Unconventional Calculus Spreadsheet Functions

Chahid K. Ghaddar

Abstract—The spreadsheet engine is exploited via a non-conventional mechanism to enable novel worksheet solver functions for computational calculus. The solver functions bypass inherent restrictions on built-in math and user defined functions by taking variable formulas as a new type of argument while retaining purity and recursion properties. The enabling mechanism permits integration of numerical algorithms into worksheet functions for solving virtually any computational problem that can be modelled by formulas and variables. Several examples are presented for computing integrals, derivatives, and systems of deferential-algebraic equations. Incorporation of the worksheet solver functions with the ubiquitous spreadsheet extend the utility of the latter as a powerful tool for computational mathematics.

Keywords—Calculus functions, nonlinear systems, differential algebraic equations, solvers, spreadsheet.

I. Introduction

THE ubiquitous spreadsheet application, primarily Excel, is widely used by professionals for diverse applications in business [1], engineering and science [2], [3]. It is also commonly used in the classroom as a math tool, thanks in part to its ease of use, rich built-in mathematical and statistical functions, and graphing tools. There are over 500 built in intrinsic functions in Excel grouped in a dozen categories, including engineering, math, financial, statistical, etc. The author was intrigued by the logical question: why the spreadsheet has never offered a calculus category among its math functions.

Calculus is devoted to the study of functions which the spreadsheet offers an intuitive interface for defining (i.e., formulas). It would thus present an ideal platform for supporting built-in calculus functions. (Here we use the term `function' in its strict mathematical sense that is a preserving map from input to output, i.e., a function does not modify its input and produces no side effects.) For instance, one can conceive of a pure worksheet integration function that takes a formula and limits as inputs and computes its accurate integral, much like a built-in math function takes a number and computes its square root. Evidently, people have come up with all sorts of procedures to approximate a formula integral, from implementing trapezoidal summing integration rules on discrete data in the spreadsheet to writing custom VBA programs [4], [5]. Not only are such procedures generally inaccurate, but they are inherently limited in scope, and reusability.

Introducing worksheet calculus functions to the spreadsheet requires the ability of the latter to support first class functions, that is, functions that can take other functions as arguments

C. Ghaddar is with ExcelWorks LLC, Sharon, MA 02067 USA (e-mail: cghaddar@excel-works.com).

while preserving mathematical properties. It turns out, unfortunately, that the spreadsheet's inherent design does not naturally support the creation of first class worksheet functions. Specifically, there are two distinct venues for adding functionality to a spreadsheet: commands and functions [6]. A command is the standard mechanism for evaluating formulas in the spreadsheet: values for the independent cells are changed and the dependent formulas cells are recalculated. A command works by mutating its own inputs and does not constitute a mathematical function. On the other hand, the spreadsheet restricts intrinsic and user defined functions that can be invoked in formulas to operate on constant inputs only, and unlike commands, grants them limited access to its features [7], [8]. As such a user defined function cannot evaluate formulas or change any data in the spreadsheet.

The benefits to be gained by overcoming the spreadsheet's restrictions on worksheet function input types are noteworthy. For one, the spreadsheet's computational engine could be exploited to support calculus functions such as integration, differentiation, and solvers for virtually any system defined by functions (e.g., differential equations), in an intuitive manner. Second, a worksheet solver encapsulates its underlining algorithm and 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, 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. More importantly, by preserving function properties such as purity and recursion, the functions can achievably support a functional paradigm for solving more complex problems implicating multiple topics and solvers such as dynamical optimization or optimal control [9]. As commands, (the only mean for evaluating formulas) do not possess essential properties to support a functional paradigm, solving dynamical optimization problems have remained outside the scope of traditional spreadsheet applications.

Accordingly, the author has developed a method which supports the creation of worksheet first class functions in the spreadsheet while preserving essential properties of purity and recursion. Details of the method are provided in [10] and are rather technical in nature as they relate to the spreadsheet object model, Advanced Programming Interface (API), and topics in computer science. The main idea of the method, however, is to capture the definition of a function's input formula using the spreadsheet API and construct 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

World Academy of Science, Engineering and Technology
International Journal of Mathematical and Computational Sciences
Vol:10, No:4, 2016

interdependence based on the supplied values of the variables, and aggregating the values of the nodes to obtain the value of the input formula all without modifying any data in the spreadsheet. The method has been employed, according to the flowchart of Fig. 1, to create several novel worksheet functions for solving a variety of computational calculus problems [12].
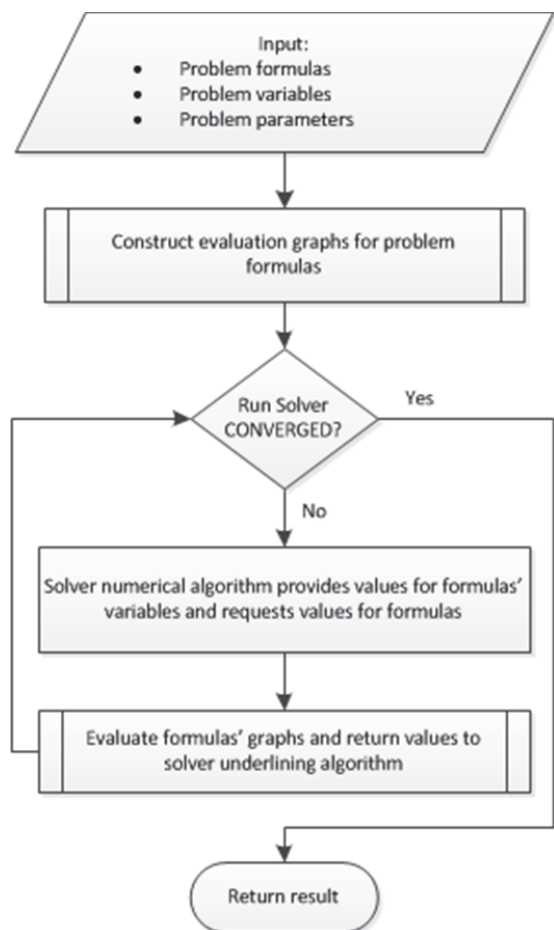

Fig. 1 Flowchart for worksheet calculus function design

The remainder of this article is divided into three sections, with each section dedicated to presenting selected worksheet solvers and examples for the following topics:

- Computing integrals and derivatives of any order,
- Solving nonlinear systems of algebraic equations,
- Solving ordinary differential algebraic equations.

We remark that the focus of the paper is to illustrate the application of the new spreadsheet solution methodology rather than analyze any specific problem. Furthermore, it is not our intent here to provide a review of other non-spreadsheet tools capable of solving similar problems. Instead the reader can withdraw own conclusion on the merits of this approach in comparison to other familiar mathematical software. Finally, we recommend reviewing Appendix A1 which includes a brief description of basic spreadsheet concepts for any reader not familiar with spreadsheet prior to reviewing the worksheet solvers.

II. INTEGRALS AND DERIVATIVES

We begin by introducing the worksheet numerical integration function

$$=QUADF (f, x, a, b, [Options]) \tag{1}$$

for computing definite or improper one-dimensional integrals $\int_a^b f(x)dx$. QUADF implements the algorithms in the package QUADPACK [11] which include fixed-order, and adaptive algorithms suitable for smooth and irregular integrands. In addition to the required parameters f, x, a, and b, QUADF accepts optional arguments for algorithm selection control, and for supplying any known singularities as described in [12]. We demonstrate QUADF by computing the following integral:

$$\int_0^1 \frac{\ln x}{\sqrt{x}} dx = -4 \tag{2}$$

The integrand formula in (2) is defined in cell A1 using X1 as variable as shown in Fig. 2. Note that Excel reports an error in A1 since X1 is undefined and defaults to zero. This error can be ignored since X1 merely serves as a dummy variable and its value is irrelevant. The integration formula is defined in A2, passing in the integrand formula, the variable of integration, and values for the limits. Evaluating A2 yields the result. Note, that since QUADF is a pure function, it does not modify its input or any data in the spreadsheet other than displaying the result in its own cell A2.

Fig. 2 Computing integral (2) in Excel

Using recursion, multiple integrals of any order can be computed by a direct nesting of QUADF, as demonstrated by the following volume integral example:

$$\int_0^2 dx \int_0^{3-\frac{3}{2}x} dy \int_0^{6-3x-2y} 1 - x \ dz = 3 \tag{3}$$

To compute (3), we construct a simple functional program consisting of three nested calls of QUADF, as shown in Fig. 3. Using X1, Y1 and Z1 as dummy variables, the integrand formula is defined in A1, and the inner, middle, and outer integrals formulas are inserted in A2, A3 and A4 respectively, with each inner QUADF formula serving as the integrand for the next outer QUADF formula. Evaluating the outer integral in A4 computes the triple integral value.

World Academy of Science, Engineering and Technology
International Journal of Mathematical and Computational Sciences
Vol:10, No:4, 2016

	A
1	=1-X1
2	=QUADF(A1,Z1,0,6-3*X1-2*Y1)
3	=QUADF(A2,Y1,0,3-3*X1/2)
4	=QUADF(A3,X1,0,2)

	A
1	1
2	6
3	9
4	3

Fig. 3 Computing triple integral (3) in Excel

Using the same procedure, we can as easily compute improper integrals such as the double integral (4) by specifying the strings "inf" or "-inf" for infinite limits as demonstrated in Fig. 4.

$$\int_0^\infty dx \int_x^\infty e^{-x-2y}\, dy = \frac{1}{6} \qquad (4)$$

	B
1	=EXP(-X1-2*Y1)
2	=QUADF(B1,Y1,X1,"inf")
3	=QUADF(B2,X1,0,"inf")

	B
1	1
2	0.5
3	0.1666666667

Fig. 4 Computing improper integral (4) in Excel

Likewise, we introduce the numerical differentiation worksheet function:

$$=DERIVF(f,\, x,\, p\,, n,\, [options]) \qquad (5)$$

to compute the nth exact derivative of a function $f(x)$ at a specified point p, $\frac{d^n}{dx^n} f(x = p)$. DERIVF implements Ridders' algorithm [13], [14] which uses an adaptive step size to produce superior accuracy compared to a simple finite difference scheme. Algorithm settings can be adjusted via the optional arguments [12]. We illustrate DERIVF in Fig. 5 by computing the derivative for the following formula at the point $x= 0.5$.

$$\frac{d}{dx}\frac{\ln x}{\sqrt{x}}\Big|_{0.5} = (x^{-\frac{3}{2}} - \frac{1}{2}x^{-\frac{3}{2}}\sqrt{x}\ln x)\big|_{0.5} \cong 3.808685268 \qquad (6)$$

	A
1	=LN(X1)/SQRT(X1)
2	=DERIVF(A1,X1,0.5)

	A
1	#NUM!
2	3.808685268

Fig. 5 Computing derivative (6) in Excel

Applying recursion again, DERIVF can be employed to compute a mixed partial derivative for any formula in Excel as demonstrated in Fig. 6 for the following example:

$$\frac{\partial}{\partial y}\frac{\partial}{\partial x}\cos(xy)\,\big|_{(\pi,\pi)} = -\sin(\pi^2) - \pi^2\cos(\pi^2)$$
$$\cong 9.3394486379 \qquad (7)$$

	B
1	=COS(X1*Y1)
2	=DERIVF(B1,X1,PI())
3	=DERIVF(B2,Y1,PI())

	B
1	1
2	0
3	9.3394486379

Fig. 6 Computing mixed partial derivative (7) in Excel

III. NONLINEAR ALGEBRAIC SYSTEMS

To compute the least-squares solution to a system of algebraic equations and inequalities, we require the system be presented in the following ordered form in which any inequalities are listed last:

$$\begin{aligned} f_i(\boldsymbol{x}) &= 0, & i = 1, k \\ f_i(\boldsymbol{x}) &\geq 0, & i = k+1, m \end{aligned} \qquad (8)$$

To solve the system (8) we introduce the worksheet solver function NLSOLVE:

$$=NLSOLVE\,(lhs,\, vars,\, [ineq],\, [options]) \qquad (9)$$

NLSOLVE is passed references to the system LHS formulas f_i, the variables, and the number of inequalities. The system analytic Jacobian and algorithm settings may be supplied via optional parameters [12]. NLSOLVE employs the Levenberg-Marquardt algorithm [15], [16] to find optimal values for the system variables, $\boldsymbol{x}$, by minimizing an implicit objective function representing the sum of squares of the equations and active inequalities. We demonstrate using NLSOLVE for solving the following system which has the solution (1, 10, 1, 5, 4, 3) [17]:

$$\begin{aligned} &x_3 e^{-0.1x_1} - x_4 e^{-0.1x_2} + x_6 e^{-0.1x_5} - e^{-0.1} + 5e^{-1} - 3e^{-0.4} \\ &\qquad = 0 \\ &x_3 e^{-0.2x_1} - x_4 e^{-0.2x_2} + x_6 e^{-0.2x_5} - e^{-0.2} + 5e^{-2} - 3e^{-0.8} \\ &\qquad = 0 \\ &x_3 e^{-0.3x_1} - x_4 e^{-0.3x_2} + x_6 e^{-0.3x_5} - e^{-0.3} + 5e^{-3} - 3e^{-1.2} \\ &\qquad = 0 \\ &x_3 e^{-0.4x_1} - x_4 e^{-0.4x_2} + x_6 e^{-0.4x_5} - e^{-0.4} + 5e^{-4} - 3e^{-1.6} \\ &\qquad = 0 \\ &x_3 e^{-0.5x_1} - x_4 e^{-0.5x_2} + x_6 e^{-0.5x_5} - e^{-0.5} + 5e^{-5} - 3e^{-2} \\ &\qquad = 0 \\ &x_3 e^{-0.6x_1} - x_4 e^{-0.6x_2} + x_6 e^{-0.6x_5} - e^{-0.6} + 5e^{-6} - 3e^{-2.4} \\ &\qquad \geq 0 \end{aligned} \qquad (10)$$

We define the system formulas in Excel as shown in Fig. 7, and compute the solution by evaluating the following NLSOLVE formula in an allocated range B1:C7:

$$=NLSOLVE\,(A1:A6,\, X1:X6,\, 1) \qquad (11)$$

passing in the formulas, the variables which are seeded by an initial guess of one, and the number 1 to indicate the last formula in the argument A1:A6 represents an inequality.

World Academy of Science, Engineering and Technology
International Journal of Mathematical and Computational Sciences
Vol:10, No:4, 2016

	A
1	=X3*EXP(-0.1*X1)-X4*EXP(-0.1*X2)+X6*EXP(-0.1*X5)-EXP(-0.1)+5*EXP(-1)-3*EXP(-0.4)
2	=X3*EXP(-0.2*X1)-X4*EXP(-0.2*X2)+X6*EXP(-0.2*X5)-EXP(-0.2)+5*EXP(-2)-3*EXP(-0.8)
3	=X3*EXP(-0.3*X1)-X4*EXP(-0.3*X2)+X6*EXP(-0.3*X5)-EXP(-0.3)+5*EXP(-3)-3*EXP(-1.2)
4	=X3*EXP(-0.4*X1)-X4*EXP(-0.4*X2)+X6*EXP(-0.4*X5)-EXP(-0.4)+5*EXP(-4)-3*EXP(-1.6)
5	=X3*EXP(-0.5*X1)-X4*EXP(-0.5*X2)+X6*EXP(-0.5*X5)-EXP(-0.5)+5*EXP(-5)-3*EXP(-2)
6	=X3*EXP(-0.6*X1)-X4*EXP(-0.6*X2)+X6*EXP(-0.6*X5)-EXP(-0.6)+5*EXP(-6)-3*EXP(-2.4)

Fig. 7 Definition of system (10) formulas in Excel

Since NLSOLVE formula (11) computes an array of numbers, it must be executed as an array formula in the allocated range by pressing the keys CTRL+SHIFT+ENTER. The solver computes the solution shown in Fig. 8.

	B	C
1	X1	1
2	X2	10
3	X3	1
4	X4	5
5	X5	4
6	X6	3
7	SSERROR	5.86E-29

Fig. 8 Solution computed by (11) in Excel

IV. ORDINARY DIFFERENTIAL EQUATIONS

By the same method, we developed pure worksheet functions for solving a general ordinary differential algebraic system (DAE). We require the system be represented as a set of first order ODEs followed by any algebraic equations as:

$$\frac{du_i}{dx} = f_i(x, \boldsymbol{u}, \boldsymbol{y}), \quad i = 1, n$$
$$0 = g_j(x, \boldsymbol{u}, \boldsymbol{y}), \quad j = 1, m \tag{12}$$

where $\boldsymbol{u}$ are the differential variables, and $\boldsymbol{y}$ are the algebraic variables. We define the DAE system (12) in Excel by ordered RHS formulas $(f_1, .., f_n, g_1, .., g_m)$ and corresponding variables $(x, u_1, .., u_n, y_1, .., y_m)$, and represent the computed solution in a tabular array as shown in Fig. 9. In the solution layout, values for the independent variable are reported at uniform intervals according to the available number of rows in the allocated range that holds the solution. Alternatively, we can also report the solution at custom values for the independent variable via optional parameters to the solver [12]. Depending on the type of boundary conditions, the system (12) could describe either an initial value problem or a multi-point boundary value problem. We present below two worksheet solvers and examples for both types of problems.

	A	B	C	D
1	x	u_1	u_2	u_3
2				
3	Uniform or custom output values for independent variable		Corresponding solution values for dependent variables	
4				
5				
..				
N				

Fig. 9 Solution layout for DAE systems in Excel

A. Initial Value Problems

To computes the solution to an initial value DAE system (12) with initial conditions $u_i(0) = a_i$, $y_j(0) = b_j$, and over the interval $x \in [0\ T]$ we introduce the worksheet solver function IVSOLVE:

$$=IVSOLVE\ (rhs, vars, interval, m, [options]) \tag{13}$$

References to the system RHS formulas are supplied via rhs, and the system variables are seeded with initial conditions and supplied via $vars$, The integration interval is defined in the third parameter, $interval$, and the number of algebraic constraints is supplied in m. IVSOLVE implements several integration schemes [18], [19] suitable for stiff and smooth problems. Algorithmic control an optional system analytic Jacobian can be supplied via $[options]$ [12]. We illustrate IVSOLVE by solving the index 1 DAE system given in (14) on the interval $t \in [0, 1000]$ starting from initial conditions $y_1 = 1, y_2 = 0, y_3 = 0$.

$$\frac{dy_1}{dt} = -0.04y_1 + 10^4 y_2 y_3$$
$$\frac{dy_2}{dt} = 0.04y_1 - 10^4 y_2 y_3 - 3*10^7 y_2^2 \tag{14}$$
$$0 = y_1 + y_2 + y_3 - 1$$

The system RHS formulas are defined in cells A1:A3 using T1 for the time variable, and Y1, Y2, Y3 for the state variables with the specified initial conditions as shown in Fig. 10.

	A		Y
1	=-0.04*Y1+10000*Y2*Y3	1	1
2	=0.04*Y1-10000*Y2*Y3-30000000*Y2^2	2	0
3	= Y1+Y2+Y3-1	3	0

Fig. 10 Definition of system (14) in Excel

Next the IVSOLVE formula (15) is executed in the range C1:F22 which computes and displays the solution shown partially in Fig. 11, and plotted in Fig. 12.

$$=IVSOLVE\ (A1:A3,(T1,Y1:Y3),\{0,1000\},1) \tag{15}$$

World Academy of Science, Engineering and Technology
International Journal of Mathematical and Computational Sciences
Vol:10, No:4, 2016

	C	D	E	F
1	T1	Y1	Y2	Y3
2	0	1	0	0
3	50	0.69288	8.34415E-06	0.307111
4	100	0.617245	6.15388E-06	0.382748
5	150	0.570229	5.12407E-06	0.429765
--	--	--	--	--
20	900	0.349743	2.13047E-06	0.650255
21	950	0.343131	2.06992E-06	0.656867
22	1000	0.336882	2.01377E-06	0.663116

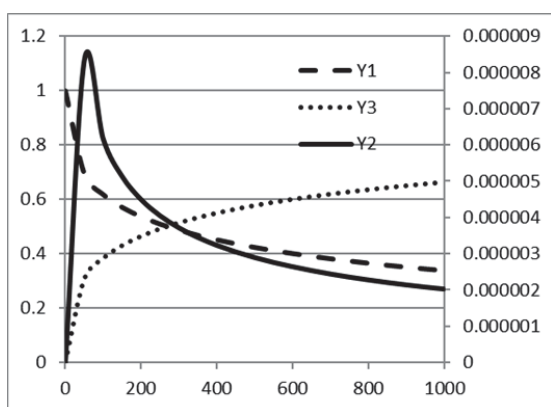Fig. 11 Solution computed by (15) in Excel

Fig. 12 Plot of solution computed by (15)

B. Boundary Value Problems

To compute solutions to multi-point boundary-value DAE systems (12), we introduce the worksheet solver function BVSOLVE:

$$=BVSOLVE\ (rhs,\ vars,\ bpts,\ bcs,\ interval,\ m,\ [options]) \tag{16}$$

BVSOLVE implements the COLDAE collocation algorithm [20], [21]. The steps for defining a boundary value problem in Excel are similar to the initial value problem, but requires additional input to specify the boundary points and associated conditions which are supplied via $bpts$ and bcs respectively. Algorithmic control, as well as Jacobian matrices, can be supplied via optional parameters [12].

We demonstrate BVSOLVE for the following nonlinear stiff differential equation [22]:

$$\varepsilon\,A(x)yy'' - \left(\frac{2.4}{2} - \varepsilon A'(x)\right)yy' + \frac{y'}{y} + \frac{A'(x)}{A(x)}\left(1 - \frac{0.4}{2}y^2\right) = 0$$

$$A(x) = 1 + x^2$$
$$y(0) = 0.9129, \qquad y(1) = 0.375$$
$$0 \leq x \leq 1 \tag{17}$$

Using a standard substitution, we convert the 2nd order equation (17) to two 1st order equations:

$$\frac{dy}{dx} = z$$
$$\frac{dz}{dx} = \frac{1}{\varepsilon\,A(x)y}$$
$$\left(\left(\frac{2.4}{2} - \varepsilon A'(x)\right)yz - \frac{z}{y} - \frac{A'(x)}{A(x)}\left(1 - \frac{0.4}{2}y^2\right)\right) \tag{18}$$

To model (18) in Excel, we define the RHS formulas in C1:C2 using X1 for the free variable and, Y1 and Z1 for the differential variables, as shown in Fig. 13.

	C
1	=Z1
2	=((2.4/2-E1*B1)*Y1*Z1-Z1/Y1-B1/A1*(1-0.4/2*Y1^2))/(E1*A1*Y1)

Fig. 13 Definition of system (18) formulas in Excel

In the RHS formulas, we make references to three additional cells E1, A1 and B1. These cells define formulas for $\varepsilon, A(x),\ and\ A'(x)$, respectively, as shown in Fig. 14. We could have substituted these values directly into the RHS formulas C1 and C2; however, this permits us to vary the definitions for these parameters later, and Excel automatically recalculates a new solution.

	C
1	=Z1
2	=((2.4/2-E1*B1)*Y1*Z1-Z1/Y1-B1/A1*(1-0.4/2*Y1^2))/(E1*A1*Y1)

Fig. 14 Definitions for $\varepsilon, A(x), A'(x)$ of system (18)

Next, we define the boundary points and corresponding condition formulas in cells F1:F2 and G1:G2, respectively, as shown in Fig. 15. Note transformation (18) preserves the original boundary conditions assigned to the variable y.

	F	G
1	0	=Y1-0.9129
2	1	=Y1-0.375

Fig. 15 Boundary points and conditions for system (18)

Finally, since the system RHS formula involves division by the variable y, we need to start from a nonzero initial guess for Y1; therefore, we assign the value 1 for cell Y1 to avoid division by zero. This completes the definition for the boundary value problem (17) in Excel.

To solve the boundary value problem (17) we execute the following BVSOLVE formula:

$$=BVSOLVE\ (C1:C2,\ (X1,Y1,Z1),\ F1:F2,\ G1:G2,\ \{0,1\})) \tag{19}$$

in an allocated range H4:J25, passing in the system RHS formulas, variables, boundary points and conditions, and the

World Academy of Science, Engineering and Technology
International Journal of Mathematical and Computational Sciences
Vol:10, No:4, 2016

domain [0 1]. BVSOLVE computes and displays the solutions shown in Fig. 16 and plotted in Fig. 17.

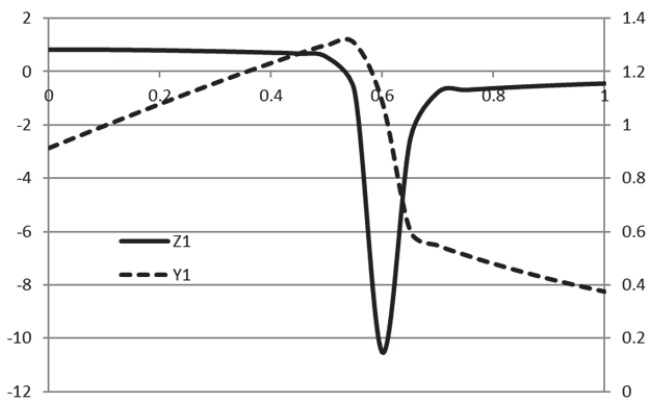|	H	I	J
4	X1	Y1	Z1
5	0	0.9129	0.835223
6	0.05	0.954646	0.834019
7	0.1	0.996237	0.828936
8	0.15	1.037468	0.819592
9	0.2	1.078129	0.806201
10	0.25	1.118026	0.789141
11	0.3	1.15699	0.7689
12	0.35	1.194872	0.745969
13	0.4	1.231542	0.72026
14	0.45	1.26676	0.684865
15	0.5	1.298649	0.549927
16	0.55	1.305552	-0.73972
17	0.6	1.083584	-10.5522
18	0.65	0.603002	-2.53229
19	0.7	0.547885	-0.76692
20	0.75	0.511759	-0.68353
21	0.8	0.479252	-0.6188
22	0.85	0.449708	-0.56442
23	0.9	0.422687	-0.51745
24	0.95	0.397869	-0.47612
25	1	0.375	-0.43931

Fig. 16 Solution computed by (19) in Excel

Fig. 17 Plot of solution computed by (19) for boundary value system (17)

V. CONCLUSION

Excel's computing engine was exploited to develop a novel set of worksheet calculus solver functions with no precedent in spreadsheet utility. Design of the solvers was made possible by bypassing inherent spreadsheet limitations that restricted functions to operating on constant inputs only, while retaining essential properties of purity and recursion. The solvers are assembled in an Add-In software library [12], which integrates seamlessly with MS Excel. Detailed descriptions, additional solvers and examples can be found in [12].

Several examples were presented to demonstrate the merits of the worksheet solvers, including simplified modeling, transparency with no hidden settings, or user dialogues, and separation of input, algorithms, and output which are inherently mixed when using conventional methods.

Recognizing that performance is central to the viability of any computational strategy, the solvers achieve competitive performance thanks to the direct coupling to the spreadsheet engine API. 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 effort invested in this development aims at offering intuitive and readily accessible advanced computing suited for both novices and experts. Future work will focus on extending this frame work to support a functional paradigm approach for problems in dynamical optimization and optimal control.

APPENDIX

A. 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 $f(y)=X1+y+Z$, 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] Laughbaum, Edward D., Seidel, Ken, "Business math Excel applications," Prentice Hall 2008.
[2] Larsen, R. W., "Engineering with Excel," Pearson Prentice Hall 2009, New Jersey. ISBN 0-13-601775-4
[3] Bourq, David M., "Excel scientific and engineering cookbook," O'Reilly, 2006
[4] E. J. Billo, Excel for Scientists and Engineers, WILEY-INTERSCIENCE, 2007
[5] Kim Gaik Tay, Tau Han Cheong, Nur Kamil Adli Mohd Nawar, Sie Long Kek, Rosmila Abdul-KaharA, "Romberg Integral Spreadsheet Calculator", Spreadsheets in Education (eJSiE), 2015
[6] Excel Commands, Functions, and States, MSDN publication, accessed 1/20/2016, https://msdn.microsoft.com/en-us/library/bb687832(v=office.15).aspx
[7] S. Dalton, Financial Applications using Excel Add-in Development in C/C++ , The Wiley Finance Series, 2007.

[8]   Description of limitations of custom functions in Excel, accessed 1/20/2016, https://support.microsoft.com/en-us/kb/170787
[9]   C. Ghaddar, "Modeling and Optimization of Dynamical Systems by Unconventional Spreadsheet Functions." American Journal of Modeling and Optimization. Vol. 4, No. 1, 2016.
[10]   C. Ghaddar, "Method, Apparatus, and Computer Program Product for Optimizing Parameterized Models Using Functional Paradigm of Spreadsheet Software," USA Patent No. 9286286.
[11]   R. Piessens, E. de Doncker-Kapenga, C.W. Ueberhuber, and D.K. Kahaner, "QUADPACK A subroutine package for automatic integration," Springer Verlag, 1983.
[12]   C. Ghaddar, "ExceLab Reference Manual", accessed 3/7/2016, www.excel-works.com
[13]   C.J.F. Ridders, Advances in Engineering Software, vol 4, 75-76, 1982.
[14]   Numerical Recipes in C: The Art of Scientific Computing, Cambridge University Press, 1992.
[15]   K. Levenberg, "A Method for the Solution of Certain Non-Linear Problems in Least Squares," Quarterly of Applied Mathematics vol 2, 164–168, 1944.
[16]   D. Marquardt "An Algorithm for Least-Squares Estimation of Nonlinear Parameters," SIAM Journal on Applied Mathematics vol 11 (2), 431–441, 1963.
[17]   J. More, B. S. Garbow, and K. E. Hillstrom, "Testing unconstrained optimization software," ACM Trans. Math. Softw, vol 7, 17-41, 1981
[18]   E Hairer and G Wanner, "Solving Ordinary Differential Equations II: Stiff and Differential-Algebraic Problems," Springer Series in Computational Mathematics, 1996.
[19]   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.
[20]   U. M. Ascher, R. M. Mattheij and R. D. Russell, "Numerical Solution of Boundary Value Problems for Ordinary Differential Equations," SIAM, 1995.
[21]   U. Ascher and R. Spiteri "Collocation software for boundary value differential-algebraic equations," SIAM Journal on Scientific Computing. 1994, 15,938-952.
[22]   K Soetaert, J. Cash, and F. Mazzia, Package bvpSolve, solving test problems, accessed 1/20/2016, http://www.ma.ic.ac.uk/~jcash/BVP_software/PROBLEMS.PDF