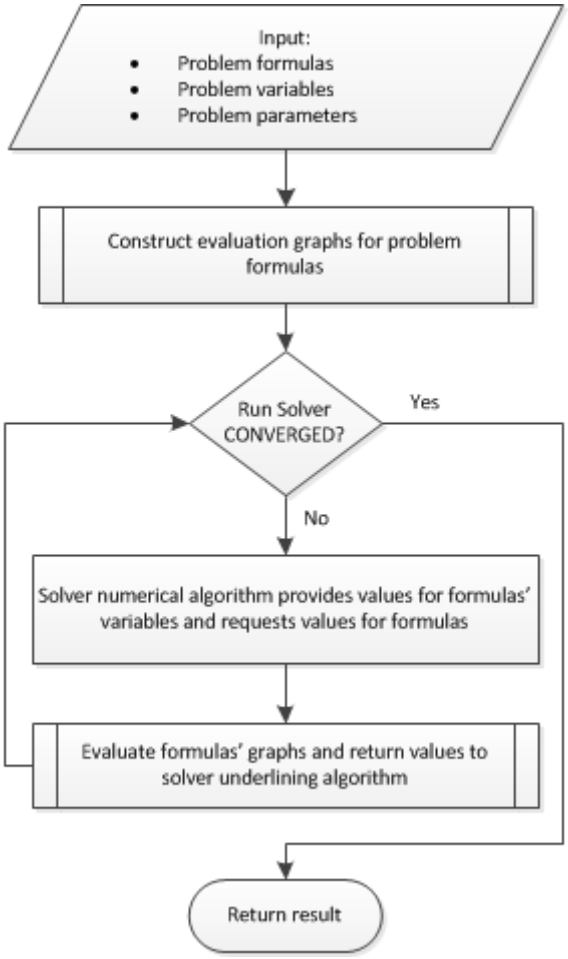

Figure 1. Flowchart for worksheet solver function design

The method enabled the development of a new class of worksheet calculus functions [6] based on the flowchart logic of Fig 1. The benefits of the pure functional solver design over conventional interactive command approaches are noteworthy. First the solver fully separates the numerical procedure, which is often of less interest to the user, from the problem input model and output solution. In contrast, the command mechanism which is the standard procedure for updating variable cells values and evaluating dependent formulas, utilizes the spreadsheet explicitly as the computational grid for the numerical algorithm. In essence, it mixes up inputs, algorithmic procedure, and results overwriting inputs by results. Second, and more importantly, by preserving purity and recursion properties, the functions can achievably support a functional paradigm for solving more complex problems including dynamical optimization and optimal control as demonstrated by the author in [8]. Since commands are not mathematical functions, and do not possess such essential properties to support a functional paradigm, solving dynamical optimization problems of differential systems have remained outside the scope of traditional spreadsheet applications.

The remainder of this article is divided into three sections. In the next section, we describe the PDE problem formulation and the employed numerical algorithm. Next, the worksheet solver interface and output format are described. This is followed by demonstrating the solver with three PDE examples. We recommend reviewing Appendix A1, which includes a brief description of basic spreadsheet concepts, for any reader not familiar with the spreadsheet prior to reviewing the examples.

II. PROBLEM STATEMENT AND NUMERICAL ALGORITHM

We consider a system of initial boundary-value nonlinear partial differential equations of at most second order which is given in the form (1) in the domain $0 \leq x \leq L$, and for $t \in [0, T]$. Initial condition $u_i(x, 0) = ic_i(x)$, is required for each state variable $u_i, i = 1, .. n$. To facilitate defining the initial condition function as a formula in the spreadsheet, we represent it with respect to zero:

$$0 = ic_i(x), \quad i = 1, .. n \tag{2}$$

Depending on the highest derivative order for each equation in system (1), one or two boundary conditions are required for each equation. The boundary conditions are specified at the left (x=0) and/or right (x=L) points of the domain. Each boundary condition may depend on $t, \boldsymbol{u}$, and $\boldsymbol{u_x}$. Again to enable definition by a formula, we represent each boundary condition function with respect to zero:

$$0 = bc_i(t, \boldsymbol{u}, \boldsymbol{u_x}) \tag{3}$$

The general form (1) can describe a variety of PDE types in terms of geometric properties, nonlinearity and smoothness for which a particular solution method may be more effective than others. The design of the pure solver permits encapsulating multiple finite difference and Galerkin methods which can be selected by the user via options. The first order initial value form (1) makes the method of lines [9], [10] particularly attractive given the availability of several ODE implicit adaptive integration schemes codes, and hence has been selected for the default algorithm. Spatial discretization is carried out on a uniform mesh by standard collocation over piecewise polynomials using B-spline basis functions [10]. The resulting implicit ODE system is integrated by any of the implicit schemes RADAU5, BDF, or ADAMS with adaptive step control [11], [12].

III. SPREADSHEET SOLVER DESCRIPTION

A. Representation of PDE in spreadsheet

The system (1) can be described by *n* formulas for the right-hand-side (RHS) equations f_i, which are defined in terms of a selected set of *(2+3n)* cells corresponding to the system variables $(t, x, \boldsymbol{u}, \boldsymbol{u_x}, \boldsymbol{u_{xx}})$. The system relationship can be preserved by maintaining the formulas and variables are passed to the solver in a pre-determined order as follows: $(f_1, f_2, .., f_n)$ for formulas, and $(t, x, u_1, .., u_n, u_{1,x}, .., u_{n,x}, u_{1,xx}, .., u_{n,xx})$ for variables. The selection of the cells which define the formulas and variables is arbitrary. In practice, it is convenient to work with a contiguous set of named cells rather than raw cell addresses. Each variable $u_1, .., u_n$ cell must contain a formula which defines the initial condition, $ic_i(x)$ for the variable. The initial condition formula is defined in terms of the cell variable representing x.

To define the boundary conditions (3), we use two additional cells for each equation to hold the left and right boundary conditions formulas which are defined using the cells representing $(t, u_1, .., u_n, u_{1,x}, .., u_{n,x})$. If no boundary condition is required, the text 'NA' is added to the corresponding cell. This completes the representation of the PDE system in Excel in terms of standard formulas. Additional parameters including time and spatial domains are passed as standard value arguments.

B. Solver Interface

Based on the preceding representation of the PDE system in Excel, the design for the solver interface follows naturally as shown in the formula:

$$=PDSOLVE\ (rhs, vars, lbc, rbc, L, T, [options])) \quad (4)$$

References to the system RHS formulas are supplied to PDSOLVE in the first parameter *rhs*, and the system variables in *vars*. References to the left and right boundary conditions formulas are specified via *lbc* and *rbc* respectively. Spatial and temporal domains limits are defined in the 5th and 6th parameters *L* and *T*. The solver accepts additional optional arguments which are described in detail in [7]. These arguments include supplying analytic Jacobians for the system formulas, as well as custom settings for the underlining algorithm such as tolerances and selection of temporal integration algorithm.

Once the arguments are specified, the user simply evaluates the solver as an array formula in a pre-allocated range in Excel large enough to hold the solution. The solver computes and displays a formatted solution as described in the next section.

C. Solution layout in the spreadsheet

We present the numerical solution for system (1) in Excel in one of two tabular layouts that are convenient for plotting a transient or snapshot views of the solution. Fig 2 shows a snapshot layout for a system of two equations. This layout allows for direct plotting of the system variables spatial profiles at selected time points in the first row. In the transient layout, the order of x and t are exchanged to allow for direct plotting of the variables transient behavior at selected spatial points. Values of the independent variables x and t are reported at uniform intervals as determined by the size of the allocated array for output. However, this default behavior can be changed by either providing a step size or specific custom points using optional formats for the solver parameters *L* and *T*. Furthermore, displaying the columns for the first and second derivatives can be turned on or off via solver options [7].

	A	B	C	D	E	F	G	H	I	J	K
1	t	t_0	t_0	t_0	t_0	t_0	t_0	t_1	t_1	t_1 ➡	
2	X	u_1	u_1	$u_{1,x}$	$u_{1,x}$	$u_{1,xx}$	$u_{2,xx}$	u_2	u_2	$u_{2,x}$	..
3	x_0										..
4	x_1										..
5	x_2										..
..	⬇										
N	L										

Each column block has solution values for dependent variable at (x, t) values

Figure 2. Snapshot solution layout in Excel for partial differential equation solver PDSOLVE. The display of 1st and 2nd derivative variables is optional. In the transient view layout, the order of 'x' and 't' is exchanged

IV. EXAMPLES

A. Parabolic Heat Equation

We demonstrate the use of PDSOLVE by computing the solution to a transient heat transfer problem across a slab that is initially at zero temperature with an insulated right side. At time equals zero, the left side is brought to 100 degrees. The problem is described by the parabolic equation:

$$\frac{\partial u}{\partial t} = k\frac{\partial^2 u}{\partial x^2} \tag{5}$$

with initial condition $u(x,0) = 0$, left boundary condition $u(0,t) = 100$, and right boundary condition $\partial u(x,t)/\partial x = 0$ at $x = 1$. The problem's spatial domain is $x \in [0,1]$ and the time interval is $t \in [0,1]$. Fig 3 shows the complete system model in Excel. Using cells T1, X1, U1, U2, and U3 for system variables t, x, u, u_x, u_{xx} respectively, the system RHS formula is defined in A1, the left boundary condition in B1, and the right boundary condition in C1, the conductivity k in K1 and the initial condition in U1.

	A	**B**	**C**	**K**	**U**
1	=K1*U1	=U1-100	=U2	1	=IF(X1=0,100,0)

Figure 3. Problem setup for Example 1 in Excel

To compute the solution, we simply evaluate the following PDSOLVE formula:

=PDSOLVE(A1, (T1,X1,U1:U3), B1, C1, {0,1}, {0,.5,1}) (6)

in an allocated range E1:H23. In (6), we pass references to the system RHS equation, variables, and boundary condition formulas defined in Fig 3. The 6th argument {0, 0.5, 1} instructs the solver to report output time points at 0, 0.5 and 1 only, whereas output spatial points in the domain {0, 1} are reported uniformly in accordance with the available number of rows in the allocated solution range. PDSOLVE populates the range with the snapshot solution format shown in Fig 4 and plotted in Fig 5.

Alternatively; we can specify via the solver's optional parameters a transient format for the output [7], which simplifies plotting transient response at selected spatial points as is shown in Fig 6.

	E	F	G	H
1	T1	0	0.5	1
2	X1	U1	U1	U1
3	0	100	100	100
4	0.05	100	97.0906084	99.15263854
5	0.1	100	94.19916828	98.31050871
6	0.15	100	91.34351958	97.47880964
7	0.2	100	88.54127962	96.66267545
8	0.25	100	85.80973399	95.86714337
9	0.3	100	83.16572917	95.09712222
10	0.35	100	80.62556836	94.35736176
11	0.4	100	78.20491103	93.65242317
12	0.45	100	75.91867607	92.9866506
13	0.5	100	73.78095094	92.36414463
14	0.55	100	71.8049046	91.78873701
15	0.6	100	70.00270828	91.26396785
16	0.65	100	68.3854604	90.79306425
17	0.7	100	66.96312022	90.37892153
18	0.75	100	65.7444465	90.02408583
19	0.8	100	64.73694473	89.73073923
20	0.85	100	63.94682086	89.50068639
21	0.9	100	63.37894262	89.33534325
22	0.95	100	63.03680972	89.23572814
23	1	100	62.92253049	89.20245472

Figure 4. Computed solution for Example 1 by PDSOLVE

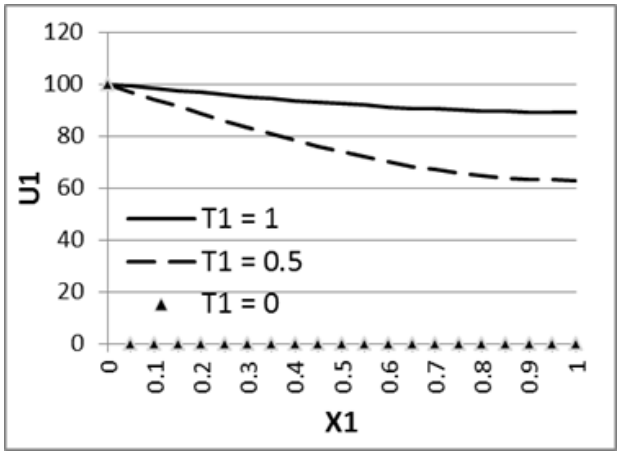

Figure 5. Plot of Fig 4 results showing temperature distribution at various times

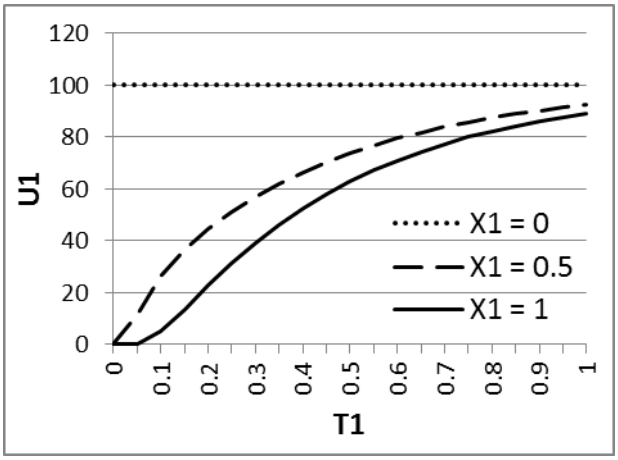

Figure 6. Transient temperature profiles at various positions

B. Hyperbolic Wave Equation

Next we solve the following 2nd order inhomogeneous wave equation:

$$\frac{\partial^2 u}{\partial t^2} = \alpha \frac{\partial^2 u}{\partial x^2} + 2x^2 + \sin(t) \tag{7}$$

for the initial and boundary conditions shown in Table 1.

Table 1. Problem definition for Example 2

Time period	$\boldsymbol{t} \in [\mathbf{0}, \mathbf{1}]$
Spatial range	$0 \leq x \leq 4$
Parameter	$\propto = 1$
Initial conditions	$u(x,0) = \sin(\pi x)$ $\frac{\partial u(x,t)}{\partial t}\|_{t=0} = -\pi\, cos(\pi x)$
Left boundary condition	$u(x,t)\|_{x=0} = 0$
Right boundary condition	$u(x,t)\|_{x=4} = 0$

Using standard substitution, we to convert (7) to two 1st order equations in time:

$$\frac{\partial u}{\partial t} = v$$
$$\frac{\partial v}{\partial t} = \alpha \frac{\partial^2 u}{\partial x^2} + 2x^2 + \sin(t) \quad (8)$$

With this substitution, the 2nd initial condition is now assigned to $v(x, 0)$. Note that the left and right boundary conditions which constrain u must be defined in relation to the first equation governing $\frac{\partial u}{\partial t}$ and no boundary conditions are required for the 2nd equation governing $\frac{\partial v}{\partial t}$. To define (8) in Excel, we use the cells T1, X1, U1, V1, U2, V2, U3, V3 to represent the system variables $t, x, u, v, u_x, v_x, u_{xx}, v_{xx}$ respectively, and define the equations and boundary conditions as shown in Fig 7.

	RHS	Left BC	Right BC
	A	**B**	**C**
1	=V1	=U1	=U1
2	=A1*U3+2*X1^2+SIN(T1)	NA	NA

Figure 7. Problem setup for Example 2 in Excel

We also assign the initial condition formulas for variables U1 and V1 as shown in Fig 8.

	U	**V**
1	=SIN(PI()*X1)	=-PI()*COS(PI()*X1)

Figure 8. Definition of initial conditions for Example 2

To compute the solution, we evaluate the following PDSOLVE formula:

=PDSOLVE(A1:A2, (T1,X1,U1:V2), B1:B2, C1:C2, {0,4}, {0,.5,1}) (9)

in allocated range H1:N23, and obtain the results shown in Fig 9.

	H	I	J	K	L	M	N
1	T1	0	0	0.5	0.5	1	1
2	X1	U1	V1	U1	V1	U1	V1
3	0	0.0000	-3.1416	0.0000	0.3559	0.0000	0.2296
4	0.2	0.5878	-2.5416	0.0374	0.6525	0.2184	0.4696
5	0.4	0.9511	-0.9708	0.0663	1.0155	0.4283	1.1009
6	0.6	0.9511	0.9708	0.4391	-2.3810	0.6660	1.7395
7	0.8	0.5878	2.5416	0.9999	-1.0115	0.9596	2.1886
8	1	0.0000	3.1416	1.2810	1.2056	1.3664	1.9869
9	1.2	-0.5878	2.5416	1.2000	3.4923	2.3485	1.3592
10	1.4	-0.9511	0.9708	0.8300	5.1536	3.2362	4.0835
11	1.6	-0.9511	-0.9708	0.3620	5.7536	3.8363	7.2172
12	1.8	-0.5878	-2.5416	0.0320	5.2924	4.1530	10.1480
13	2	0.0000	-3.1416	0.0310	4.2058	4.3252	12.2679
14	2.2	0.5878	-2.5416	0.4320	3.1992	4.5774	13.3480
15	2.4	0.9511	-0.9708	1.1620	2.9780	5.1342	13.6172
16	2.6	0.9511	0.9708	2.0300	3.9779	6.1341	13.6839
17	2.8	0.5878	2.5416	2.8000	6.1992	7.5730	14.2051
18	3	0.0000	3.1416	3.2810	9.2056	9.3689	17.5690
19	3.2	-0.5878	2.5416	3.3999	12.2825	11.0936	15.0290
20	3.4	-0.9511	0.9708	3.2380	14.7019	11.2085	11.3838
21	3.6	-0.9511	-0.9708	3.1089	16.6364	9.3706	7.5006
22	3.8	-0.5878	-2.5416	2.2000	9.1226	5.5533	3.6277
23	4	0.0000	-3.1416	0.0000	0.2805	0.0000	0.1835

Figure 9. Computed solution for Example 2 by PDSOLVE

Fig 10 shows a snapshot plots of $u(x,t)$ at $t = 0, 0.5$, and 1.0. The plot is easily constructed in Excel by highlighting the data of Fig 9 and inserting a scatter plot.

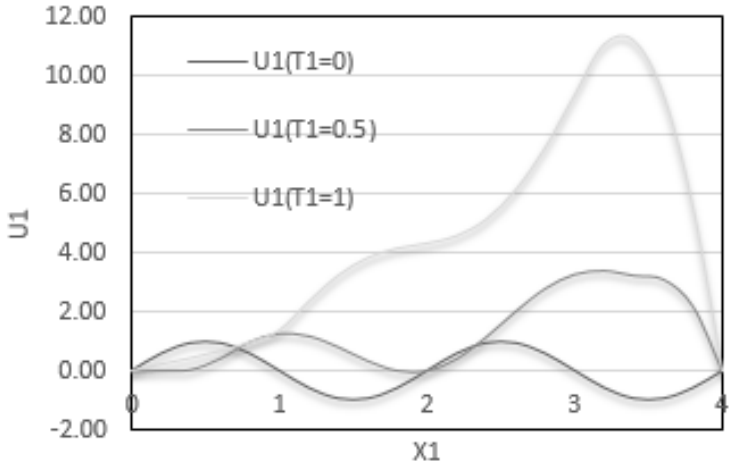

Figure 10. Plot of Fig 9 solution showing wave propagation at various times

C. Burgers Equation

In the 3rd example, we solve the 1D dissipative Burgers equation and compare results to the exact Fourier solution. Burgers equation arises in various topics including fluid mechanics in particular. We consider the following Burgers equation:

$$\frac{\partial u}{\partial t} = -u\frac{\partial u}{\partial x} + v\frac{\partial^2 u}{\partial x^2} \tag{10}$$

with initial condition

$$u(x,0) = \sin(\pi x)\,, \quad 0 < x < 1, \tag{11}$$

and homogenous boundary conditions:

$$u(0,t) = \mathrm{u}(1,\mathrm{t}), \quad t > 0 \tag{12}$$

The exact Fourier solution for (10)-(12) is given by [13]:

$$u(x,t) = 2\pi v \frac{\sum_{n=1}^{\infty} a_n \exp(-n^2\pi^2 vt)\, n \sin(n\pi x)}{a_0 + \sum_{n=1}^{\infty} a_n \exp(-n^2\pi^2 vt)\cos(n\pi x)}, \tag{13}$$

where

$$a_0 = \int_0^1 \exp(-(2\pi v)^{-1}[1-\cos(\pi x])dx, \tag{14}$$

and,

$$a_n = 2\int_0^1 \exp(-(2\pi v)^{-1}[1-\cos(\pi x])\cos(n\pi x)\,dx \tag{15}$$

Table 2 displays vales of the exact solution computed using (13) as reported in [13] for viscosity values 0.1 and 0.01, at the shown values for x and t.

Table 2. Exact solution values for Example 3 as reported in [13]

x	*t*	*u(x,t), v=0.1*	*u(x,t), v=0.01*
0.25	0.4	0.30889	0.34191
	0.6	0.24074	0.26896
	1	0.16256	0.18819
	3	0.02720	0.07511
0.75	0.4	0.62544	0.91026
	0.6	0.48721	0.76724
	1	0.28747	0.55605
	3	0.02977	0.22481

To model (10) in Excel, we refer to the system variables t, x, u, u_x, u_{xx} by the named cells t, x, u, ux, uxx assigned to T1, X1, U1, U2, U3 respectively. The system RHS formula is defined in A1, the left and right boundary condition formulas in B1 and C1, and the initial condition in U1 as shown in Fig 11. We make use of D1 (named v) to hold the viscosity value set initially to 0.1.

	A	B	C	D	U
1	=-u*ux+v*uxx	=u	=u	0.1	=SIN(PI()*x)

Figure 11. Problem setup for Example 3 in Excel

We compute the solution by evaluating the following PDSOLVE formula

=PDSOLVE(A1, (t,x,U1:U3), B1, C1, {0,1}, {0,.4,.6,1,2,3}) (16)

in allocated range A5:G27, and obtain the default snapshot solution format shown in Fig 12. Note that we have requested the solution be reported at the specific time

points provided in argument 6, whereas the spatial points are reported at uniform intervals according to 21 available rows in the allocated solution range.

	A	B	C	D	E	F	G
5	t	0.0	0.4	0.6	1.0	2.0	3.0
6	x	u	u	u	u	u	u
7	0.00	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
8	0.05	0.15643	0.06297	0.04891	0.03324	0.01448	0.00592
9	0.10	0.30902	0.12565	0.09764	0.06631	0.02875	0.01171
10	0.15	0.45399	0.18776	0.14601	0.09904	0.04261	0.01727
11	0.20	0.58779	0.24897	0.19379	0.13120	0.05584	0.02247
12	0.25	0.70711	0.30891	0.24074	0.16256	0.06820	0.02720
13	0.30	0.80902	0.36715	0.28653	0.19278	0.07944	0.03134
14	0.35	0.89101	0.42315	0.33074	0.22145	0.08932	0.03481
15	0.40	0.95106	0.47622	0.37282	0.24803	0.09754	0.03749
16	0.45	0.98769	0.52547	0.41200	0.27182	0.10382	0.03931
17	0.50	1.00000	0.56965	0.44721	0.29191	0.10788	0.04019
18	0.55	0.98769	0.60706	0.47695	0.30714	0.10942	0.04010
19	0.60	0.95106	0.63530	0.49913	0.31607	0.10821	0.03899
20	0.65	0.89101	0.65105	0.51089	0.31702	0.10405	0.03688
21	0.70	0.80902	0.64973	0.50845	0.30811	0.09684	0.03378
22	0.75	0.70711	0.62542	0.48724	0.28750	0.08657	0.02976
23	0.80	0.58779	0.57124	0.44229	0.25375	0.07339	0.02491
24	0.85	0.45399	0.48074	0.36953	0.20631	0.05760	0.01934
25	0.90	0.30902	0.35091	0.26782	0.14609	0.03968	0.01322
26	0.95	0.15643	0.18602	0.14122	0.07583	0.02024	0.00671
27	1.00	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000

Figure 12. Computed solution for Example 3 by PDSOLVE

Fig 13 shows the solution plot at the requested time points. Comparing values of Fig 12 to the exact values of Table 2 shows the maximum numerical error is on the order of 1e-05.

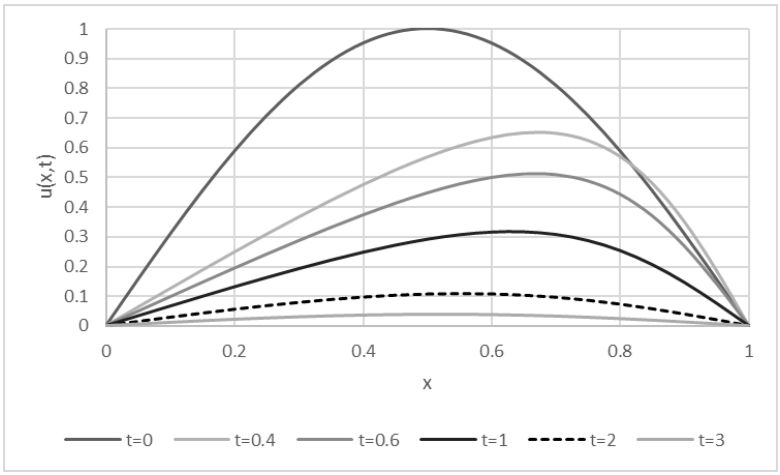

Figure 13. Plot of Fig 12 results showing effects of convection-diffusion of initial sinusoidal condition at various times. Viscosity = 0.1

To compute the solution of Burgers equation (10) for $\nu = 0.01$, we simply change the value of cell D1 in Fig 11 to 0.01. Excel automatically computes the new solution and updates the plot which is shown in Fig 14. Although not shown here, the maximum numerical error in comparison to Table 2 was again on the order of 1e-05.

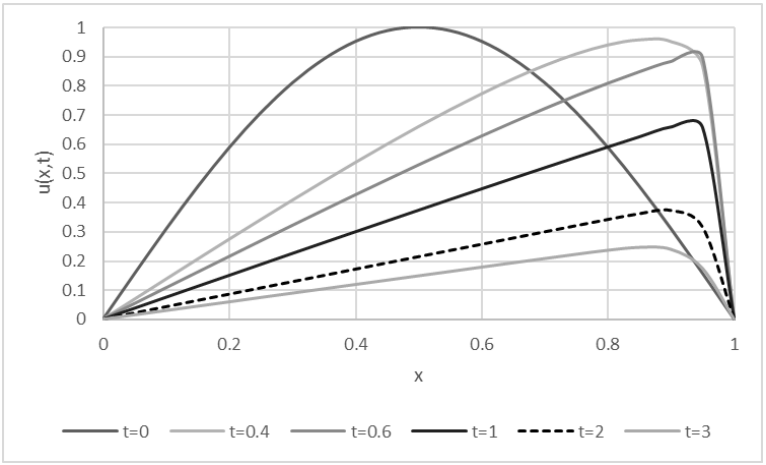

Figure 14. Plot showing effects of convection-diffusion of initial sinusoidal condition at various times. Viscosity = 0.01

V. CONCLUSION

We exploited Excel's computing engine to develop a novel worksheet solver for 1D partial differential equations. Design of the solver was made possible by bypassing inherent restrictions in order to allow a worksheet function to receive and evaluate formulas via input arguments. Several examples were presented to demonstrate the merits of the new design which encapsulates the algorithm, and separates problem model from results, in contrast to standard methods which mix inputs, algorithm and results.

Although we do not provide benchmark performance data in this article, we comment that all the preceding examples compute on the order of a second or less, on a typical computer with an Intel core i5 processor. The solver and several other calculus and ODE spreadsheet solvers are available in an add-in software library at excel-works.com [7], which integrates seamlessly with Excel Spreadsheet.

APPENDIX

A1 BASIC SPREADSHEET CONCEPTS

A typical worksheet in Excel is composed of a large structured grid. Each cell in the grid is referenced by its column label and row number, e.g., A1, and represents a global memory placeholder. A range of cells can be referenced as a rectangular array, e.g., A1:B3, or a union of disjoint arrays and cells, e.g., (X1, A1:A3). A cell may store a constant value or a formula defined using basic spreadsheet syntax, e.g., '= SQRT (X1^2 + Y1*Y1)'. The spreadsheet engine insures orderly evaluation of all dependent formulas upon a change in the value of any cell. A general function can thus be identified by a root formula and a list of variable cells. Nested dependency allows arbitrarily complex functions to be constructed. To motivate the possibilities, consider the formula '=SUM (X1:Z1)' assigned to A1, the pair (A1, Y1) identifies the function f(y)=X1+y+Z1, where X1 and Z1 are treated as constant values. In another example, consider the formula '=1+COS(B1)' assigned to A1, and the formula '=SQRT(ABS(X1))' assigned to B1, the pair (A1, X1) identifies the function $f(x)=1+\cos(\sqrt{|x|})$.

Excel supports two types of formulas: simple formulas and array formulas. A simple formula is assigned to one cell and evaluates to a single value, e.g.,'=SUM (A1:B4)'. Alternatively, an array formula is assigned to a range of cells and evaluates to an array of values, e.g., '=MINVERSE (A1:C3)' which computes the inverse of the 3 by 3 matrix A1:C3.

REFERENCES

[1] Robert L. Sternberg (Editor), Anthony J. Kalinowski (Editor), John S. Papadakis (Editor), "Nonlinear Partial Differential Equations in Engineering and Applied Science (Lecture Notes in Pure and Applied Mathematics)," CRC Press; 1 edition (June 1, 1980).

[2] Suren Basov, "Partial Differential Equations in Economics and Finance," Nova Science Pub Inc (October 29, 2007).

[3] Chung-Yau Lam and F. H. Alan Koh, "A Partial Differential Equation Solver for the Classroom," Int. J. Engng Ed.Vol. 22, No. 4, pp. 868-875, 2006.

[4] Hagler, Marion, "Spreadsheet Solution of Partial Differential Equations," IEEE Transactions on Education, Volume:E-30 Issue:3

[5] C. Ghaddar, "Method, Apparatus, and Computer Program Product for Optimizing Parameterized Models Using Functional Paradigm of Spreadsheet Software," USA Patent No. 9286286.

[6] C. Ghaddar, "Unconventional Calculus Spreadsheet Functions", World Academy of Science, Engineering and Technology, International Science Index 112. International Journal of Mathematical, Computational, Physical, Electrical and Computer Engineering (2016), 10(4), 160 – 166. http://waset.org/publications/10004374

[7] C. Ghaddar, "ExceLab Reference Manual", https://excel-works.com

[8] C. Ghaddar, "Modeling and Optimization of Dynamical Systems by Unconventional Spreadsheet Functions", American Journal of Modeling and Optimization. Vol. 4, No. 1, 2016, pp 1-12. http://pubs.sciepub.com/ajmo/4/1/1

[9] Schiesser W.E (1991). The Numerical Method of Lines, San Diego, CA: Academic Press, 1991.

[10] U. M. Ascher, R. M. Mattheij and R. D. Russell, "Numerical Solution of Boundary Value Problems for Ordinary Differential Equations," SIAM, 1995.

[11] E Hairer and G Wanner, "Solving Ordinary Differential Equations II: Stiff and Differential-Algebraic Problems," Springer Series in Computational Mathematics, 1996.

[12] A. C. Hindmarsh, "ODEPACK, A Systematized Collection of ODE Solvers," in Scientific Computing, R. S. Stepleman et al. (Eds.), North-Holland, Amsterdam, 1983, pp. 55-64.

[13] S. Kutluay, A.R. Bahadir, A. Ozdes, Numerical solution of one-dimensional Burgers equation, explicit and exact-explicit finite difference method. Journal of Computational and Applied Mathematics 103 (i 999) 251-261