PDASOLVE( ): Excel's Advanced Solver for Partial Differential Algebraic Equations

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

Multi-region system tutorial

▼ Objective

In this tutorial we will simulate a model involving three implicitly coupled partial differential equations defined over three regions with different material properties. The equations describe a simplistic battery setup but retain many of the essential features of more complex models including:

- Flux continuity conditions across interfaces separating regions with different properties.
- Coupled time derivatives of state variables.

This tutorial has two main objectives:

1. To show you how to systematically model and solve a complex partial differential system in Excel with PDASOLVE().
2. To show you how to optimize your model parameters to accurately fit experimental measurements.

▼ Mathematical Model:

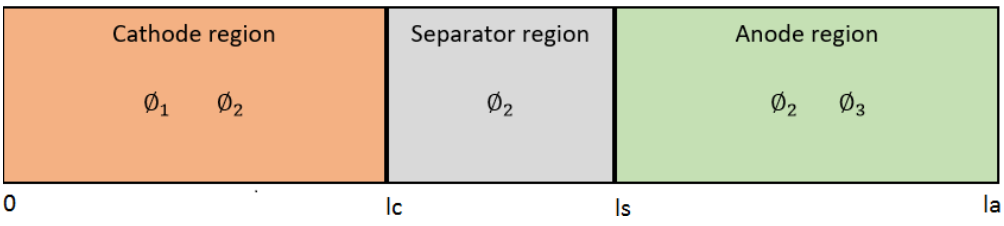

Figure 1: Three-region battery model.

The following model describes three coupled system of partial differential equations. Equation 1 is defined in the cathode region only (see Figure above). Equation 2 is defined in all three regions, and equation 3 is defined in the anode region only. The three equations are implicitly coupled by a mass matrix *M*.

$$m_{11}\frac{\partial \emptyset_1}{\partial t} + m_{12}\frac{\partial \emptyset_2}{\partial t} = \sigma(x)\,\emptyset_{1,xx} - a(x)(\emptyset_1 - \emptyset_2 - u(t,x)) \quad (1)$$

$$m_{21}\frac{\partial \emptyset_1}{\partial t} + m_{22}\frac{\partial \emptyset_2}{\partial t} + m_{23}\frac{\partial \emptyset_2}{\partial t} = \kappa(x)\,\emptyset_{2,xx} + a(x)(\emptyset_1 - \emptyset_2 - u(t,x)) \quad (2)$$

$$m_{32}\frac{\partial \emptyset_2}{\partial t} + m_{33}\frac{\partial \emptyset_3}{\partial t} = \sigma(x)\,\emptyset_{3,xx} - a(x)(\emptyset_3 - \emptyset_2 - u(t,x)) \quad (3)$$

where:

$$\sigma(x) = \begin{cases} 1 & if \quad 0 \le x \le l_c \\ 0 & if \quad l_c < x < l_s \\ 1 & if \quad l_s \le x \le l_a \end{cases}$$

$$\kappa(x) = \beta * \begin{cases} 1 & if \quad 0 \le x \le l_c \\ 2 & if \quad l_c < x < l_s \\ 1 & if \quad l_s \le x \le l_a \end{cases}$$

$$u(t,x) = \begin{cases} 3 - t^2 & if\ x \le l_c \\ 0 & else \end{cases}$$

$$a(x) = \gamma \begin{cases} 10^6 & if \quad 0 \le x \le l_c \\ 0 & if \quad l_c < x < l_s \\ 10^6 & if \quad l_s \le x \le l_a \end{cases}$$

$$M = \begin{bmatrix} m_{11} & m_{12} & m_{13} \\ m_{21} & m_{22} & m_{23} \\ m_{31} & m_{32} & m_{33} \end{bmatrix} = \begin{bmatrix} 10^8 & -m_c & 0 \\ -m_c & 10^8 & -m_c \\ 0 & -m_c & 10^8 \end{bmatrix}$$

Constant Parameters

$l_c = 0.0002$

$l_s = 0.0002$

$l_a = 0.0004254$

$\beta = 1$

$\gamma = 37.4$

$m_c = 10^8$

Although we can simply hardcode the numeric values of the system parameters directly into the equations; by assigning variables, we gain the ability to study the effect of these parameters on the system behavior. In the second part of this tutorial, we will show how to compute optimal values for β, γ and m_c which modify the system behavior for best fit with empirical data.

Initial Conditions

$$\emptyset_1(0,x) = \begin{cases} 3 & if\ x \le l_c \\ 0 & else \end{cases}$$

$\emptyset_2(0,x) = 0$

$\emptyset_3(0,x) = 0$

Boundary Conditions

The table below lists the boundary conditions at the spatial domain's end points and regions' interior interfaces for the partial equations systems (1)-(3). Two continuity conditions are required for the flux $q = -\kappa\, \emptyset_{2,x}$ at both $x = l_c$ and l_s because of the discontinuity in κ values across the different regions. Note that $\emptyset_1$ and $\emptyset_2$ are not defined in region 2 so we do not need to impose any continuity conditions for their fluxes at $x = l_c$ or l_s

$x = 0$	$x = l_c$	$x = l_s$	$x = l_a$
$\sigma(0)\, \emptyset_{1,x} = 1$	$\sigma(l_c)\, \emptyset_{1,x} = 0$	$\sigma(l_s)\, \emptyset_{3,x} = 0$	$\emptyset_3 = 0$
$\kappa(0)\, \emptyset_{2,x} = 0$	$\kappa(l_c)\, \emptyset_{2,x}\|^- = \kappa(l_c)\, \emptyset_{2,x}\|^+$	$\kappa(l_s)\, \emptyset_{2,x}\|^- = \kappa(l_s)\, \emptyset_{2,x}\|^+$	$\kappa(l_a)\, \emptyset_{2,x} = 0$

Table 1: Boundary conditions for the three-region model.

▼ Solution

+ Step 1: model definition in Excel

Simulating the above system in Excel with PDASOLVE() is straight forward. The first step is to define the model in Excel with standard formulas and variables. It is convenient to work with named cells and ranges rather than raw address. We start by naming the cells in the range B2:B12
as *t*, *x*, *phi_1*, *phi_2*, *phi_3*, *phi1x*, *phi2x*, *phi3x*, *phi1xx*, *phi2xx*, *phi3xx* to represent, respectively, the system's variables: $t, x, \phi_1, \phi_2, \phi_3, \phi_{1,x}, \phi_{2,x}, \phi_{3,x}, \phi_{1,xx}, \phi_{2,xx}, \phi_{3,xx}$. We also name the range B2:B12 itself as *SysVars*. Next we name the cells in the range B20:B26 as*lc*, *ls*, *la*, *sigma*, *kappa*, *a*, and *u* to represent the system's constant parameters and property functions: $l_c, l_s, l_a, \sigma, \kappa, a$ and, u. We also name cells C24, C25 for the parameters *beta*, *gamma*, and define the system mass matrix M in range C4:E6 which we name as *M*. (Note that we chose to parameterize the off diagonal terms of *M* for the purpose of the optimization exercise below. This will permit us to treat *mc* as a design variable.)

We are ready to define the model formulas in terms of our named variables. We assign values to the system's parameters *lc*, *ls* and *la*, and define formulas for the property functions *sigma*, *kappa*, *a* and *u*. Next, we define the system's three right hand side equations in range B15:B17which we name as *SysEqs*. Next to each equation, in range D15:E17, which we name as *Regions*, we specify region's end points for each pde equation. We also assign the initial condition formulas for state variables *phi_1*, *phi_2*, and *phi_3*. Figure 2 below shows our named Excel variables and formulas for the pde system: (the colored ranges represent input arguments to the solver PDASOLVE())

	A	B	C	D	E
1		**System variables**			
2	t			mc	1.00E+08
3	x		**M Mass matrix**		
4	phi_1	=u	1.00E+08	=-mc	0
5	phi_2	0	=-mc	1.00E+08	=-mc
6	phi_3	0	0	=-mc	1.00E+08
7	phi1x				
8	phi2x				
9	phi3x				
10	phi1xx				
11	phi2xx				
12	phi3xx				
13					

14		**System RHS Equations**		**Regions Definitions**	
15	EQ1	=sigma*phi1xx-a*(phi_1-phi_2-u)		0	=lc
16	EQ2	=kappa*phi2xx+a*(phi_1-phi_2 -u)		0	=la
17	EQ3	=sigma*phi3xx-a*(phi_3-phi_2-u)		=ls	=la
18					
19	**System Parameters/Functions**				
20	lc	2.000E-04			
21	ls	2.254E-04			
22	la	4.254E-04			
23	sigma	=IF(x<=lc,1,IF(x<ls,0,1))			
24	kapp	=beta*IF(x<=lc,1,IF(x<ls,2,1))	1	beta	
25	a	=gamma*IF(x<=lc,1E6,IF(x<ls,0,1E6))	37.4	gamma	
26	u	=IF(x<=lc,C26,0)	=3-t^2		

Figure 2: model definition in Excel. Colored ranges are input arguments to PDASOLVE()

Next we define the boundary conditions formulas using a 3-column range A29:C36 which we name as *BCs* as shown in Figure 3. The first column specifies the *x* locations of the boundary conditions, the 2nd column specifies the types which are identified by any of the letters 'D', 'N', 'R' or 'C', and the 3rd column specifies the boundary condition formulas which are arranged with respect to zero on one side.

	A	B	C
28	**BCs**		
29	0	R	=sigma*phi1x-1
30	0	R	=kappa*phi2x
31	=lc	R	=sigma*phi1x
32	=lc	C	=kappa*phi2x
33	=ls	C	=kappa*phi2x
34	=ls	R	=sigma*phi3x
35	=la	R	=kappa*phi2x
36	=la	D	=phi_3

Figure 3: boundary conditions definition

+ Step 2: running the solver

PDASOLVE() must be run as an array formula in a pre-allocated range. You can customize the reported output in the solution array by modifying parameters 4 and 5

which define the system spatial domain and time interval (please refer to this description of the solution layout and control). The default behavior, is to report the output for both time and space at uniform increments determined by the size of the allocated solution array. In our first run, we will rely on the default behavior. We evaluate PDASOLVE() formula as an array formula in the range A41:M53 (by pressing Ctrl+Shift+Enter), and obtain the solution shown in Figure 4 below instantaneously.

=PDASOLVE(**SysEqs**, **SysVars**, **BCs**, {0,0.0004254}, {0,1}, **M**, **Regions**, , {"FORMAT","TCOL1"}) (4)

In the 4rth and 5th parameters, we specify the problem's spatial domain and temporal end points (using Excel's constant arrays syntax for convenience). We skip over the 8th optional parameter (tolerances) to use defaults, and specify in the 9th optional parameter a control key/value pair (using constant array again for convenience) which instructs the solver to report the solution using the transient format in which the time is reported in column 1 of the output (A43:A53). This format makes it easier to plot transient views as opposed to the snapshot format (*x* in column 1).

	A	B	C	D	E	F	G	H	I	J	K	L	M
41	x	0	0	0	0.0002	0.0002	0.0002	0.0002254	0.0002254	0.0002254	0.0004254	0.0004254	0.0004254
42	t	phi_1	phi_2	phi_3	phi_1	phi_2	phi_3	phi_1	phi_2	phi_3	phi_1	phi_2	phi_3
43	0	3	0	0	3	0	0	0	0	0	0	0	0
44	0.1	2.99971531	-0.000138	0	2.9998153	-3.8E-05	0	0	-2.96955E-05	-2.55896E-05	0	-6.97E-06	0
45	0.2	2.99882537	-0.0001612	0	2.9989254	-6.12E-05	0	0	-5.16503E-05	-4.04538E-05	0	-1.302E-05	0
46	0.3	2.99650637	-0.0001779	0	2.9966064	-7.79E-05	0	0	-6.74764E-05	-4.93987E-05	0	-1.811E-05	0
47	0.4	2.9920726	-0.0001903	0	2.9921726	-9.03E-05	0	0	-7.92916E-05	-5.5026E-05	0	-2.243E-05	0
48	0.5	2.9848692	-0.0001997	0	2.9849692	-9.97E-05	0	0	-8.83963E-05	-5.86522E-05	0	-2.617E-05	0
49	0.6	2.97425031	-0.0002073	0	2.9743503	-0.000107	0	0	-9.57037E-05	-6.10849E-05	0	-2.946E-05	0
50	0.7	2.95961998	-0.0002135	0	2.95972	-0.000113	0	0	-0.000101657	-6.26738E-05	0	-3.24E-05	0
51	0.8	2.94038879	-0.0002186	0	2.9404888	-0.000119	0	0	-0.000106656	-6.37299E-05	0	-3.507E-05	0
52	0.9	2.91598511	-0.000223	0	2.9160851	-0.000123	0	0	-0.000110981	-6.44555E-05	0	-3.755E-05	0
53	1	2.88586891	-0.0002269	0	2.8859689	-0.000127	0	0	-0.000114784	-6.49463E-05	0	-3.988E-05	0

Figure 4: solution obtained by running PDASOLVE() array formula (4).

Figure 5 below shows transient plots for the system *phi_2* variable at different spatial points.

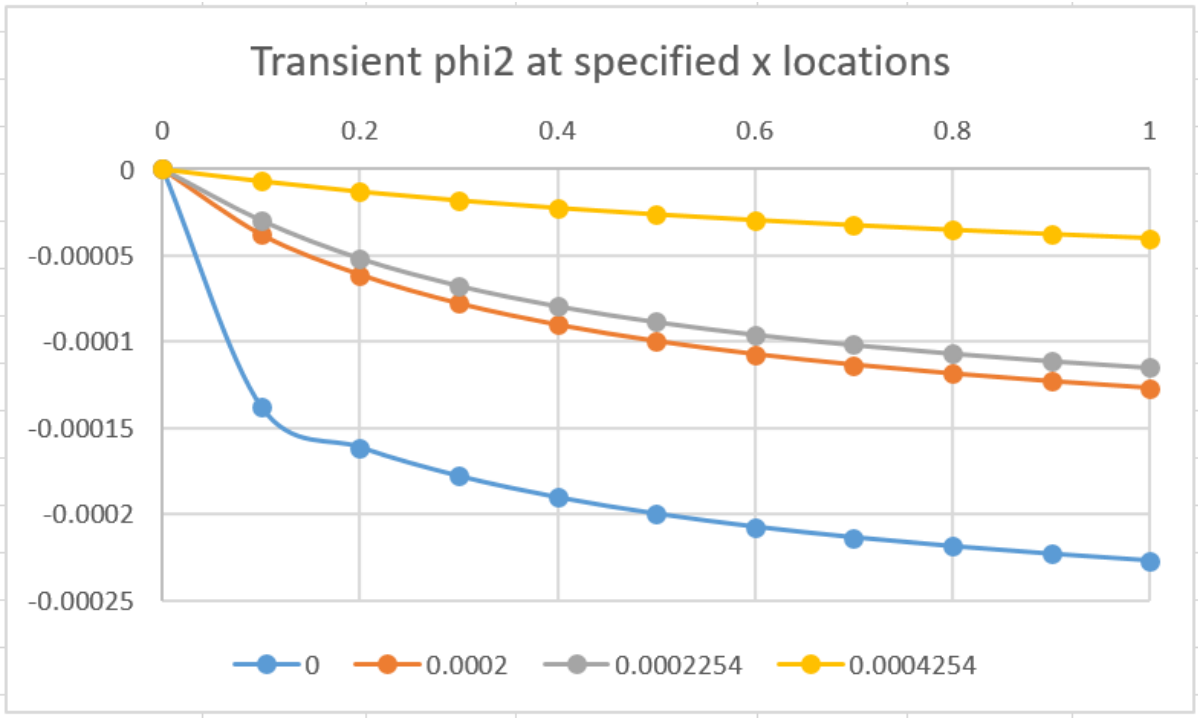

Figure 5: solution obtained by running PDASOLVE() array formula (4).

+ Verification of continuity boundary conditions

Below, we demonstrate additional features of the solver as well as verify that the continuity conditions:

$\kappa(l_c)\ \emptyset_{2,x}|^- = \kappa(l_c)\ \emptyset_{2,x}|^+$ at $x = lc$,

$\kappa(l_s)\ \emptyset_{2,x}|^- = \kappa(l_s)\ \emptyset_{2,x}|^+$ at $x = ls$

have been satisfied. To show this, we need to report the derivative variables in the output solution just before and after the $x = lc$, and ls. We can easily accomplish this by specifying a custom output spatial points for PDASOLVE() in argument 4. We define the desired output x locations in range B57:I57 which we name as *xout*. They include points just before and after *lc* and *ls*. Next, we define additional control key/value pairs in range A59:B62 which we name as *cntrls*. The keys include NDRVOUT with value 1 which instructs the solver to report the first derivative variables like *phi1x* in the solution. Specifying a value of 2 would instruct the solver to report 2nd derivatives variables like *phi1xx* as well. To increase the numerical accuracy, we define the key MAX_GRID_SPACING which ensures that the maximum distance between generated grid nodes does not exceed the specified value

of 1.0e-5. To avoid excessive number of grid node generation, we also specify a 1000 limit on the maximum number of generated nodes with the key MAX_GRID_NODES. We can also set relative tolerances for the algorithm for each variable if we desire. For example, we see from the solution above that *phi_1* is on the order of (1) but *phi_2* and *phi_3* are on the order (10e-5). In some cases it may be desirable to set different values for the convergence tolerance for each variable. To demonstrate, this we define three relative tolerance values for *phi_1*, *phi_2* and *phi_3* in range D59:D61 which we name *rtols*. The new defined ranges are shown in Figure 6 below:

	A	B	C	D	E	F	G	H	I
57	**xout**	0	0.0001998	0.00002	0.0002002	0.000225175	0.0002254	0.000225625	0.0004254
58	**Controls**			**Rel. Tol**					
59	FORMAT	TCOL1		1.00E-4					
60	NDRVOUT	1		1.00E-6					
61	MAX_GRID_SPACING	1.00E-05		1.00E-6					
62	MAX_GRID_NODES	1000							

Figure 6: optional input to PDASOLVE()

We evaluate the array formula in a new range and obtain instantaneously a new solution

=PDASOLVE(**SysEqs**, **SysVars**, **BCs**, **xout**, {0,1}, **M**, **Regions**, **rtols**, **ctrls**)

A partial listing of the solution is shown in Figure 7 below around $x = lc = 0.0002$:

0.0001998	0.0001998	0.0001998	0.0001998	0.0001998	0.0001998	0.0002	0.0002	0.0002	0.0002	0.0002	0.0002	0.0002	0.0002002	0.0002002	0.0002	0.0002002
phi_1	phi_2	phi_3	phi1x	phi2x	phi3x	phi_1	phi_2	phi_3	phi1x	phi2x	phi3x	phi_1	phi_2	phi_3	phi1x	phi2x
3	0	0	0	0	0	3	0	0	0	#NUM!	0	0	0	0	0	0
2.999814	-3.79875E-05	0	0.1013504	1.0145069	0	2.999814	-3.7855E-05	0	0	#NUM!	0	0	-3.78E-05	0	0	0.49724955
2.9989254	-6.11412E-05	0	0.0534586	1.0108751	0	2.998925	-6.0991E-05	0	0	#NUM!	0	0	-6.09E-05	0	0	0.49806212
2.9966068	-7.77552E-05	0	0.0434047	1.0112848	0	2.996607	-7.7606E-05	0	0	#NUM!	0	0	-7.75E-05	0	0	0.49857964
2.992173	-9.01352E-05	0	0.0416906	1.0116037	0	2.992173	-8.9986E-05	0	0	#NUM!	0	0	-8.99E-05	0	0	0.49892271
2.9849663	-9.96676E-05	0	0.0429016	1.0118346	0	2.984966	-9.9519E-05	0	0	#NUM!	0	0	-9.94E-05	0	0	0.49915705
2.974352	-0.000107236	0	0.0449714	1.0120213	0	2.974352	-0.00010709	0	0	#NUM!	0	0	-0.000107	0	0	0.49932122
2.9597249	-0.000113397	0	0.0470184	1.0121614	0	2.959725	-0.00011325	0	0	#NUM!	0	0	-0.000113	0	0	0.49943786
2.9404879	-0.00011858	0	0.0486628	1.0122627	0	2.940488	-0.00011843	0	0	#NUM!	0	0	-0.000118	0	0	0.49952362
2.9160842	-0.000123016	0	0.0499124	1.0123335	0	2.916084	-0.00012287	0	0	#NUM!	0	0	-0.000123	0	0	0.49958681
2.8859693	-0.000126896	0	0.0508069	1.0123813	0	2.885969	-0.00012675	0	0	#NUM!	0	0	-0.000127	0	0	0.49963418

Figure 7: partial listing of the solution obtained by evaluating PDASOLVE() array formula (5)

Note that the 1st derivative *phi2x* is not defined at *x=lc* because of the jump in the property *kappa* across the two different regions. As a result, *phi_2* is not a smooth function and its derivative is not defined at *x=lc*. A quick inspection of the ratio *phi2x-/phi2x+* shows that it is approximately 2.0 as expected since *phi2x-/phi2x+ = kappa(lc+)/kappa(lc-) = 2.0/1.0*. The ratios are shown in Figure 8 below for both *x=lc* and *ls* (at *x=ls* the ratio is 0.5 as *kappa* changes from 2 to 1)

Ratio at lc		Ratio at ls
2.040237		0.51498
2.029616		0.507006
2.028331		0.504454
2.027576		0.503231
2.027087		0.502535
2.026794		0.5021
2.026601		0.501813
2.026456		0.501613
2.026341		0.501472
2.026245		0.501369

Figure 8: ratios of phi2x-/phi2x+ at x=lc and ls

▼ Parametric Optimization

Figure 9 below shows a plot of *phi_2* at *x=lc* for the initial system's parameters *beta=1*, *gamma=37.4* and *mc=1.0e8*, along with actual measured values for *phi2* (in blue squares). Our objective is to find optimal values for *beta*, *gamma*, and *mc* so our model accurately fits the experimental values.

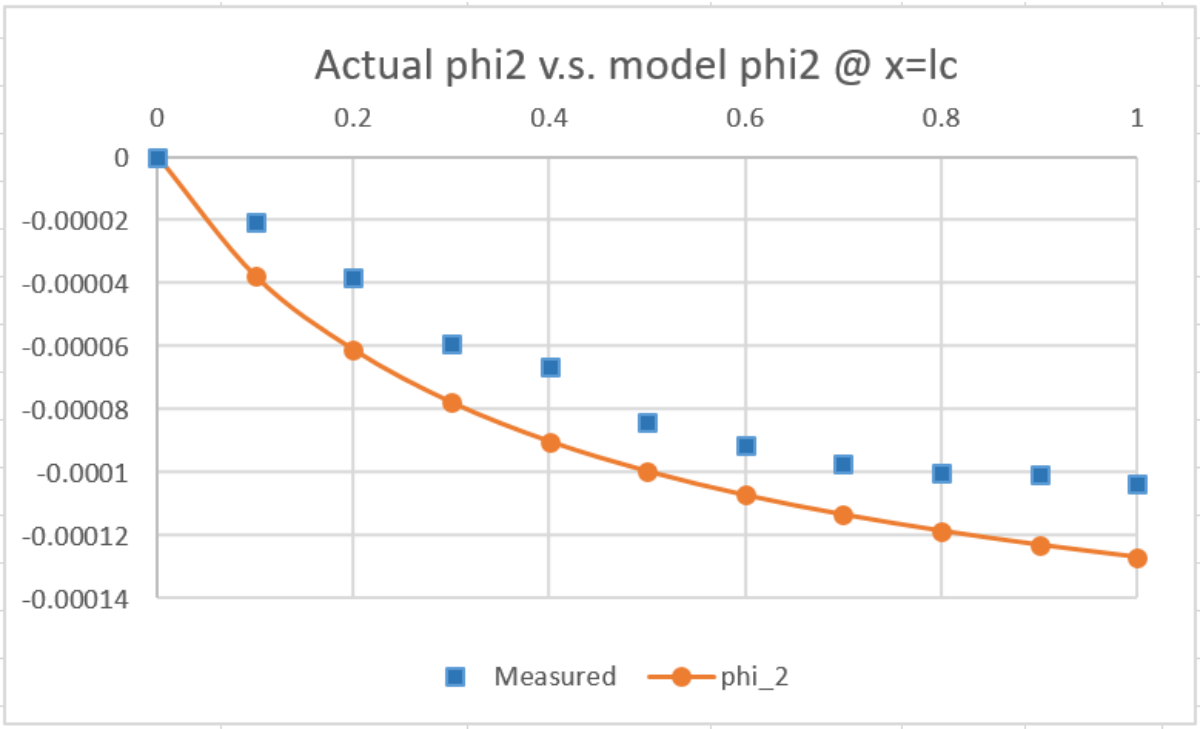

Figure 9: plots of phi_2 at x=lc along with empirical measurements in blue squares

Optimization problems are significantly streamlined in ExceLab with the aid of the criterion function ARRAYVAL() and the nonlinear solver NLSOLVE() requiring no more than a few additional formulas. The basic idea is to define one or more constraint formulas on the solution you have already obtained, then use NLSOLVE() to find optimal values for the design parameters which minimize the sum of square errors in the constraints.

For convenience, we show below a re-listing of the original solution obtained in Figure 4, along with the numerical values of the measured *phi2* values in range P43:P53, and our constraint definition in range R44:R53. (Corresponding constraint formulas as shown in Figure 10a below.)

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
41	x	0	0	0	0.0002	0.0002	0.0002	0.0002254	0.0002254	0.0002254	0.0004254	0.0004	0.000425					
42	t	phi_1	phi_2	phi_3	phi_1	phi_2	phi_3	phi_1	phi_2	phi_3	phi_1	phi_2	phi_3			phi2m(measured @ x=lc)		constraints
43	0	3	0	0	3	0	0	0	0	0	0	0	0			0		
44	0.1	2.999716	-0.000137	0	2.9998	-3.781E-05	0	0	-2.957E-05	-2.548E-05	0	-7E-06	0			-2.053E-05		-1.72821E-05
45	0.2	2.998827	-0.00016	0	2.9989	-6.066E-05	0	0	-5.121E-05	-4.008E-05	0	-1E-05	0			-3.805E-05		-2.26032E-05
46	0.3	2.996511	-0.000174	0	2.9966	-7.636E-05	0	0	-6.627E-05	-4.84E-05	0	-2E-05	0			-5.945E-05		-1.69101E-05
47	0.4	2.99208	-0.000183	0	2.9922	-8.71E-05	0	0	-7.673E-05	-5.299E-05	0	-2E-05	0			-6.669E-05		-2.04108E-05
48	0.5	2.984881	-0.000187	0	2.985	-9.416E-05	0	0	-8.379E-05	-5.508E-05	0	-2E-05	0			-8.436E-05		-9.7936E-06
49	0.6	2.97427	-0.000188	0	2.9744	-9.847E-05	0	0	-8.83E-05	-5.55E-05	0	-3E-05	0			-9.136E-05		-7.10957E-06
50	0.7	2.959646	-0.000186	0	2.9597	-0.0001005	0	0	-9.07E-05	-5.46E-05	0	-3E-05	0			-9.734E-05		-3.1588E-06
51	0.8	2.940423	-0.000182	0	2.9405	-0.0001006	0	0	-9.132E-05	-5.266E-05	0	-3E-05	0			-0.0001003		-3.05383E-07
52	0.9	2.91603	-0.000175	0	2.9161	-9.911E-05	0	0	-9.044E-05	-4.992E-05	0	-3E-05	0			-0.0001005		1.3971E-06
53	1	2.885926	-0.000167	0	2.886	-9.613E-05	0	0	-8.82E-05	-4.647E-05	0	-3E-05	0			-0.0001039		7.74831E-06

Figure 10: re-listing of calculated phi_2 at x=lc from Figure 4 along with empirical measurements data and constraint definitions

A constraint formula calculates the difference between current value and a target value such as '=ARRAYVAL(F44)-P44'. F44 refers to the solution result obtained earlier and P44 refers to the corresponding measured target value. Note that we must use a criterion function such as ARRAYVAL() or PDEVAL() to refer to the solution values; using a constraint formula like =F44-P44 will not work. PDEVAL() is more general and enables us to constrain implicit properties not just explicit values in the output solution array. In this exercise, however, we will stick with ARRAYVAL().

Figure 10a shows our constraint formulas definitions in the range R44:R53 which are easily generated using the auto fill feature of Excel. Our objective is to find the optimal values for the system parameters in order to drive these constraints values to zero. We demonstrate this in the following exercises.

	R
41	
42	**constraints**
43	
44	=ARRAYVAL(F44)-P44
45	=ARRAYVAL(F45)-P45
46	=ARRAYVAL(F46)-P46
47	=ARRAYVAL(F47)-P47
48	=ARRAYVAL(F48)-P48
49	=ARRAYVAL(F49)-P49
50	=ARRAYVAL(F50)-P50
51	=ARRAYVAL(F51)-P51
52	=ARRAYVAL(F52)-P52
53	=ARRAYVAL(F53)-P53

Figure 10a: constraint formulas definitions

+ Exercise 1

In the first exercise, we will investigate the effect of the Mass matrix off diagonal coupling term *mc*. We simply run NLSOLVE() passing in the constraint formulas defined in R44:R53 and *mc* for the design variable as shown in Figure 12a. Since we have one design variable, we can run NLSOLVE as a simple formula in one cell. However, by running NLSOLVE() as an array formula, we obtain additional information from the solver such as how well it was able to minimize the sum of square errors in the constraints and the number of iterations it took.

	R	S

21	=NLSOLVE(R44:R53, mc)	
22		
23		

Figure 12a: NLSOLVE() optimizer formula

We evaluate NLSOLVE() array formula by pressing Ctrl+Shift+Enter. NLSOLVE() spins for a few moments and displays the following solution:

	R	S
21	mc	99993691.37
22	SSERROR	1.73025E-09
23	ITRN	16

Figure 12b: NLSOLVE() optimizer formula result

To view the results graphically, we simply copy the computed value 99993691.37 over current *mc* value in cell E2 as a number. This triggers Excel to automatically re-compute the sheet. The new result is shown in Figure 13 below:

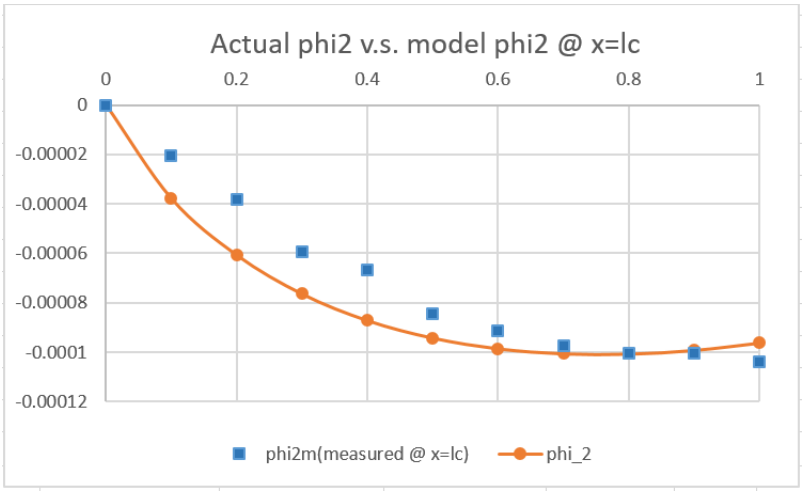

Figure 13: new solution obtained by using the optimal value of *mc* computed in Figure 12

Although, the new solution is an improvement over the original solution in Figure 9, it clearly indicates that *mc* cannot account alone for the discrepancy between the numerical solution and measured values. In the next exercise, we will include *beta* and *gamma* as design parameters.

+ Exercise 2

In this exercise, we will use *beta* and *gamma* as design variables starting from the optimal value of *mc* computed in Exercise 1. We will also show how to enforce additional inequality constraints such as demanding that $beta \geq 0.5$ and $gamma \geq 1$. NLSOLVE() accepts inequality constraints greater than or equal to zero. We define our additional inequality constraint formulas in Range R20:R21, and the new NLSOLVE() array formula in range R22:R25 as shown in Figure 14a.

	R	S
20	=beta-0.5	
21	=gamma-1	
22	=NLSOLVE((R44:R53,R20:R21),(beta,gamma),2)	
23		
24		
25		

Figure 14a: optimization formulas for Exercise 2

Inequality constraints must be ordered at the end among the constraints passed to NLSOLVE() in the first argument. Here we use standard Excel's paranthesis syntax to group the two disjoint ranges R44:R53 and R20:R21 into one union reference, (similarly for beta and gamma). In the 3rd argument, we pass 2 to indicate to NLSOLVE() that the last two constraint formulas are inequalities. NLSOLVE() computes the following result:

	R	S
22	beta	3.622657334
23	gamma	1.000145141
24	SSERROR	3.66343E-10
25	ITRN	22

Figure 14b: result computed by NLSOLVE() array formula of Figure 14a

Copying the obtained values as numbers over the original *beta* and *gamma* values yields the new optimal result shown in Figure 15:

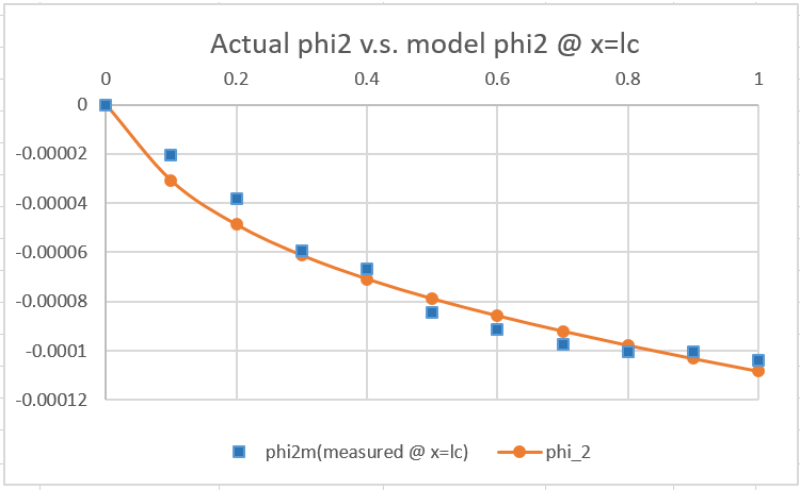

Figure 15: new solution obtained by using the optimal value of *beta* and *gamma* computed in Figure 14b

Please note that because the problem is nonlinear, other possible solutions exist for the combination of our selected design variables. The above exercises are merely intended to demonstrate the steps of carrying out parametric optimization.