

Clement 2

Optimized Best Regression Models

By
Namir Clement Shammass

Contents

Disclaimer	2
Dedication	2
About the Clement 2 Project	3
Introduction	3
Taking the Code for a Quick Spin!.....	4
The Excel File.....	6
The Switches Sheet	8
The Data sheet	11
The OriginalData sheet	12
The Models Sheet	12
Working with Term Math Expressions.....	14
Creating Simpler Regression Models with Simple Cross-Product Terms	19
The ModelsList Sheet	21
The BestResults Sheet.....	22
The Scratch Sheet	22
The Normalization Sheet.....	24
The NormalizedData Sheet	25
The ErrorsReport Sheet.....	25
Using the Excel File.....	25
1. Set Operational Switches	25
2. Enter the Data in the Data Sheet	26
3. Enter the Initial Transformation Ranges and Steps	26
4. Select Normalization Options	26

5. Running the VBA Code.....	27
Notes on the VBA Code in the Data Sheet	29
The Function Fx.....	29
The Subroutine BuildSuperRegressionModels	31
Fatal Runtime Errors	35
Customizing the Transformations with VBA and User-Defined Functions	35
Using Term Math Expressions	36
Epilogue	41
Appendix A – Source of Math-Related Runtime Errors	42
Document History	42

Disclaimer

The Excel VBA code described in this paper comes with no guarantees. The author is not liable for any loss or injury due to the use of the Excel VBA code. Use it at your own risk. I highly recommend that you first test the VBA code with small samples of your own data before you dive in and put the VBA code to serious and full use. You are exclusively and ultimately responsible for interpreting the results of the best regression models generated. You are granted the rights to customize the code (or translate it to other programming languages) to fit your own needs and for your inhouse use. By using the VBA code, you agree with the terms of this disclaimer.

Dedication

To Richard Nelson, the founder of the PPC group for programmable calculator fans. His genius is being the catalyst that built a community of calculator users that lasted over four decades!

About the Clement 2 Project

I started my career as a programming book author in the mid-eighties. My first book, a collaboration with other programmers, presented interesting Turbo Pascal applications. One chapter presented the Clement application (named in honor of my father) that performed sophisticated search for the best regression model by successive improvements. I decided recently to revive the old project and chose to start from scratch. I chose to use Excel and VBA as the tools to build an application that selects the best regression model. I called the programming project Clement 2. At a certain point, I decided to spin off the first program into multiple versions. This study presents the third part of the Clement 2 project. It offers a scheme that uses a simple optimization-based transformation search for the best model. I assume that you are a bit familiar with the first version of the Excel application.

Introduction

This paper presents Clement2, a powerful Excel file and VBA code that allows you to investigate a wide range of **empirical** linearized multiple regression models that fit your data.

The search for the best regression model, in this study, essentially uses a simple *hill-climbing* optimization approach. The baseline application in file *BLMRv1_0.xlsm* searches for the best regression model within a big trust region of transformation powers. By contrast, the application in this study starts with sets of (recommended) three powers per regression variables. The application performs several iterations to slowly move the best transformation powers from an initial set of values to better values that yield higher adjusted R-square values. There is no predefined big trust region, as is the case with *BLMRv1_0.xlsm*. Keep in mind that error in your data may favor certain regression models over others. It's a good idea to apply the VBA code to multiple sets of data and compare the best models.

👉 When this document uses the term *VBA code* it refers to ensemble of global constants and variables, functions, and subroutines that work together to operate the best curve fitting application. The terms *VBA code* and *program* are one and the same.

The VBA code in this study does not explicitly perform the regression calculations. Instead, it obtains the regression ANOVA table from a standard VBA toolkit. The

VBA code prepares the regression data for each model and invokes subroutine ATPVBAEN.XLAM!Regress located in the *Analysis ToolPak – VBA* engine that generates the familiar ANOVA table. This table provides the VBA code with the results it needs.

Taking the Code for a Quick Spin!

Let me guide you through a quick inspection and test for the Excel file using the demo file *Clement2demo.xlsm*. The file contains worksheets I prepared for this demo. Make sure that the *Analysis ToolPak* and the *Analysis ToolPak – VBA* are enabled in your Add-ins dialog box. Perform the following tasks:

1. Load the file *Clement2demo.xlsm*. Activating this demo workbook also brings up the nonmodal Main Menu form shown in Figure 1. Keep that form open while you inspect various worksheets.
2. Click on the *Data* sheet to inspect the name of the variables and their values. Notice the layout of the variable names and the data.
3. Click on the *Models* sheet to see the basic information for the transformations. Notice the variable names in column A and the values in the other columns. The information in this sheet assigns the initial power transformations for each variable. The iterative search process will update these values.
4. Click on the *ModelsList* sheet to view the list of regression models.
5. Right-click the *ModelsList* sheet's tab and select the *View Code* option. This action switches you to the VBE. Click on the *ThisWorkbook* project in the Project window of the VBE. **This is the repertoire of all the VBA code.**
6. Click on the Excel application windows again to select it.
7. Click on the Main Menu form and then click on the topmost button—the one with a number 1 to its left. This choice displays an automatically closing message form that shows the name of the subroutine *DoMultipleRegression* before invoking it. The latter subroutine will ask you to proceed with the process. Click the Yes button. The VBA code will repeat complete cycles of building the regression models list and then testing each set of models. Eventually this repetitive process will stop. During the program execution you will hear messages related to the progress of this process. You will see the *BestResults* sheet. Inspect that sheet with the results. Notice that the values of the R-square and F statistic increase down columns A and B, respectively. Each row in that sheet shows the best model obtained in each iteration cycle.

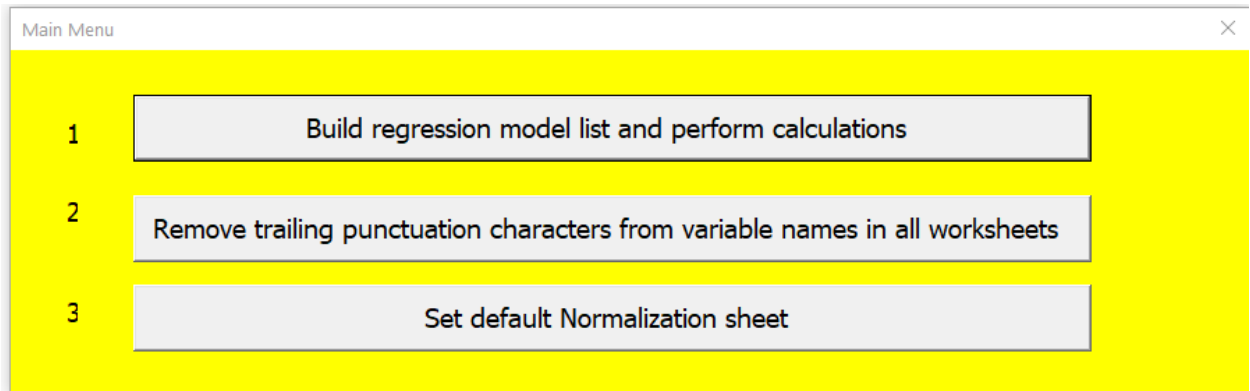







Figure 1. The Menu input box

8. Click on the *Scratch* sheet to view a sample the regression data and results.
9. Once you are done with the demo file, close it. The file *Clement2v1_0.xlsm* is the file you want to work with from now on, so load that file. The demo Excel file can serve a second role as a backup file.

	As a new user to the software, the Main Menu form is your friend and guide.
	Always use backup copies of the Excel file if you plan to <i>tinker</i> and <i>experiment</i> with data and regression models.
	Make sure that the <i>Analysis ToolPak</i> and the <i>Analysis ToolPak – VBA</i> are enabled in your Add-ins dialog box.
	The VBA code checks for many critical errors, displays (and speaks) an advisory message before the program stops. You should inspect the data in the various worksheets to fix the source of the error.

 The Main Menu form is a nonmodal form which you can leave open (and even drag down to just above the task bar level) while inspecting various worksheets and VBA code in the *ThisWorkbook* project. When you run the demo Excel file, the Main Menu form loads automatically. When you load the distribution file *Clement2v1_0.xlsm* you can open the Main Menu form using one of these tasks (which also work for the demo Excel file):

- Clicking on the *Switches* sheet. Activating this sheet also brings up the Main Menu form. If this action does not display the form, click on any other sheet and then click on the *Switches* sheet again.
- Locate subroutine *MiniMenu* (it is the first subroutine after the global declarations of constants and variables) and execute it. This subroutine displays the nonmodal Main Menu form.

The form has three command buttons with numbers placed to their left as shown in Figure 1. I will be referring to the buttons by these numbers. Each command button invokes a subroutine that performs a task related to the regression modeling. By default, when you click on any of these command buttons, the VBA code first displays the name of the subroutine it will invoke in an automatically closing message form (closes after five seconds) and then invokes that subroutine. I have provided this help feature so that you can learn the names of the subroutines you are using.

The Excel File

The *Clement2v1_0.xlsm* file has the following worksheets that play different roles in input, output, and calculations. The next table summarizes the roles of these sheets. The sheets with the yellow background are your main input sheets. The sheet with the green background is your main output worksheet.

<i>Sheet Name</i>	<i>Purpose</i>
Switches	Sheet that contains operational switches that fine-tune how the software works.
Data	Sheet where you place your data to be processed. The values in this sheet are on the proverbial front line ready to be involved in regression calculations.
Models	Sheet containing the range of power patterns that create the regression models' list.
ModelsList	Full list of regression models with each regression term occupying a separate cell on the same row. This list is generated by the VBA code and can be manually edited by the user.
Results	Sheet showing regression results sorted by the adjusted R-square statistic. You can choose to sort the results using the F statistic. This sheet changes with each iteration cycle. When the iterations stop, this sheet shows the most recent results.

<i>Sheet Name</i>	<i>Purpose</i>
BestResults	Sheet showing the best regression results for all the iteration cycles.
Scratch	A scratch sheet used to perform regression calculations.
ErrorsReport	Sheet that lists non-fatal runtime errors that occur during the regression calculations.
OriginalData	Sheet that contains the copy of the data originally appearing in sheet <i>Data</i> . The VBA code backs up (and later restores) the original values of sheet <i>Data</i> to this sheet when normalized data are used.
Normalization	Sheet that lists the regression variables and indicates if data normalization is needed.
NormalizedData	Sheet that contains the last copy of the normalized data. This sheet is available mainly for the inspection of the curious user. The values in this sheet are also the source of data for a subroutine that examines user-selected models in more details.
BigData	This sheet is strictly for the user who wishes to store a large number of variables and data points and analyze a portion of that data, one at a time. The user can then select columns and rows of data to copy to the <i>Data</i> sheet. This sheet is the user's ultimate data storage repertoire. The VBA code does not access this sheet.


The VBA code is comprised of the following operational software components:

- Primary code:
 - The code that builds the list of regression models.
 - The code that performs the regression calculations, writes the results, and sorts the results.
 - The code for data normalization.
- Secondary code:
 - The code that supports VBA functions and user-defined functions used in math expressions.
 - The code that manages the regression variable names.
 - The code that manages math expressions and quasi-variables.
 - The code that manages displaying, hiding, and working with the Main Menu form.

The Switches Sheet

The *Switches* sheet contains over a dozen of switches that allow you to fine-tune how the VBA code works. The sheet, shown in Figure 2, has three columns:

- Column A has the switch names. The names in this column match the name of identifiers in the VBA code. As you become more familiar and comfortable with the VBA code, you can search for the identifiers that match the switch names in column A.
- Column B has the switch values. The switches can be strings, integers, and numeric Boolean flags (0 for False and other integers for True).
- Column C contains a short explanation for what the switches do.

 In the above and next outlines, I use the term *numeric Boolean flag* to mean a logical flag that takes on the value of 0 for False and non-zero (preferably 1) for True. It is easier, quicker, and less error-prone to type 0 or 1 than to type True or False. Beyond the next outline, I will drop the *numeric* pre-qualifier.

The switches are:

- The switch RESULTS_SORT_COLUMN has a single-character string that determines which column in the *Results* sheet is used to sort the results. The default value is “A” which selects the adjusted R-square column. Assigning “B” to the constant selects the F Statistic column for sorting the results. Assigning “C” to the constant selects the Akaike information criteria column for sorting the results.
- The switch SAYIT is a numeric Boolean flag that lets the VBA code speak messages when the value is set to 1 (for True). When the value is set to 0 (for False), the VBA code executes without verbal communications. The default setting is 1.
- The switch WAIT_ON is a Boolean flag that lets you browse at the *Results* sheet, switching back and forth between that sheet and the *Data* sheet. This pause does slow down the program but keeps you in touch with the best regression models. The default setting is 1.
- The switch WAIT_DURATION specifies the number of integer seconds to pause the VBA code while viewing the *Results* sheet. The default setting is 1.
- The switch NORMALIZE_DATA is a numeric Boolean flag that tells the VBA code whether you want to normalize at least one regression variable

(when set to 1) or disable the data normalization feature altogether. Set the value for this switch to 0 (False) if you wish to skip data normalization. The default setting is 0.

- The switch APPENDED_CHAR_TO_VARNAMEs contains the punctuation character that is temporarily appended (and later removed) to the variable names during regression calculations.
- The switch QUASI_VAR_FIRST_CHAR contains the first character(s) in the names of quasi-variables used in math expressions. The switch has the default value of “A”. More about this switch in subsection *Using Term Math Expressions*.
- The switch MAX_ERROR_TO_STOP specifies the number of handled runtime errors that will make the VBA code stop the regression calculations to let you inspect the data and transformations. The default setting is 4.
- The switch SCALE_POWER is set to 100. It is the scaling factor that magnifies the floating-point power values/increments to have integer values.
- The switch MAIN_MENU_HELP is a numeric Boolean flag that tells the VBA code to display the message form (when set to 1) or suppress that form (when set to 0). The default setting is 1.
- The switch ENABLE_MAIN_MENU_FORM is a numeric Boolean flag that tells the VBA code to display the Main Menu form (when set to 1) when you activate the *Switches* sheet. Set the value of this switch to 0 (False) when you wish to prevent the display of the Main Menu form. The default setting is 1. If you set this switch to 0 while the Main Menu form is in view, the VBA code will close that form.
- The switch MAX_RESULTS specifies the maximum number of results to display in the sheet *Results*. The default value is 50.
- The switch BAR_CHAR specifies the character used after a math expression to define the last row covered by that expression.
- The switch REMOVE_TRAILING_PUNCT_CHAR is Boolean flag that, when set to 0, makes the VBA prompt you to delete the trailing punctuation characters. When the switch is set to a 1, the VBA code will go ahead and delete the trailing punctuation characters without asking the user.
- The switch MAX_CYCLES sets the maximum number of cycles/iterations used to search for the best regression model. The default value is 20.
- The switch COMPLETE_CYCLE is a numeric Boolean flag that tells the VBA code whether to stop (when set to zero) or to continue (when set to a


non-zero values) when the adjusted R-square values of the new regression models do not improve. The default value is 0.

- The switch BREAK_BETWEEN_CYCLE is a Boolean flag that tells the VBA code to break (at a Stop statement in the VBA code) between cycles. This break allows you to inspect intermediate results in various worksheets.
- The switch BREAK_EACH_NUM_OF_CYCLES tells the VBA code how many cycles to skip between breaks. The default setting is 1. Increase this number if you want the VBA code to break less frequently.
- The switch BEST_POWER_IN_MID_RANGE tells the VBA code how to select the best mid-range power for the next cycle. The value of 1=best power appears in the middle of next range, otherwise it appears at either end of next range.
- The switch LN_SHIFT_VAL sets the value of the shift in calculating the natural logarithm. **Use a positive value, of say 1, when you anticipate the calculations to divide by the natural logarithm.** The default setting is 0 which assumes no division by a natural logarithm with an argument of 1.

	A	B	C
1	<i>Switch</i>	<i>Value</i>	<i>Comments</i>
2	RESULTS_SORT_COLUMN	A	Name of column in Results sheet used to sort the results
3	SAYIT	0	Turn on verbal messages 0=OFF non-0=ON
4	WAIT_ON	2	lets you view results sheets (slows down the process) 0=OFF non-0=ON
5	WAIT_DURATION	1	Number of seconds to wait
6	NORMALIZE_DATA	0	turn on/off data normalization feature 0=OFF non-0=ON
7	APPENDED_CHAR_TO_VARNames	\$	character appended to variable names
8	QUASI_VAR_FIRST_CHAR	A	first character(s) of the quasi-variable names
9	MAX_ERROR_TO_STOP	4	maximum number of errors that cause the calculations to stop
10	SCALE_POWER	100	Scale factor for powers used to raise variables to
11	MAIN_MENU_HELP	1	Switch to display help form message 0=OFF non-0=ON
12	ENABLE_MAIN_MENU_FORM	1	Switch to ENABLE DISPLAYING Main Menu form 0=OFF non-0=ON
13	BAR_CHAR		Specifies the character used after a math expression to define the last row covered by that expression
14	REMOVE_TRAILING_PUNCT_CHAR	1	Remove trailing punctuation character without offering choice for user when set to non-zero.
15	MAX_CYCLES	20	Maximum cycles for small increment power search
16	COMPLETE_CYCLES	0	Complete all iteration cycles even if R2-adj stops improving. 0=No, non-zero=Yes
17	BREAK_BETWEEN_CYCLE	0	Flag to optionally break between cycles.
18	BREAK_EACH_NUM_OF_CYCLES	1	Number of cycles to break

	A	B	C
19	BEST_POWER_IN_MID_RANGE	0	1=best power appears in the middle of next range, otherwise it appears at either end of next range
20	LN_SHIFT_VAL	0	Yields $\text{LN}(X + \text{LN_SHIFT_VAL})$. Value = 1 useful when dividing by LN transformation in a math expression

Figure 2. The Switches form.

 The Main Menu form has three command buttons with numbers placed to their left as shown in Figure 1. You can disable the message forms that pops up after you click a command button. To do that, set the switch `MAIN_MENU_HELP` to 0 (False). You can also disable displaying the Main Menu form when you activate the *Data* sheet by setting the switch `ENABLE_MAIN_MENU_FORM` to 0 (False). Once you are very comfortable with the VBA code you can even go a step further and run the various subroutines directly from the VBA code listing in the *ThisWorkbook* project in the VBE. I have placed these subroutines right after the declaration of the global variables and constants. Remember that you can locate any function or subroutine from the drop-down list of sorted routine names in the VBA when viewing the project *ThisWorkbook*.

The Data sheet

The *Data* sheet, as the name suggests, is where you enter, paste, and store your *working* data. The first row has the headers for the variable names. The first column is for the data of the dependent variable. The other contiguous columns are for the independent variables. Figure 3 shows a sample *Data* sheet. The names of the variables are Z, T, P, and V. I could have used slightly longer names such as Zfct, Temp, Press, and Vol, respectively. The character case of the variable names is not subject to any rule. However, the name of a variable should NOT contain the punctuation character specified by the switch `APPENDED_CHAR_TO_VARNames` (set by default as the \$ sign). Why? The VBA code removes all occurrences of the punctuation character from the variable names. Thus, for example, if you have variables named Z\$fct, Z\$fct\$, or Zfct\$, the VBA code will rename them all as Zfct. The VBA code will then temporarily append that punctuation character to the same regression variables. In the final stages of the calculations the VBA code will also delete the trailing punctuation characters from the variable names. So, to have peace of mind, just use characters and digits in naming your variables.

	A	B	C	D
1	Z	T	P	V

	A	B	C	D
2	721	11	55	1.5
3	466	4	65	2
4	380	2	67	3
5	662	9	48	4
6	658	8	50	5
7	604	13	34	6
8	563	12	33	7
9	534	1	55	8
10	493	3	41	9
11	480	5	34	10
12	340	10	21	11
13	467	6	29	12
14	439	7	25	13

Figure 3. A sample Data worksheet.

The OriginalData sheet

This sheet has the copy of the original values in sheet *Data* and before any requested transformations are performed. This sheet is distinct from the *BigData* sheet and does not serve the same purpose.

The Models Sheet

This section looks at two ways to work with the sheet *Models* to create regression models. The first subsection presents you the power of using *term math expression*. By that I mean inserting math expression in any term of the regression models. The second subsection presents a simpler approach for creating simple cross-product terms.

The *Models* sheet plays a paramount role in setting up the various regression models **in each iteration cycle**. The scheme for the data transformation has the following rules:

- Column A lists the regression variables that will contribute to the regression models. Each regression variable is raised to a narrow range of initial/current powers defined by values in columns B, C, and D. These powers can be

positive, zero, or negative. The VBA code updates the values in columns B and C for each iteration cycle. The initial values of columns A, B, and C are copied to columns H, I, and J. When a power is zero, the VBA code applies the natural logarithm transformation instead of raising values to zero to always get the value 1—a wasted opportunity. Therefore, the general transformation for a non-zero power i is x^i and for a power of 0 is $\ln(x)$. These transformations represent different types of curvatures! A linear transformation with a power of 1 (i.e., taking the values of a variable as they are) represents zero curvature. Contrast this with taking the squared values of the reciprocal values of a variable and the kind of curvature these transformations offer. Here is a representation that shows how the power values affect the level of concave and convex curvatures. Thus, changing the curvature for a variable is the basic concept behind the default transformation scheme that I present:

Increasing convex curvature _____ Increasing concave curvature
 $-5 < -4 < -3 < -2 < -1 < 1 < 2 < 3 < 4 < 5$

- The non-zero powers can be integers or non-integers. If your data has negative values or high-valued positive values, I strongly suggest that you normalize the related variables to fall in the range (1, 2). The VBA code supports data normalization by applying $x = (x - x_{\min}) / (x_{\max} - x_{\min}) + 1$. For data with positive values that vary wildly, you can also request the VBA code to first calculate the natural logarithms of the variables and then proceed with the normalization in the range (1, 2). This approach basically normalizes the magnitude of the data points. By normalizing data into the range (1, 2) you can apply positive or negative powers (both as integers or as floating-point values) and have your mind rest at ease!
- The iterative scheme of transformation uses a relatively narrow range of powers with a step increase in power. For each variable, you specify three power values in a narrow range. The first value (defined by *From Power*) is for the lower power. The middle value (calculated using the values in the other columns) is the mid-range power. The third value (defined by *To Power*) is for the upper power. This scheme represents a simple optimization method. The VBA code determines, for each regression variable, which of the three power values (the lower, median, or upper power) participate in yielding the best regression model in each iteration cycle. The best powers are set as the new mid-range or lower/upper values for the next iteration cycle.

Working with Term Math Expressions

This subsection presents the more advanced tool to create advanced transformations using math expressions for the various terms of a regression model. Figure 4 shows a sample *Models* sheet.

	A	B	C	D	E
1	Variable	From Power	To Power	Power Step	TME
2	Z	0.9	1.1	0.1	
3	T	0.9	1.1	0.1	
4	P	0.9	1.1	0.1	
5	V	0.9	1.1	0.1	
6	T	0.9	1.1	0.1	'=A6*A7
7	P	0.9	1.1	0.1	
8	T	0.9	1.1	0.1	'=A8*A9*A10
9	P	0.9	1.1	0.1	
10	V	0.9	1.1	0.1	

Figure 4. A sample *Models* worksheet that uses math expressions.

The *Models* sheet has the following columns:

- Column A is labeled *Variable* and contains the names of the variables that appear in a term (either all by itself or in math expressions). The regression variables in column A must also appear in the first row in sheet *Data*. Unlike the sheet *Data*, the regression variables can appear on multiple rows in column A of the sheet *Models*. The cell A2 has the name of the dependent variable. The cells below it list the names of independent variables and can even have the name of the dependent variable (to create a Pade-style empirical fit). You also need to observe the following rules in naming variables:
 - The variable names in the *Models* sheet must match those in the *Data* sheet. If you spell the variable names differently in the *Models* sheet, the VBA code will overwrite that spelling with the one appearing in the *Data* sheet.
 - All the named variables in column A must also appear in the first row of the *Data* worksheet (but the reverse is not mandatory).

- The regression variables in column A can appear in multiple rows.
- Column B is labeled *From Power*. This column contains the initial lower range power value for each variable. Values should not have more than 2 decimal places. The values in this column need not be the same for the same regression variables that appear in different rows of the worksheet.
- Column C is labeled *To Power*. This column contains the initial upper range power value for each variable. Values should not have more than 2 decimal places. The values in this column need not be the same for the same regression variables that appear in different rows of the worksheet.
- Column D is labeled *Power Step*. This column contains the power increment value for each variable. Values should not have more than 2 decimal places. The values in this column need not be the same for the same regression variables that appear in different rows of the worksheet. The values in column D remain fixed during all the iteration cycles.
- Column E is labeled *TME (short for Term Math Expression)*. This column tells the VBA whether you want to involve the subsequent variable(s) in a math expression. Leave the cells in this column empty if the corresponding variables appear in a term all by itself OR if it is in the range of regression variables already covered by a math expression that appears in a previous row. Otherwise, you can enter a mathematical expression that involves the variable in the same row and one or more variables in subsequent rows. The math expressions require you to observe the following syntax and rules:
 - The first character is the single quote which tells Excel not to process the remaining text.
 - The equal sign.
 - A *special* math expression using operators, parentheses, functions, and quasi-variables. These quasi-variables have names that use the format A_n where n is the row number that refers to each regression variable in column A. Think of quasi-variables as transformed regression variables and also as unique pointers to regression variables in column A. If you enter a lowercase A for a quasi-variable name, the VBA code will convert it to uppercase. While the names of the regression variables in column A can be repeated, the names of the quasi-variables appearing in a math expression **must be unique and must not appear in another math expression in the column TME**. The VBA code will replace the quasi-variables A_n with the names of the corresponding

regression variables and the applied transformations. The VBA code needs to know the range of n in the quasi-variable names An that appear in a math expression. The row in which the math expression appears also specifies the minimum value of n in the quasi-variable names An . To let VBA know about the last row covered by the regression variables included in the math expression, you have two choices:

- Use the rightmost reference to a quasi-variable An to refer to the last sought row by the value of n . Thus, for example, the expression $'=A10*(A9+A8)/A11$ tells the VBA code that the last row to include a regression variable is row 11. The expression should appear in row 8, so VBA knows the range of rows covered by the math expression.
 - Append a vertical bar character |, defined by switch BAR_CHAR, followed by the number of the last sought row. Thus, for example, the expression $'=A10*(A11+A8)/A9|11$ tells the VBA code that the last row to include a regression variable is row 11. The expression should appear in row 8, so VBA knows the range of rows covered by the math expression.
- The digits n for the quasi-variables An can appear in any order BUT **must appear only once**. For example, the expression $'=A8*(A8+A9)/A10$ is incorrect because the quasi-variable $A8$ appears more than once in the expression.
 - The TME cells below a math expression, and whose rows are referenced by the quasi-variables, must be empty.
 - The math expression can use operators, parentheses, and even function names. There is no restriction on the character-case for function names. The functions and explicit operators used in a math expression are *fixed transformations*. This means that they appear in every single term generated by the VBA code for that math expression and for different transformations of the regression variables.
 - If you raise one quasi-variable to the power of another quasi-variable, make sure you enclose both quasi-variables in pairs of parentheses, to ensure proper precedence evaluation by VBA. I also highly recommend that you normalize both variables referred to by the quasi-variables so the results of raising to powers don't go through the

proverbial roof! For example the expression $'=(A8)^{(A9)}$ will correctly preserve the proper order of evaluation done by function *Evaluate* in function *Fx*. However, the expression $'=A8^{A9}$ may produce the wrong results when A8 and A9 refer to regression variables that are themselves raise to other powers. The VBA code may wrongly evaluate, as an example, the string “ $1.2^2^{1.8^3}$ ” and not the intended string “ $(1.2^2)^{(1.8^3)}$ ”.

- The VBA code sets the contents of the TME cells in the second sheet row (corresponding to the entry for the dependent variable) to an empty string as its content is logically irrelevant.
- The VBA code translates the quasi-variables into the names of the associated regression variables. The code then appends a punctuation character to each variable name. The function *Fx* then translates the variable names (with their trailing punctuation characters) into string images of numbers. The final stage is where function *Fx* calls the VBA function *Evaluate* to pass a string with numbers, operators, and function to the Excel interpreter and obtain a numeric value. Thus, the process of going from the quasi-variables to the numeric result has multiple layers before the Excel interpreter does its job.

👉 In this study I list the leading single-quote character used in the math expressions. You **do need** to key in that character to start entering a math expression with quasi-variables in any Excel cell. The second character in a math expression must be the equal sign. Keep in mind that Excel will hide that leading single-quote character in its worksheets. Excel will display the single-quote character in the text edit box when you select a cell that contains a math expression. The single-quote character for math expression appears in all the examples of the math expressions as a reminder. The sheets *ModelsList* and *Results* do not include the leading single-quote characters and the equal signs when displaying the math expressions containing the names of the actual regression variables.

👉 Since you can enter all kinds of text in the TME cells, you are highly responsible to make sure that your input will translate into valid math expressions. The VBA code will catch any runtime error generated by a faulty evaluation of a math expression. It is possible to create math expressions with *logical errors* that yield values, albeit the wrong ones. This logical error affects

the quality of the regression results. So, if the results look weird to you, check the math expressions carefully for logical errors.

In Figure 4 you see a sample input with the following data:

- The second row declares the variable Z to have initial transformations in the range of 0.9 to 1.1 in steps of 0.1. These values will change during the different iteration cycles. The TME cell is empty to signal that the dependent term is separate. In fact, the VBA code will enforce this rule.
- The third, fourth, and fifth rows declare the names of their variables to be T, P, and V, respectively. Each variable has the transformations in the initial range from 0.9 to 1.1 in steps of 0.1. These values will change during the different iteration cycles. The TME cells for these variables are empty to indicate that they occupy separate terms in the regression model.
- Rows 6 defines a simple cross-product for the variables T and P. The range of 0.9 to 1.1 in steps of 0.1. These values will change during the different iteration cycles. The TME cell E6 contains the math expression '=A6*A7' to state that you want multiplicative cross-products of various powers of variables T and P (named in cells A6 and A7). Notice that cell E7 is empty.
- Rows 8 defines multiplicative double cross-products for the variables named T, P, and V. The range of initial transformations are 0.9 to 1.1 in steps of 0.1. These values will change during the different iteration cycles. The TME cell E8 contains the math expression '=A8*A9*A10' to define the cross-product of the transformations of variables T, P, and V (named in cells A8, A9, and A10). Notice that TME cells E9 and E10 are empty.

The above information indicates that we seek an empirical model of the general form:

$$f_0(Z) = a_0 + a_1 \cdot f_1(T) + a_2 \cdot f_2(P) + a_3 \cdot f_3(V) + a_4 \cdot f_{41}(T) \cdot f_{42}(P) + a_5 \cdot f_{51}(T) \cdot f_{52}(P) \cdot f_{53}(V) \quad (4)$$

The power of the information in the *Model* sheet is that you can apply different transformation ranges to the same variable appearing in different terms, cross-products, and math expressions.

👉 It does not take many transformations for each variable/term to generate over tens of thousands of models! That is why I highly recommend making the range of

powers for each regression variable cover three points. If you have a few terms in your regression model you can use ranges of five points. I also recommend you run the VBA code on a separate and dedicated computer. Enabling the VBA code to speak messages is very practical when you are running the VBA code on one machine while you are working on another computer. The verbal messages include the percent progress in the calculations.

Creating Simpler Regression Models with Simple Cross-Product Terms

This subsection presents a scheme to create regression include simpler versions of the cross-product terms. Figure 5 shows a sample *Models* sheet with such information.

	A	B	C	D	E
1	Variable	From Power	To Power	Power Step	TME
2	Z	0.9	1.1	0.1	
3	T	0.9	1.1	0.1	
4	P	0.9	1.1	0.1	
5	V	0.9	1.1	0.1	
6	T	0.9	1.1	0.1	*
7	P	0.9	1.1	0.1	
8	T	0.9	1.1	0.1	*
9	P	0.9	1.1	0.1	*
10	V	0.9	1.1	0.1	

Figure 5. a sample Models worksheet with no math expressions.

Columns A through D in Figure 5 match those in Figure 4 and serve the same purpose. The TME column E serves a similar and simpler purpose, albeit using a different syntax. This column tells the VBA whether you want to involve the subsequent variable in a cross-product term. Leave the cells in this column empty if the corresponding variable appears in a term all by itself. Otherwise, you can enter * to declare that the variable and the one in the subsequent row are cross-products. You can have more than two variables declared in cross-products. The TME columns in Figures 4 and 5 are equivalent and create the same cross-product terms. You can also enter / to indicate that you want to calculate the ratio of the transformed variables. You can even enter + and –, although using these operators makes more

sense in math expressions. Note that the TME cell of the last variable in a cross-product term must be empty. This empty cell tells the VBA code that the variable in the next row will be in the next term of the regression model. The VBA code sets the contents of the TME cells in the second sheet row (corresponding to the entry for the dependent variable) to an empty string as its contents are logically irrelevant. You must make sure that the TME cell of the last row, declaring variables, is empty.

If you just use the power-based transformation scheme that I discussed above, then you may need to only tweak the values of the switches. If you plan to use custom transformation functions, then you need to do more in editing the VBA code. More about this later.

As each iteration cycle changes the values in the columns *From Power* and *To Power*, the VBA code generates a new list of regression model based on the updated power values. The code then calculates the regression model statistics for the new set of models. The iterations stop either because they reach the maximum number of iterations, or the calculations stopped improving the adjusted R-square values.

You can even let the name of the dependent variable appear with the cross-product terms to fit the best Pade-type rational model variant:

$$f_0(y) = a_0 + a_1 \cdot f_1(x_1) + a_2 \cdot f_2(x_2) + a_3 \cdot f_3(x_3) + \dots \\ - a_n \cdot f_{n1}(x_1) \cdot f_{n2}(y) - a_{n+1} \cdot f_{n+1,1}(x_2) \cdot f_{n+1,2}(y) - \dots \quad (5)$$

In Figure 5 you see a sample input with the following data:

- The second row declares the variable Z to have transformations in the initial range from 0.9 to 1.1 in steps of 0.1. These values will change during the different iteration cycles. The TME cell is empty to signal that the dependent term is separate. In fact, the VBA code will enforce this setting.
- The third, fourth, and fifth rows declare the names of their variables to be T, P, and V, respectively. Each variable has the transformations in the initial range from 0.9 to 1.1 in steps of 0.1. These values will change during the different iteration cycles. The TME cells for these variables are empty to indicate that they occupy separate terms in the regression model.
- Rows 6 and 7 declare the variables named T and P. The range from 0.9 to 1.1 in steps of 0.1. These values will change during the different iterations.

The cell E6 has a * character to indicate that the corresponding term has the cross product of the transformations of T and P. Notice that cell E7 is empty.

- Rows 8, 9 and 10 declare the variables named T, P, and V. Each variable has the transformations in the initial range from 0.9 to 1.1 in steps of 0.1. These values will change during the different iteration cycles. The cells E8 and E9 have a * character to indicate that the corresponding terms has the cross-product of the transformations of T, P, and V. Notice that cell E10 is empty.

The ModelsList Sheet

The *ModelsList* sheet shows the list of all the models that will be tested. The VBA code uses the data in sheet *Models* to create the big list. Figure 6 shows a partial view of the sheet. Notice that the worksheet has no header row:

	A	B	C	D	E
1	$Z^{(-0.2)}$	$T^{(-0.2)}$	$P^{4.7}$	$V^{(-1.4)}$	T^*P^*V
2	$Z^{(-0.1)}$	$T^{(-0.2)}$	$P^{4.7}$	$V^{(-1.4)}$	T^*P^*V
3	$LN(Z+1)$	$T^{(-0.2)}$	$P^{4.7}$	$V^{(-1.4)}$	T^*P^*V
4	$Z^{(-0.2)}$	$T^{(-0.1)}$	$P^{4.7}$	$V^{(-1.4)}$	T^*P^*V
5	$Z^{(-0.1)}$	$T^{(-0.1)}$	$P^{4.7}$	$V^{(-1.4)}$	T^*P^*V
6	$LN(Z+1)$	$T^{(-0.1)}$	$P^{4.7}$	$V^{(-1.4)}$	T^*P^*V
7	$Z^{(-0.2)}$	$LN(T+1)$	$P^{4.7}$	$V^{(-1.4)}$	T^*P^*V
8	$Z^{(-0.1)}$	$LN(T+1)$	$P^{4.7}$	$V^{(-1.4)}$	T^*P^*V
9	$LN(Z+1)$	$LN(T+1)$	$P^{4.7}$	$V^{(-1.4)}$	T^*P^*V
10	$Z^{(-0.2)}$	$T^{(-0.2)}$	$P^{4.8}$	$V^{(-1.4)}$	T^*P^*V
11	$Z^{(-0.1)}$	$T^{(-0.2)}$	$P^{4.8}$	$V^{(-1.4)}$	T^*P^*V
12	$LN(Z+1)$	$T^{(-0.2)}$	$P^{4.8}$	$V^{(-1.4)}$	T^*P^*V
13	$Z^{(-0.2)}$	$T^{(-0.1)}$	$P^{4.8}$	$V^{(-1.4)}$	T^*P^*V
14	$Z^{(-0.1)}$	$T^{(-0.1)}$	$P^{4.8}$	$V^{(-1.4)}$	T^*P^*V
15	$LN(Z+1)$	$T^{(-0.1)}$	$P^{4.8}$	$V^{(-1.4)}$	T^*P^*V
16	$Z^{(-0.2)}$	$LN(T+1)$	$P^{4.8}$	$V^{(-1.4)}$	T^*P^*V
17	$Z^{(-0.1)}$	$LN(T+1)$	$P^{4.8}$	$V^{(-1.4)}$	T^*P^*V
18	$LN(Z+1)$	$LN(T+1)$	$P^{4.8}$	$V^{(-1.4)}$	T^*P^*V
19	$Z^{(-0.2)}$	$T^{(-0.2)}$	$P^{4.9}$	$V^{(-1.4)}$	T^*P^*V
20	$Z^{(-0.1)}$	$T^{(-0.2)}$	$P^{4.9}$	$V^{(-1.4)}$	T^*P^*V
21	$LN(Z+1)$	$T^{(-0.2)}$	$P^{4.9}$	$V^{(-1.4)}$	T^*P^*V

Figure 6. A partial view of the *ModelsList* worksheet.

Each row in sheet *ModelsList* represents a regression model. The individual cells of that row show the various terms of the regression model. Notice that the negative powers are enclosed in parentheses to support a proper order of evaluation, especially in cross-product terms. The first column shows the dependent variable and its transformations. The second column and on show the various terms for the

independent variables and their transformations. Some terms show single variables while others show cross-product terms.

The BestResults Sheet

The *BestResults* sheet displays a sorted list of the best regression models, each obtained in a different iteration cycle. The results include the values for the adjusted R-square, the F statistic, the transformations for each term of the regression model, and the regression coefficients. The *Results* sheet has many columns, as shown in Figure 7.

Rsqr Adj	F Stat	Transf of Y	Transf of X1	Transf of X2	Transf of X3	Transf of X4	Coeff0	Coeff1	Coeff2	Coeff3	Coeff4
0.67600059	7.259276115	ZS^0.9	TS^1.1	PS^1.1	VS^0.9	TS^PS^VS	2.168284669	-1.17405446	-	-1.538427724	1.329997389
0.70141557	8.047409362	ZS^0.7	TS^1.1	PS^1.3	VS^0.7	TS^PS^VS	3.226414658	1.058329636	-0.50306872	-1.985366072	1.034593673
0.737347033	9.42191554	ZS^0.5	TS	PS^1.5	VS^0.5	TS^PS^VS	3.988252115	0.958216952	-	-2.411068603	0.732647138
0.785686863	11.99820862	ZS^0.3	TS^0.8	PS^1.7	VS^0.3	TS^PS^VS	4.544686618	0.839990415	0.371890901	-2.831858268	0.433810773
0.837337542	16.44310018	ZS^0.1	TS^0.6	PS^1.9	VS^0.1	TS^PS^VS	4.537790475	0.415430163	0.216965227	-3.144129297	0.140225591
0.88522777	24.13872704	ZS^(-0.1)	TS^0.4	PS^2.1	VS^(-0.1)	TS^PS^VS	3.676964271	0.664854529	0.054946194	-3.330399243	-0.133656681
0.923610083	37.27219855	ZS^(-0.3)	TS^0.2	PS^2.3	VS^(-0.3)	TS^PS^VS	0.308589045	4.112399657	0.133829072	-3.396597028	-0.376432354
0.949824729	57.79040983	ZS^(-0.5)	TS^0.1	PS^2.5	VS^(-0.5)	TS^PS^VS	-8.846254999	13.26792698	0.173713284	-3.349851029	-0.586180757
0.964727893	83.05304304	ZS^(-0.7)	TS^(-0.1)	PS^2.7	VS^(-0.7)	TS^PS^VS	22.76603162	-	0.188602473	-3.26287722	-0.7573049
0.973323046	110.4566156	ZS^(-0.7)	TS^(-0.1)	PS^2.9	VS^(-0.9)	TS^PS^VS	21.5613424	18.38829549	0.150166737	-2.576553105	-0.751505999
0.975897938	122.4706769	ZS^(-0.6)	TS^(-0.1)	PS^3.1	VS^(-1)	TS^PS^VS	18.18459264	17.77025521	0.105791475	-2.006315738	-0.645243382
0.978082256	134.875402	ZS^(-0.5)	TS^(-0.1)	PS^3.3	VS^(-1)	TS^PS^VS	15.00546998	14.93485793	0.073510622	-1.643821873	-0.530292636
0.979871173	147.0399829	ZS^(-0.4)	TS^(-0.1)	PS^3.5	VS^(-1)	TS^PS^VS	11.9954477	12.13905854	0.049347049	-1.297711563	-0.419919382
0.981577611	160.8453325	ZS^(-0.3)	TS^(-0.1)	PS^3.7	VS^(-1.1)	TS^PS^VS	9.152843155	9.505419665	0.031087138	-0.904720723	-0.319052664
0.983207773	176.6540827	ZS^(-0.2)	TS^(-0.1)	PS^3.9	VS^(-1.1)	TS^PS^VS	6.370063607	7.079724958	0.017602899	-0.599662574	-0.211898383
0.98457781	192.5249036	ZS^(-0.1)	TS^(-0.1)	PS^4.1	VS^(-1.1)	TS^PS^VS	3.659675533	4.652409013	0.0075099	-0.298941773	-0.105826446
0.985688765	207.625111	ZS^(-0.1)	TS^(-0.1)	PS^4.3	VS^(-1.1)	TS^PS^VS	3.565743903	2.299012235	0.00623213	-0.288990283	-0.102818863
0.986510585	220.3966035	ZS^(-0.1)	TS^(-0.1)	PS^4.5	VS^(-1.1)	TS^PS^VS	3.480124996	2.214296923	0.005187617	-0.279867586	-0.100106521
0.987282154	233.8890028	ZS^(-0.1)	TS^(-0.1)	PS^4.7	VS^(-1.2)	TS^PS^VS	3.417214174	2.137158093	0.004315037	-0.257143335	-0.099575199
0.98825162	253.3543643	LN(Z+1)	T^(-0.1)	P^4.9	V^(-1.2)	T^P^V	-13.09952807	11.88173039	-	1.510418396	0.575495602

Figure 7. A partial view of the Results worksheet

The Scratch Sheet

This sheet is used by the VBA code to perform multiple regression and copy selected results to the *Results* sheet. The sheet shows the familiar regression ANOVA table and the columns of the transformed variables. The regression ANOVA table and the data columns always start in columns A and K, respectively. Column J is always empty, separating the regression ANOVA table from the data columns.

SUMMARY OUTPUT				
Regression Statistics				
Multiple R 0.990326899				
R Square 0.980747367				
Adjusted R Square 0.971121051				
Standard Error 0.020446718				
Observations 13				
ANOVA				
	df	SS	MS	F
Regression	4	0.170374361	0.04259359	101.8818958
Residual	8	0.003344546	0.000418068	6.76378E-07
Total	12	0.173718907		

	Coefficients Upper 95.0%	Standard Error	t Stat	P-value	Lower 95%	Upper 95%	Lower	95.0%
Intercept	0.057945509 -0.161550927	0.095184753 0.277441944	0.608768804		0.559562711	-0.161550927	0.277441944	
LN(T+1)	-1.943964816 -2.293713322	0.151668638 -1.594216309	-12.81718382		1.29571E-06	-2.293713322	-1.594216309	
P^4.9	-0.020509647 -0.025813384	0.002299969 -0.01520591	-8.917359844		1.98305E-05	-0.025813384	-0.01520591	
V^(-1.2)	1.484596686 1.255043201	0.099545999 1.714150171	14.91367512		4.02974E-07	1.255043201	1.714150171	
T*P*V	0.592395975 0.472408716	0.052032543 0.712383234	11.38510524		3.19772E-06	0.472408716	0.712383234	
RESIDUAL OUTPUT								
Observation	Predicted LN(Z+1)		Residuals					
1	1.098056348		0.00055594					
2	0.854154993		-0.007982625					
3	0.751204463		-0.006895231					
4	1.034036872		0.011576935					
5	1.061936603		-0.020019658					
6	0.965954173		0.024669478					
7	0.958264187		-0.008421921					
8	0.881527375		0.038431163					
9	0.885655744		-0.009531054					
10	0.867475161		-0.005660017					
11	0.684757598		0.008389583					
12	0.863922908		-0.016625047					
13	0.823782675		-0.008487546					

Figure 8a. The ANOVA table in Scratch worksheet

LN(Z+1)	LN(T+1)	P^4.9	V^(-1.2)	T*P*V
1.098612289	1.041453875	15.05315595	1	3.188405797
0.846172368	0.810930216	26.80864853	0.950210693	2.551984877
0.744309233	0.733969175	29.85705573	0.863187983	2.449275362
1.045613807	0.980829253	9.611001374	0.789739328	3.219911783
1.041916945	0.949080555	10.97202568	0.726989097	3.367202268
0.990623651	1.098612289	3.385801412	0.672811536	3.56899811
0.949842266	1.070441412	3.113750378	0.625602827	3.572463768
0.919958538	0.693147181	15.05315595	0.584131198	2.722117202
0.87612469	0.773189888	5.864782477	0.547435664	2.765595463
0.861815145	0.84729786	3.385801412	0.514755995	2.974165091
0.693147181	1.011600912	1	0.485483255	3.195652174
0.84729786	0.88238918	2.193896058	0.459124207	3.18147448
0.815295129	0.916290732	1.504664401	0.435275282	3.260869565

Figure 8b. The data columns in Scratch worksheet

The Normalization Sheet

The switch `NORMALIZE_DATA` acts as a main switch and tells the VBA code whether to normalize at least one regression variable or skip the data normalization feature all together. If the switch `NORMALIZE_DATA` is set to `False`, the VBA code will ignore the contents of the *Normalization* sheet.

The VBA code first copies the *Data* sheet into the *OriginalData* sheet and then performs the transformations. The *Normalization* sheet allows you to specify if you want to transform any regression variable and also if you want to first apply the natural logarithm transformation to that variable. Figure 9 shows a sample *Normalization* sheet.

	A	B	C	D	E
1	Z	T	P	V	
2	N	N	N	N	Normalize?
3	N	N	N	N	Take the Log values First?
4					Minimum
5					Maximum

Figure 9. The Normalization sheet with sample data.

The Normalization sheet has the following rows and columns:

- The first row lists the regression variables. These variable names must match those in the *Data* sheet. The VBA code makes sure that the names of the variables in the first row match those in the sheet *Data*.
- The second row has the single-character switches that state if a variable is to be normalized. The VBA code regards any cell that is neither `Y` nor `y` as a request not to normalize values and vice versa.
- The third row has the switches that state if the variable is to be first transformed into its natural logarithm values. The VBA code regards any cell that is neither `Y` nor `y` as a request not to transform and vice versa. The values in this row are ignored if the corresponding value in row 2 does not request normalization. The VBA code will check if a normalized variable has non-positive values. If it does, the VBA code will set the normalization switch to `N` for that variable and will skip taking its natural logarithm. You will receive

a notification to that effect and be prompted if you wanted to stop the calculation process altogether to *cure* your data, change normalization switches, and then try again.

- The fourth and fifth rows show the calculated minimum and maximum values used in the data normalization.
- The column that appears right after the last variable name has rows that describe the meaning of rows 2 to 4.

The NormalizedData Sheet

This sheet will contain the normalized values of your data if you apply data normalization. The values in this sheet are mainly for the curious user's inspection. The values are also used by sub *ExamineARegressionModel* when the switch NORMALIZE_DATA is set to 1 (True) to quickly retrieve normalized data.

The ErrorsReport Sheet

The *ErrorsReport* sheet store the regression runtime errors. This information includes the offending model number and the error message. The VBA code makes a verbal warning (when enabled) when a runtime error occurs and logs in that error in the *ErrorsReport* sheet. The VBA code allows a specified maximum number of errors before it ends program execution. This scheme prevents the error warnings from getting out of hand and becoming annoying, since it is very possible that the numbers of regression models are in the hundreds, thousands, or even tens of thousands! Besides, I do not want the reader to get mad with her/his computer, break it, and then bill me for a new one!

Using the Excel File

This section looks at the steps involved in using the Excel file and the VBA code to perform empirical curve fitting. The steps involved are outlined in the next subsections. I would like to point out that I have enabled verbal messages from the code to confirm the prompt for action, completion of certain tasks, and report error messages. The verbal messaging offers valuable non-visual communications that allows you to run the VBA code on a separate dedicated machine (while you work on your main computer) and listen to the messages that, for example, report on the progress of the calculations.

1. Set Operational Switches

You may need to change the operational switches listed in sheet Switches (and shown in Figure 2) to influence how the VBA code works. For example, if you intend

to normalize one or more regression variables, then you set switch `NORMALIZE_DATA` to 1.


2. Enter the Data in the Data Sheet

Select the *Data* sheet and clear it to start entering new data or pasting it from another source. Keep in mind the following rules:

1. The first column must store the data for the dependent variable.
2. The second columns and on store data for various independent variables.
3. All columns must have the same number of rows. You must deal with missing data by doing one of the following:
 - a. Delete the row that has missing data (especially if several columns have missing data in that row).
 - b. Estimate the missing data by averaging or any other suitable method.

3. Enter the Initial Transformation Ranges and Steps

The *Models* sheet provide you with a powerful framework to generate hundreds and thousands of regression models that will appear in the *ModelsList* sheet. The *Models* sheet allows you to specify the initial and narrow range of transformations and their increments. The range for each regression variable will change with each iteration. Select the *Models* sheet and populate columns A to E with data that select the variable(s) for each term, specify the initial narrow range and small increment in the transformational powers, and optionally enter math expressions or specify if two or more rows make up a single cross-product term. Your *Models* sheet should resemble the one in Figure 4 or in Figure 5.

 If you want to use VBA functions and/or user-defined functions (that you declare in a module) then you need to follow the instructions in subsections *Customizing the Transformations with VBA Functions and User-Defined Functions*.

4. Select Normalization Options

To enable the data normalization feature, you must first set the switch (in sheet *Switches*) `NORMALIZE_DATA` to 1. Otherwise, the VBA code bypasses subroutines that perform data normalization related tasks. After setting the switch to 1 (True), I suggest that you click button 7 in the Main Menu form to execute subroutine *SetDefaultNormalization*. This subroutine setups the default *Normalization* sheet. This setup lists the variable names and the right-side tags. The cells for the normalization switches are all set to N (short for no). Select the *Normalization* sheet to choose which variable you need to normalize. Set the cell in row 2 under the selected variable name(s) to y or Y to normalize. Set the cell in row

3 under the selected variable name(s) to y or Y if you want the VBA code to first calculate their natural logarithms before normalizing their values.

The VBA code copies the values in the Data sheet into the *OriginalData* sheet before performing the normalization and regression calculations. The VBA code restores the original values of sheet *Data*:

- After the regression calculations end without a runtime error.
- After the maximum number of runtime errors have been reached.

As a failsafe, you can invoke subroutine *RestoreData* to restore the original values in sheet *Data*.

5. Running the VBA Code

Before I discuss running the VBA code you need to make sure that the options for the Add-ins *Analysis Toolpak* and *Analysis Toolpak – VBA* are checked in the Add-ins dialog box, as shown in Figure 10.

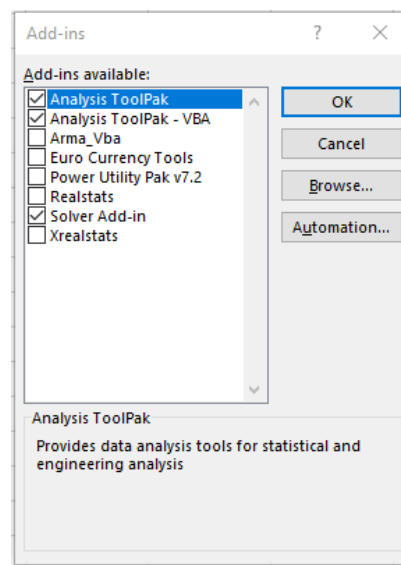


Figure 10. The Add-ins dialog box.

You can use the command buttons in the Main Menu form (see Figure 1) to invoke the regression calculations as a single task (using button 1). You can also run the subroutine *DoMultipleRegression* directly. Select any sheet and choose to view the VBA code in the *ThisWorkbook* part. The code uses *DoEvents* to prevent Excel from shutting you out and allows you to pause or stop the VBA code if need be, by clicking on the pause or stop icons in the Debug toolbar of the VBE. I recommend

that you have the VBA code of the *Data* sheet remain in view to give you full control (an easier task if you have two screens). Figure 11 is a table that gives you a heads-up summary of the subroutines, in the *ThisWorkbook* part, you can use to perform the empirical regression models calculations. The subroutines are grouped by colors. Your *go to* subroutine is *MiniMenu*. The middle column of that figure shows the command button numbers, in Main Menu form, associated with the subroutines.

<i>Subroutine Name</i>	<i>Button Number</i>	<i>Purpose</i>
MiniMenu		Displays a multiline input box that allows you to easily invoke the rest of the subroutines listed in this table.
DoMultipleRegression	1	The main “one-stop shop” subroutine to build the models list and run the regression calculations.
SetDefaultNormalization	3	Setup the default <i>Normalization</i> sheet with all normalization switches off.
restoreVarNames	2	Remove trailing punctuation characters from all the variable names in all the sheets.

Figure 11. The list of the relevant subroutines you will be using.

You can build the list of regression models and then perform the regression calculations in one swoop. Click on sheet *Switches* to activate it (or Run the *MiniMenu* subroutine) to launch the Main Menu form. Click on button 1 to invoke the subroutine *DoMultipleRegression*. This subroutine requests your confirmation that you have prepared your data and regression model information. The message also announces the number of regression models that will be tested. If that number overwhelms you, then click the Cancel button and edit the values in the columns To Power, From Power, and Power Step, in the *Models* sheet, to test fewer models.

👉 The simplest way to use the VBA code to perform regression model selection is to:

1. Tweak the switches in worksheet *Switches*.
2. Select your initial transformation ranges.

3. Click on button 1 of the Main Menu form to execute subroutine *DoMultipleRegression*.

Notes on the VBA Code in the Data Sheet

The Function Fx

The code for function *Fx* in the *ThisWorkbook* project transforms the variables in a regression term from a string of text to a string images of numeric values (with possible operators and functions) and then evaluates that string to yield a resulting floating-point number:

```
Function Fx(ByVal sExpress As String, ByRef sVarNames() As String, ByRef
nVarIdx() As Integer, ByRef X() As Double) As Double
    Dim I As Integer, I1 As Integer, I2 As Integer, J As Integer
    Dim NumFx As Integer, countFX As Integer
    Dim msg As String, sFx() As String
    Dim bFoundMyFx As Boolean

    On Error GoTo HandleErr


    sExpress = Replace(sExpress, " ", "")

    ' loop using array nVarIdx to replace correctly replace variable names with
    theirvalues
    For I = LBound(X) To UBound(X)
        J = nVarIdx(I)
        sExpress = Replace(sExpress, sVarNames(J), GetValImage(X(J)))
    Next I
    Fx = Evaluate(sExpress)
ExitProc:
    Exit Function

HandleErr:
    MsgBox "The variable sExpress contains " & sExpress, vbOKOnly + vbInformation,
    "Information"
    FinalMessage "Fatal error in function Fx. The program execution will stop.
    Error is " & Err.Description
End Function
```

The parameter *sExpress* has the string that contains a regression model terms that use the names of the regression variables. The VBA code uses a special sorting scheme that allow variable names (that appear in the same expression) that are subsets of other variable names to be replaced starting with the variables with longer names. Thus, for example you can declare variables with names such as T, T2, and TT2. The function *Fx* will replace the names of TT2, then T2, and then T with the values of these variables. To distinguish between the names of regression variables and possible transformation functions (especially those with overlapping characters

with the regression variable names), the VBA code temporarily appends a punctuation character (default is the \$ sign) to the variable names. Since function names cannot have the \$ sign as part of their names, there is no mix up in replacing the regression variable names with their numeric values. This approach allows the VBA code to work/use variable names that are not VBA-compliant, since the variable names must be replaced with their numeric values. Appending a punctuation character to the variable names is a simpler solution than, say, forcing variable names to have all uppercase characters and function names to have all lowercase characters (or vice versa). Using character case to distinguish between variables and functions is a bit restrictive and is error prone to typos. The statement in the function, located before label ExitProc, invokes the VBA function *Evaluate* to evaluate the string variable sExpress that contains a string of a math expression containing numbers, functions, and operators.

	
	<p>The VBA function <i>Evaluate</i> evaluates expressions that may include the name of functions. It performs its task by invoking the Excel mathematical interpreter (made up of an expression parser and an evaluator). The function names handled by function <i>Evaluate</i> must match the ones supported by the Excel spreadsheets and not VBA itself. For example, the function names <i>LN</i>, <i>LOG</i>, and <i>SQRT</i> are evaluated as the natural logarithm, the common logarithm, and the square root, respectively. These are functions supported by the spreadsheet. By comparison, VBA itself uses <i>Log</i> to evaluate the natural logarithm and <i>Sqr</i> to evaluate the square root. However, VBA has no predefined function for the common logarithm. You can access the wealth of Excel functions by writing short wrapper functions (in a module) that use the syntax <i>WorksheetFunction.fx_name</i> to access an Excel's function.</p> <p>The good news here is that there are no character-case restrictions for function names, as long as function <i>Evaluate</i> can call these functions.</p>

The parameter VarNames passes the array of the regression variable names. The parameter nVarIdx passes the array of indices that determine the order of accessing the elements of array VarNames. The parameter X passes the values for the regression variables. The arrays VarNames, nVarIdx, and X have zero-based indices. The values at index 0 refer to the dependent regression term/variable.

If a runtime error occurs in function *Fx*, the program execution will stop because such a runtime error is profoundly serious. The function displays (and verbalizes) a message to that effect. The message box also displays the current text in string variable *sExpress*. This information maybe key to dealing with the source of error. You may also need to inspect the values in the sheet *Data* and the models in sheet *ModelsList* to find the source of the error.

The Subroutine *BuildSuperRegressionModels*

The first task the subroutine *BuildSuperRegressionModels* does is to call function *unifyVarNames* to make sure that every variable name in sheet *Models* and *Normalize* also appear in sheet *Data*. If not, you get a fatal error message that mentions the name(s) of the unmatched variable(s) before the VBA code stops running.

The subroutine *BuildSuperRegressionModels* copies the values of sheet *Data* into sheet *OriginalData* if you request normalization of at least one regression variable. Upon successfully completing the regression calculations or reaching the maximum number of runtime errors, the VBA code copies the normalized values in sheet *Data* to sheet *NormalizedData*, before copying the original values from sheet *OriginalData* back to sheet *Data*. These copying tasks occur only if you requested the normalization of data. The normalized values in sheet *NormalizedData* are there for your curious inspection.

In subroutine *BuildSuperRegressionModels* you will see the following loop:

```

NumVars = 0
NumTerms = 0
For ModelRow = 1 To MaxModelRows
    Call checkIfCellIsEmpty("Variable name", ModelRow + 1, 1, sModelsSheet)
    sVarNames(ModelRow) = Sheets(sModelsSheet).Cells(ModelRow + 1, 1)
    Call checkIfCellIsEmpty("From Power", ModelRow + 1, 2, sModelsSheet)
    iFrom(ModelRow) = SCALE_POWER * Sheets(sModelsSheet).Cells(ModelRow + 1,
2)
    Call checkIfCellIsEmpty("To Power", ModelRow + 1, 3, sModelsSheet)
    iTo(ModelRow) = SCALE_POWER * Sheets(sModelsSheet).Cells(ModelRow + 1, 3)
    ' enforce iFrom < iTo
    If iFrom(ModelRow) > iTo(ModelRow) Then
        ' swap values
        I = iFrom(ModelRow)
        iFrom(ModelRow) = iTo(ModelRow)
        iTo(ModelRow) = I
        ' also swap cell values
        I = Sheets(sModelsSheet).Cells(ModelRow + 1, 2)

```



```

        Sheets(sModelsSheet).Cells(ModelRow + 1, 2) =
Sheets(sModelsSheet).Cells(ModelRow + 1, 3)
        Sheets(sModelsSheet).Cells(ModelRow + 1, 3) = I
    End If
    ' enforce iStep > 0
    Call checkIfCellIsEmpty("Power Scale", ModelRow + 1, 4, sModelsSheet)
    iStep(ModelRow) = SCALE_POWER * Abs(Sheets(sModelsSheet).Cells(ModelRow +
1, 4))
    If iStep(ModelRow) = 0 Then iStep(ModelRow) = 1
    ' ignore TermMathExpr contents in rows 2 and in the last row
    If ModelRow > 1 Then ' And ModelRow < MaxModelRows Then
        sTermMathExpr(ModelRow) = Sheets(sModelsSheet).Cells(ModelRow + 1, 5)
        If Left(sTermMathExpr(ModelRow), 1) = "'" Then sTermMathExpr(ModelRow)
= Mid(sTermMathExpr(ModelRow), 2)
    Else
        Sheets(sModelsSheet).Cells(ModelRow + 1, 5) = ""
        sTermMathExpr(ModelRow) = ""
    End If
    If Len(sTermMathExpr(ModelRow)) = 0 Then NumTerms = NumTerms + 1
    idx(ModelRow) = iFrom(ModelRow)
Next ModelRow

```

Also notice that the statements that assign values to the transformation-range arrays `iFrom()`, `iTo()` and `iStep()`, multiply the values in the source cells of sheet *Models* by the value of switch `SCALE_POWER` (currently set to 100)! Why do that, you may ask? I discovered that it is better to convert floating point numbers into integers to perform more robust logical tests between integers. That is why I do not recommend entering values in the *From Power*, *To Power*, and *Power Step* columns that have three or more decimal places. If you do, then you need to replace the value of switch `SCALE_POWER` with a larger power of 10 to make sure that all significant decimal digits become part of an integer.

Also notice in subroutine *BuildSuperRegressionModels* the following code fragment that builds each term using one or more variables:

```

Do
    DoEvents
    DoEvents
    Col = 1

    For I = 1 To N
        Sheets(sModelsListSheet).Range("AAA1").Offset(row - 1, I - 1).Value =
idx(I) / 100
        If idx(I) = 0 Then
            If LN_SHIFT_VAL <> 0 Then
                sTransformedVar = "LN(" & sVarNames(I) & "+" & LN_SHIFT_VAL & ")"
            Else
                sTransformedVar = "LN(" & sVarNames(I) & ")"
            End If
        End If
    Next I

```



```

ElseIf idx(I) = SCALE_POWER Then
    sTransformedVar = sVarNames(I)
ElseIf idx(I) = -SCALE_POWER Then
    sTransformedVar = sVarNames(I) & "^(-1)"
ElseIf idx(I) > 0 Then
    sTransformedVar = "(" & sVarNames(I) & "^" & CStr(idx(I) / SCALE_POWER)
& ")"
ElseIf idx(I) < 0 Then
    sTransformedVar = "(" & sVarNames(I) & "^(-" & CStr(Abs(idx(I)) /
SCALE_POWER) & ") )"
Else
    sTransformedVar = sVarNames(I)
End If

' no math expression or cross-product operator?
If Not bcMEfound And Len(sTermMathExpr(I)) = 0 Then
    If Left(sTransformedVar, 1) = "(" And Right(sTransformedVar, 1) = ")"
Then
        sTransformedVar = Mid(sTransformedVar, 2)
        sTransformedVar = Left(sTransformedVar, Len(sTransformedVar) - 1)
    End If
    Sheets(sModelsListSheet).Cells(row, Col) =
Sheets(sModelsListSheet).Cells(row, Col) & sTransformedVar

    Col = Col + 1
' found cross-product operator?
ElseIf (Not bcMEfound) And (InStr("+-/*,", Left(sTermMathExpr(I), 1)) >
0) Then
    Sheets(sModelsListSheet).Cells(row, Col) =
Sheets(sModelsListSheet).Cells(row, Col) & sTransformedVar & sTermMathExpr(I)
Else
    ' handle math expressions
    sTME = sTermMathExpr(I)
    ' continue to process current math expression?
    If bcMEfound Then
        ' now process the first variable in the compact math expression
        CMEidx = CMEidx + 1
        CMEexpress = Replace(CMEexpress, QUASI_VAR_FIRST_CHAR & CMEidx &
APPENDED_CHAR_TO_VARNames, sTransformedVar)
        bcMEfound = IIf(CMEidx = CMEmaxIdx, False, True)
        If Not bcMEfound Then
            Sheets(sModelsListSheet).Cells(row, Col) = CMEexpress
            Col = Col + 1
        End If
    ' found a new math experssion
    ElseIf Not bcMEfound And Left(sTME, 1) = "=" Then
        bcMEfound = True
        ''' CMEexpress = Replace(UCase(Mid(sTME, 2)), " ", "")
        CMEexpress = Replace(Mid(sTME, 2), " ", "")
        K = FindLastCharAndDigit(CMEexpress, QUASI_VAR_FIRST_CHAR)
        L = InStrRev(CMEexpress, "|")
        CMETrailNumber = 0
        ' Is there a trailing comma in mth expression?

```

```

    If L > K Then
        CMEmaxIdx = getNumber(CMEexpress, L + 1)
        CMEtrailNumber = CMEmaxIdx
        CMEexpress = Left(CMEexpress, L - 1)
    Else
        If K = 0 Then FinalMessage "Fatal error in math expression in row
" & CStr(I + 1) & " of Models sheet"
        CMEmaxIdx = getNumber(CMEexpress, K + L2)
        If CMEmaxIdx = 0 Then FinalMessage "Fatal error in math expression
in row " & CStr(I + 1) & " of Models sheet"
        End If
        CMEidx = I + 1
        ' check for incomplete quasi-var names
        Call checkforIncompleteQuasiVar_names(CMEexpress)
        ' Append punctuation char to quasi-var names
        Call appendPunctCharToQuasiVars(CMEexpress)
        ' check that quasi-var numbers are valid and in range
        Call checkQuasiVarIndices(CMEexpress, CMEmaxIdx, I + 1)
        ' check for duplicate quasi-vars in a math expression
        Call checkDuplicateQuasiVars(CMEexpress, I + 1, CMEmaxIdx)
        ' now process the first variable in the math expression
        CMEexpress = Replace(CMEexpress, QUASI_VAR_FIRST_CHAR & CMEidx &
APPENDED_CHAR_TO_VARNAME, sTransformedVar)
        End If
    End If
Next I

```

Notice that several ElseIf clauses compare values of idx(I) with SCALE_POWER or -SCALE_POWER. These values represent the scaled-up values of 1 and -1, respectively.

The last aspect I want to draw your attention to is the coded trick where I simulate nested loops using just one For loop:

```

' Start implementing the quasi-nested loops
idx(1) = idx(1) + iStep(1)
If idx(1) > iTo(1) Then
    idx(1) = iFrom(1)
    For I = 2 To N
        idx(I) = idx(I) + iStep(I)
        If idx(I) > iTo(I) And I < N Then
            idx(I) = iFrom(I)
        ElseIf idx(I) > iTo(I) And I = N Then
            Exit Do
        Else
            Exit For
        End If
    Next I
End If

```

The nested loop uses the arrays `idx()`, `iFrom()`, `iTo()`, and `iStep()` in a scheme that resembles a logical ticking clock. The array element `idx(i)` simulates the current loop control variable for variable `i`. You initialize the values of `idx(i)` with the corresponding values of `iFrom(i)`. The Do Until loop that I showed earlier in this subsection performs this initialization. Using this programming trick, the VBA code can handle any number of terms (i.e., nested loops) in the regression model.

Fatal Runtime Errors

The VBA code checks numerous aspects of the data, variable names, and math expressions. In addition, function *Fx* may still catch runtime errors. All these errors cause the program execution to stop, because there is no sense in continuing calculations that will yield wrong results. The VBA code speaks the error message (when enabled) and displays it in a message box. The messages identified the error and its source, in as much as possible. Check the following sources to determine the cause of the fatal errors:

- The data in the *Data* sheet.
- The names of the regression variables in the *Data*, *Models*, and *Normalization* worksheets.
- The math expressions (check the names of the quasi-variables) or single-operator transformations in the *Models* sheet.
- The names of the transformation functions in the math expressions or in the VBA code.

Customizing the Transformations with VBA and User-Defined Functions

The VBA code supports three types of transformations:

- The default transformations that apply different powers (and the natural logarithm) to single regression variables.
- Simple multiplicative cross-products of two or more regression variables.
- Mathematical expressions for different terms that can use various operators, parenthesis, and even functions. These expressions that allow the terms of regression models to support more elaborate mathematical expressions such as:

$$f_1(x_1) + f_2(x_2) / (f_3(x_3) + f_4(x_1) + f_5(x_2))$$

Where f_1 through f_5 are transformations of the different regression variables. You can include VBA functions or user-defined functions in the math

expressions. Keep in mind that each model generated will have these functions as part of the terms they are associated with.

Using Term Math Expressions

Earlier in this paper I mentioned that simple cross-product terms can be defined using text like '=A8*A9 in a TME cell. This is really the tip of the iceberg. You can write more elaborate math expressions that use math operators, parentheses, and even functions to create more sophisticated expressions for regression terms. This is really stepping on the proverbial gas! Keep in mind that using functions and explicit operators (like raising to a specific power or adding a fixed value) in a math expression **applies to each and every regression term created with that math expression**. It is what I call *fixed transformations*.

The switch QUASI_VAR_FIRST_CHAR contains the first character(s) in the names of quasi-variables used in math expressions. The constant has the default value of "A". I chose that value because the names of the regression variables in sheet *Models* appear in column A. I recommend you keep the default assignment, unless you are getting conflicts with your custom function names that end with "A" or "a" and are followed by one or more digits. In this case, you have one of two general solutions:

- You can replace the letter A with another letter that will not reproduce the above problems. Letters like Z or V are good general candidates for the first letters in quasi-variable names. You are the ultimate judge in deciding what works for you. When you assign a new character to switch QUASI_VAR_FIRST_CHAR remember to change the quasi-variable names in the math expressions to match the new leading character. Your math expressions would look like '=V14*V15*V16 or '=Z14*Z15*Z16.
- You can select appending a punctuation character like % or # to the letter A and ending up with A% or A# as the leading characters of quasi-variable names. You then edit switch QUASI_VAR_FIRST_CHAR and assign something like "A%" or "A#" to it. When you assign new characters to switch QUASI_VAR_FIRST_CHAR remember to change the quasi-variable names in the math expressions to match the new leading characters. Your math expressions would look like '=A%14*A%15*A%16 or '=A#14*A#15*A#16.
- You can combine both of the above solutions. Your math expressions would look like '=Z%14*Z%15*Z%16 or '=V#14*V#15*V#16.

When you work with any quasi-variables An think of it as “a variable with any of its transformations” and not just a variable. In other words, a quasi-variable is defined by two attributes. Also think of An as a unique pointer to a regression variable in the *Models* sheet. The regression variable in each row of the sheet *Model* can have its own transformation range and increment, even when multiple rows in sheet *Models* can refer to the same regression variable.

👉 You can use Excel functions and your own custom functions in math expressions. These functions will appear in their associated term in every regression model. When using Excel functions that are not explicitly part of the VBA math function library, like $\text{Sinh}(x)$, you will need to write a wrapper function in a module, such as:

```
Function Sinh(Byval X as Double) As Double
    Sinh = Worksheetfunction.Sinh(x)
End Function
```

Likewise, to use your custom function, such as $\text{MyFx1}(x)$, you define that function in a module:

```
Function MyFx1(Byval X as Double) As Double
    MyFx1 = Exp(x) - 3*x^2
End Function
```

You can then include the names of these functions in your math expressions. Unlike the other two versions of the Excel application in the Clement 2 project, I do not support associating custom functions with iterative search for the best model. I don't see how introducing functions like sine, cosine, sinh, tanh, and so on, helps transition the various terms through a smooth and gradual curvature. I feel using such functions would be more of a roller coaster experience—one that can be highly vulnerable to errors in the data.

Using term math expressions gives you a lot of power that you should be **use very carefully to avoid runtime errors**. You cannot use the math expression transformations with the dependent variable term. Figure 12 shows various examples of what you can do with term math expressions. Keep in mind that these expressions work on the various power transformations and custom functions for the different regression variables. What Figure 12 shows is just the tip of the iceberg. You can get creative with the sophistication of the math expressions. Expect the calculations process to take a considerable amount of time. However, you may stumble on relevant expressions that describe relations between variables that are far from being


obvious or theoretically based. And that, my dear reader, is where this VBA application shines! That is the power of brute force search for empirical regression models.

<i>Math Expression Example</i>	<i>Appears in Row</i>	<i>Covers Last Row</i>	<i>Comments</i>
'=A8*A9	8	9	Simple multiplicative cross products for the transformations of the variables named in cells A8 and A9.
'=A8*A9*A10	8	10	Multiplicative cross products for the transformations of the three variables named in cells A8, A9 and A10.
'=(A8)^(A9)	8	9	Raises the value of transformed variable named in cell A8 to the variable named in cells A9. Notice that I have enclosed each quasi-variable in pairs of parentheses to ensure that VBA correctly evaluates the expression.
'=LN((A2+1)^(A3))	2	3	Calculates the natural logarithm of the result of raising the value of transformed variable named in cell A8, plus one, to the variable named in cells A9. Notice that I have enclosed each quasi-variable in pairs of parentheses to ensure that VBA correctly evaluates the expression.
'=(A8*A10)+A9 10	8	10	Expression to multiply the transformations of the variables named in cells A8 and A10 and then add the results to the transformations of the variable named in cell A9.
'=(A8+A9)*(A10+A11)	8	11	Expression to multiply the results of adding the transformations of the variables named in cells A8 and A9 with the results of adding the transformations of the variables named in cells A10 and A11.
'=(A8+A9)/(A10+A11)	8	11	Expression to divide the results of adding the transformations of the variables named in cells A8 and A9 by the results of adding the

<i>Math Expression Example</i>	<i>Appears in Row</i>	<i>Covers Last Row</i>	<i>Comments</i>
			transformations of the variables named in cells A10 and A11.
'=(A8+A9) / (A12+A11) *A10 12	8	12	Expression to divide the results of adding the transformations of the variables named in cells A8 and A9 by the results of adding the transformations of the variables named in cells A12 and A11, and then multiplying by the variable named in cell A10.
'=LN ((A8*A11) + (A10*A9)) 11	8	11	Expression to takes the natural logarithm of adding the results of multiplying the transformations of the variables named in cells A8 and A11 with the results of multiplying the transformations of the variables named in cells A10 and A9. The use of the natural logarithm is consistent for all the terms created with that math expression.
'= ((A8*A9) + (A10*A11)) / ((A12*A15) + (A14*A13) 15	8	15	Expression for the ratio of the products of two sums of variables whose names appear in cells A8 through A15.
'=SQRT (A8) +SQRT (A9)	8	9	Expression for the sum of the square roots of the regression variables named in cells A8 and A9. The use of the square roots is consistent for all the terms created with that math expression.
'= (A8+A9) /LN ((A10+A11) ^2+1)	8	11	Expression to divide the results of adding the transformations of the variables named in cells A8 and A9 by the results of the natural logarithm of squared sum of the transformations of the variables named in cells A10 and A11 plus 1. The use of the natural logarithm, the explicit squaring operator, and the addition of 1 is consistent for all the terms created with that math expression.
'= (A8+A9) /SQRT ((A10+A11) ^2+1)	8	11	Expression to divide the results of adding the transformations of the variables named in cells A8 and A9 by the results of the square root of the sum of the transformations of the variables named in cells A10 and A11. The use of the square root, the explicit

<i>Math Expression Example</i>	<i>Appears in Row</i>	<i>Covers Last Row</i>	<i>Comments</i>
			squaring operator, and the addition of 1 is consistent for all the terms created with that math expression.

Figure 12. Various examples of math expressions that can be used in the Models sheet.

	
	<p>The math expression transformation is a powerful transformation feature. With power comes responsibility. Plan very carefully the value you enter in the TME cells. I strongly recommend you separately run subroutine <i>BuildSuperRegressionModels</i> and examine the output in the <i>ModelsList</i> sheet. Edit the math expressions in the TME cells until you get correct terms in the <i>ModelsList</i> sheet. Once you get correct models, you can then perform the regression calculations</p>

The subroutine *BuildSuperRegressionModels* performs the following basic checks for each math expression in column E of the *Models* sheet:

- The subroutine examines the number of open and close parentheses and also the order of their appearance.
- The subroutine checks for invalid quasi-variable indices.
- The subroutine checks for invalid high row limits if the math expression uses a trailing bar character followed by a number that should represent the high row limit for the math expression.
- The subroutine checks for duplicate quasi-variables sharing the same index.
- The subroutine checks for bare quasi-variable names (with no numbers) in the math expression, such as ')A(', ')A+', 'A12A*', 'AA*', '*A+', and so on. The subroutine will do its earnest to catch common errors related to invalid quasi-variable names. The subroutine may well miss more complex errors. As a result, the subroutine *BuildSuperRegressionModels* will proceed with building a list of incorrect models! The function *Fx* is the software component that will champion catching incorrect model errors, declaring them as fatal runtime errors and stopping program execution.

If the subroutine *BuildSuperRegressionModels* finds any error, it will speak (when enabled) and display a fatal error message. Keep in mind that the subroutine's tests are very basic and serve to catch trivial errors. The information in the TME cells of the *Models* sheet can still generate runtime or logical error-prone expressions.

Epilogue

Using this optimum search version of the Excel application may require several runs. You can experiment with the size of the step values and the initial ranges of the

transformations. I found that it is possible to get more than one good regression model when using different initial ranges for the transformations.

You can use three-point ranges (my recommendation) or five-point ranges. Use the latter when your models involve relatively fewer regression variables.

Appendix A – Source of Math-Related Runtime Errors

The next table summarizes the various kinds of math-related runtime errors.

<i>Case</i>	<i>Arguments</i>	<i>Comments</i>
LN function used in math expressions.	Negative or zero values.	Generate errors in all cases.
LN function used in math expressions.	1	When divided by LN(1) you get division by zero error.
LN function due to 0 transformation	Negative or zero values.	Generate errors in all cases.
LN function due to 0 transformation	1	When divided by LN(1) you get division by zero error.
Powers	Raising negative numbers to negative powers.	

Document History

<i>Version</i>	<i>Release Date</i>	<i>Comments</i>
1.0.0	June 17, 2021	Initial release.