Extensive Empirical Modeling Using Excel VBA

By
Namir Clement Shammas

Contents

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 an 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 my most beloved son Joseph Shammas, PhD. As a data scientist, he lovingly advised me that I was wasting my time doing this project using a dinosaur like Excel VBA. As someone who still tinkers with over-forty-year-old programmable calculators, using Excel VBA should come as no surprise.

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 applications. This study presents the first and initial part of the Clement 2 project. It offers a sweeping approach that uses a grid of transformations searching for the best model.

Introduction

"Magic Mirror, on the wall, who, now, is the fairest one of all?"
– quote of the Evil queen in the Snow-White legend.

This paper presents 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 essentially uses a brute force approach-*assume little and search a lot*! Such models may reveal transformations and relationships between variables that were not obvious at the outset. If you are analyzing a huge number of observations, then the VBA code would better serve you as a *prototype study* you employ to examine smaller samples of your data. To process a huge number of observations, you may opt to use code you develop (or have it written for you) in other languages like Python, SQL, MATLAB, C++, or R. The applications in these languages may still read and write information from and to Excel files but do the calculations outside Excel, giving you some speed advantage and a higher data capacity. 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. If you are willing to divide your large data into small enough sets, Excel would be able to handle the job.

The very root of the concept for the VBA code goes back four decades of using vintage HP programmable calculators and their statistical applications. These applications allowed you to select the best fit among linear, exponential, logarithmic, and power regression models for your (x, y) data. I wasted no time in extending the selection of models for these HP calculator applications by writing programs that performed more transformations on the (x, y) data. I easily went from selecting from 4 models into selecting from 64 models by simply giving each regression variable eight possible transformations—x, 1/x, ln(x), $\sqrt{x}$, $1/\sqrt{x}$, $x^{1.5}$, x^2, and $1/x^2$.

I adapted the above simple concept to multiple regression between three variables that examined up to 512 models (based on 8 transformations per variable). I applied the best curve fit to calculators and over time migrated to Excel VBA to select the models with the best transformations. The ability to view Excel sheets to inspect data, transformations, and results, proved to be most valuable and unrivaled! I was also able to explore empirical modeling for more than three variables. Over time I

explored different schemes for applying the transformations to regression variables. Early transformation schemes dealt with the various transformation as an enumerated set of functions. Recently, I used the transformations as mostly powers of values that supply different curvatures. This study brings the fruits of several decades of writing programs that seek the best empirical models. This study incorporates, for the first time, the ability to include the cross-products of two or more variables and also use powerful math expressions in the regression models. As I will discuss later, my preferred scheme of transformations is based on changing the curvature of regression variables by raising these variables to integer and floating-point powers.

This study allows you to determine the best models in the following general schemes:

$$f_0(y) = a_0 + a_1 \cdot f_1(x_1) + a_2 \cdot f_2(x_2) + \ldots + a_n \cdot f_n(x_n) \quad (1)$$

$$f_0(y) = a_0 + a_1 \cdot f_1(x_1) + a_2 \cdot f_2(x_2) + \ldots + a_n \cdot f_{n1}(x_1) \cdot f_{n2}(x_2) + \ldots \quad (2)$$

Equation (1) allows you to perform multiple linearized regression on several independent variables with $f_i(x)$ being the transformations applied to the variables. The default transformations for $f_i(x)$ are either x^i or $\ln(x)$. We can extend equation (1) to include cross–products with transformations, yielding equation (2). The latter offers a general form in which we can do powerful empirical curve fitting on data. The more independent variables we have, the more cross–product terms of two or more transformed variables we can have! It is easy to end up with hundreds or thousands of empirical models to inspect! This paper discusses how to explore the power of equation (2). Applying equation (1) is a mere subset. The paper also presents the use of math expressions that include two or more regression variables.

☛ 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!

"Fasten your seatbelts, it's going to be a bumpy night."
From the movie "The Wisdom of Eve"

Let me guide you through a quick inspection and test for the Excel file using the demo file BMLR*demo.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 BMLR*demo.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 sets the dependent variable to remain linear while all other regression variables have linear and quadratic transformations.
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 most the VBA code**. **The rest of the VBA code is in Module1.**
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 do a complete cycle of building the regression models list and then testing each model. During the program execution you will hear messages related to the progress of this process. When the VBA code is done executing it will ask you if you want to remove the trailing punctuation characters from all of the variable names in all of the sheets. Click Yes. You will see the *Results* sheet. Inspect that sheet

with the sorted results. Notice that the values of the R-square and F statistic decrease down columns A and B, respectively.

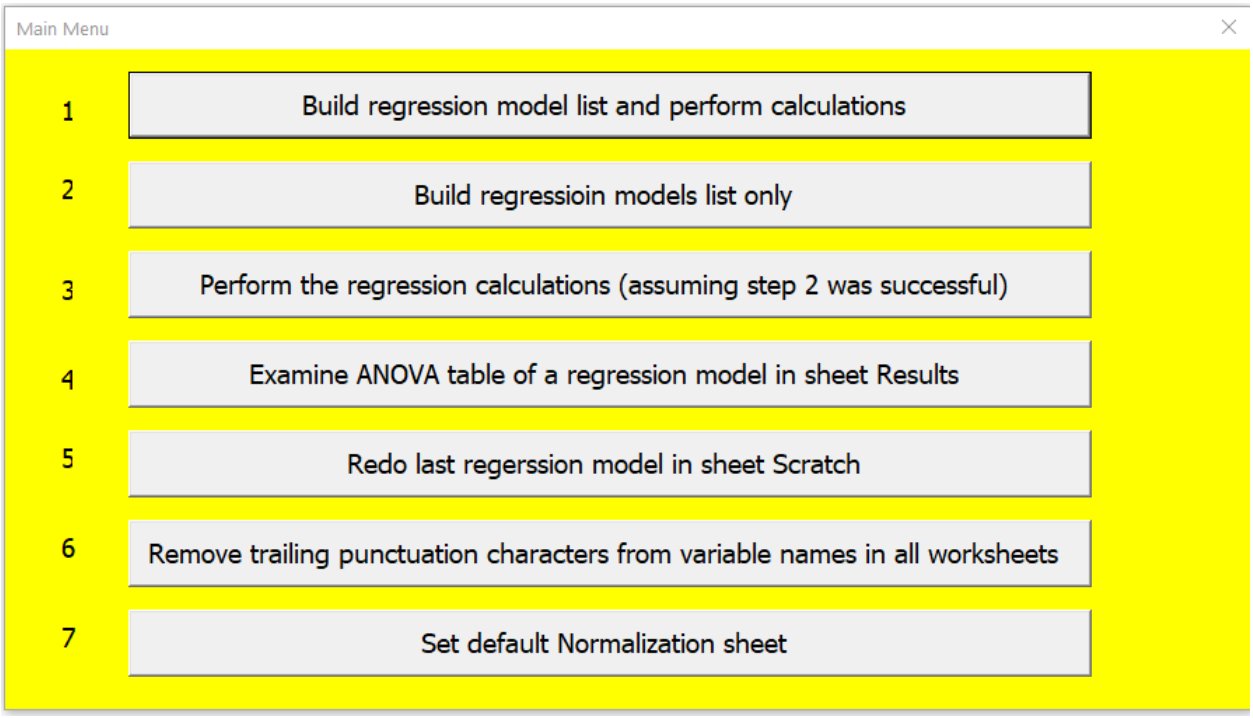

Figure 1. The Menu input box

8. Click on the *Scratch* sheet to view the regression data and results for the best regression model that the VBA code processed.
9. Once you are done with the demo file, close it. The file *BLMRv1_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 *tinker* and *experiment* with data and regression models.
☛	Make sure that the *Analysis ToolPak* and the *Analysis ToolPak – VBA* 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 *BMLRdemo.xlsm*, the Main Menu form loads automatically. When you load the distribution file *BMLRv1_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 multiple 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

"If you build it, they will come."
From the movie “Field of Dreams”

The *BMLRv1_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.

Sheet Name	***Purpose***
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.
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 *Data*. The VBA code backs up (and later restores) the original values of sheet *Data* 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 *Data* 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 for custom transformations.
    - 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.
- 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 to the variable names during regression calculations. **By *temporary* I mean that when *you are done* with the analysis of the regression models and inspecting individual regression models, you have the following choices:**
    - Set switch REMOVE_TRAILING_PUNC_CHAR to 1 to let the VBA automatically remove the trailing punctuation characters from all variable names.
    - Set switch REMOVE_TRAILING_PUNC_CHAR to 0 to make the VBA code ask you about removing the trailing punctuation characters from all variable names. Click Yes to the prompt if you are done with the analysis.
    - **If you click No to the prompt, then you need to later on click on command button 6 to remove the punctuation characters from all the variable names in all of the sheets**. **Do this after you are done with inspecting individual regression models in more details**. The default setting is the $ character. It is a good choice since valid function names cannot include the $ character.
- 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 5.
- 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 a numeric 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.

	A	B	C
1	***Switch***	***Value***	***Comments***
2	RESULTS_SORT_COLUMN	A	Name of column in Results sheet used to sort the results
3	SAYIT	1	Turn on verbal messages 0=OFF non-0=ON
4	WAIT_ON	1	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	15	maximum number of errors that cause the calculations to stop
10	SCALE_POWER	100	Scale factor for powers used to rais 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	MAX_RESULTS	50	Maximum number of results
14	BAR_CHAR	|	Specifies the character used after a math expression to define the last row covered by that expression.
15	REMOVE_TRAILING_PUNCT_CHAR	0	Remove trailing punctuation character without offering choice for user when set to non-zero.

Figure 2. The Switches form.

☛The Main Menu form has multiple 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 *Switches* 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 VBE code editor 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 Zfct, Zfct$, or Zfct$, 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
2	721	11	55	1.5
3	466	4	65	2
4	380	2	67	3
5	662	9	48	4

	A	B	C	D
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 expressions 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. Rather than forcing you to type by hand hundreds or even thousands of rows that define each model, this worksheet uses a much clever scheme as a shortcut. 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 range of powers defined by values in columns B, C, and D. These powers can be positive, zero, or negative. 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 or 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 you use negative powers or high–valued positive powers, I strongly suggest that you first 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 scheme of transformation uses a range of powers with a step increase in power. You specify the minimum power, the maximum power, and the power increment. This scheme is similar to the VBA For Next loop with a Step clause.

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	1	1	1	
3	T	1	2	1	
4	P	1	2	1	
5	V	1	2	1	

	A	B	C	D	E
6	T	1	2	1	'=A6*A7
7	P	1	2	1	
8	T	1	2	1	'=A8*A9*A10
9	P	1	2	1	
10	V	1	2	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 Padé–style empirical fit. Keep in mind that to use such a model you need to employ an iterative process to calculate the value of the independent variable). 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 minimum 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 maximum 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.
- 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 A*n* 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 A*n*. 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 A*n* 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 A*n* 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 functions 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 transformations in the range of 1 to 1 in steps of 1. Thus, the transformations include linear values only. 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 that range from 1 to 2 in steps of 1. Thus, the transformations include the linear and squared values. The TME cells for these variables are empty to indicate that they occupy separate terms in the regression models.
- Rows 6 defines a simple cross-product for the variables T and P. The range of transformations are linear and squared values. 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 as the quasi-variables 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 transformations are the linear and squared values. 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 as the quasi-variables 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 VBA code applies the various transformations to the $f_i()$ and $f_{ij}()$ functions to obtain the best models. The VBA code generates hundreds of empirical models to test and find the best models! Of course, we do not want to see ALL these results. Instead, the VBA code chooses something like the best 50 models! You can change that limit by assigning a different value to the switch MAX_RESULTS.

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.

You can set the *Power Step* values in the *Models* sheet to be less than 1 (but not less than 0.01). For example, a value of 0.5 (with *From Power* = 0 and *To Power* = 2) would generate the sequence of powers of 0 (for the natural logarithm), 0.5 (for the square root), 1, 1.5, and 2.

☛ It does not take many transformations for each variable/term to generate over tens of thousands of models! That is why I highly 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 models that 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	1	1	1	
3	T	1	2	1	
4	P	1	2	1	
5	V	1	2	1	
6	T	1	2	1	*
7	P	1	2	1	
8	T	1	2	1	*
9	P	1	2	1	*
10	V	1	2	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.

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

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

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

- The second row declares the variable Z to have transformations in the range of 1 to 1 in steps of 1. Thus, the transformations include linear values only. 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 that range from 1 to 2 in steps of 1. Thus, the transformations include the linear and squared values. 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 of transformations are linear and squared values. 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. The range of transformations are the linear and squared values. 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 transformation scheme presented in this subsection is an older application version that I had developed for the VBA code. At the time, the scheme proved to be very simple and efficient. I later decided to incorporate math expressions for the regression terms. I modified that old version, but the resulting method of working with the contents of the TME cells became a bit more complicated and quite vulnerable to easily making mistakes in defining the math expressions. That is when I switched to implementing math expressions that I presented in the previous subsection. The single-line definition allows the user to easily write, edit, and read math expressions written on one line. This is in contrast to fragmented instructions spread over several rows of the TME column—something that is a bit hard to follow.

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	F
1	Z	T	P	V	T*P	T*P*V
2	Z	T^2	P	V	T*P	T*P*V
3	Z	T	P^2	V	T*P	T*P*V
4	Z	T^2	P^2	V	T*P	T*P*V
5	Z	T	P	V^2	T*P	T*P*V
6	Z	T^2	P	V^2	T*P	T*P*V
7	Z	T	P^2	V^2	T*P	T*P*V
8	Z	T^2	P^2	V^2	T*P	T*P*V
9	Z	T	P	V	T^2*P	T*P*V
10	Z	T^2	P	V	T^2*P	T*P*V
11	Z	T	P^2	V	T^2*P	T*P*V
12	Z	T^2	P^2	V	T^2*P	T*P*V
13	Z	T	P	V^2	T^2*P	T*P*V
14	Z	T^2	P	V^2	T^2*P	T*P*V
15	Z	T	P^2	V^2	T^2*P	T*P*V
16	Z	T^2	P^2	V^2	T^2*P	T*P*V
17	Z	T	P	V	T*P^2	T*P*V
18	Z	T^2	P	V	T*P^2	T*P*V
19	Z	T	P^2	V	T*P^2	T*P*V
20	Z	T^2	P^2	V	T*P^2	T*P*V
21	Z	T	P	V^2	T*P^2	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. 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 Results Sheet

The *Results* sheet displays a sorted list of the best (fifty) regression models. 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	Transf of X5	Coeff0	Coeff1	Coeff2	Coeff3	Coeff4	Coeff5
0.9397515	38.43501634	Z	T	P^2	V^2	T^2*P^2	T^2*P^2*V	682.140109	-55.62482386	-0.042941808	-0.044230979	0.001691435	0.000315937
0.939737086	38.42548882	Z	T	P^2	V	T^2*P^2	T^2*P^2*V	688.5785405	-55.91555151	-0.043754877	-0.896002957	0.00168943	0.000317963
0.932613886	34.21564641	Z	T	P	V^2	T^2*P^2	T^2*P^2*V	817.0399804	-62.96945073	-4.655646931	-0.105849326	0.001834558	0.000352988
0.932319226	34.06058759	Z	T	P	V	T^2*P^2	T^2*P^2*V	781.4931873	-62.22100668	-4.163841766	0.153400889	0.001828041	0.000357419
0.896204052	21.72228999	Z	T	P^2	V^2	T^2*P^2	T*P^2*V	619.7030684	-35.30555443	-0.047355298	-0.556385008	0.001455125	0.002319201
0.890964304	20.61114024	Z	T	P^2	V^2	T^2*P	T*P^2*V	506.9657384	-60.06373071	-0.011191655	0.381070792	0.115623045	0.00268083
0.8887208	20.16737293	Z	T	P	V^2	T^2*P	T*P^2*V	474.1698676	-59.4222053	-0.307361077	0.536915761	0.116094049	0.002810198
0.885817007	19.61889209	Z	T	P^2	V^2	T^2*P^2	T*P*V	683.5720148	-57.61650037	-0.048814822	-1.180530661	0.001952573	0.138040888
0.885659478	19.58993392	Z	T	P^2	V	T^2*P	T*P^2*V	550.1447976	-59.48059731	-0.020334446	1.963903523	0.112713495	0.002488317
0.884411639	19.36333612	Z	T	P^2	V^2	T^2*P^2	T^2*P^2*V^2	843.5878021	-64.43745433	-0.071095858	-1.220547678	0.002175147	4.17762E-05
0.883394686	19.18225231	Z	T	P^2	V	T^2*P^2	T*P^2*V	545.7402339	-33.92930556	-0.031977285	-2.920698129	0.001466754	0.002610145
0.8827007	19.06048024	Z	T	P	V	T^2*P	T*P^2*V	444.4809529	-57.36171347	-0.11317263	8.009685475	0.114558791	0.002609426
0.878137936	18.29439805	Z	T	P	V^2	T^2*P^2	T*P^2*V	699.4556313	-38.62341299	-4.324941935	-0.526982424	0.001548402	0.002671698
0.877510928	18.19358468	Z	T	P	V	T^2*P^2	T*P^2*V	220.3868216	-24.95321475	1.803137399	12.75407683	0.001463739	0.002777845
0.876425449	18.0214745	Z	T^2	P	V	T^2*P	T^2*P^2*V	-68.91849599	-3.682397981	5.92649469	27.04585977	0.114203279	0.000220921
0.868510588	16.85242027	Z	T^2	P	V	T^2*P	T*P^2*V	-257.4829708	-2.282226835	7.924936292	31.51063679	0.103765659	0.00193785
0.866699036	16.60437096	Z	T	P	V^2	T^2*P^2	T*P*V	774.6443277	-64.73105011	-4.457432888	-1.247131221	0.002130618	0.160653385
0.8655186	16.44633411	Z	T	P^2	V	T^2*P^2	T^2*P^2*V^2	1070.412737	-74.5258043	-0.100676317	-28.50222666	0.002200419	4.95782E-05
0.857111656	15.39633149	Z	T	P^2	V^2	T^2*P^2	T*P^2*V^2	692.5556177	-36.49541351	-0.051312011	-1.582116638	0.001614557	0.000310634
0.856847594	15.36534863	Z	T	P^2	V	T^2*P^2	T^2*P*V	982.0241056	-82.65053875	-0.086801181	-15.96353871	0.002284977	0.013363922
0.855672589	15.22885782	Z	T	P^2	V^2	T^2*P^2	T^2*P*V	834.4494293	-73.34581013	-0.066932667	-0.661046234	0.002234072	0.011613377
0.855068015	15.15949164	Z	T	P	V^2	T^2*P^2	T^2*P^2*V^2	1090.233024	-78.19047407	-7.935388985	-1.501213963	0.002513504	5.04756E-05
0.849082554	14.50273396	Z	T^2	P	V	T^2*P^2	T^2*P^2*V	-177.1718888	-1.675541859	7.291185423	31.0311106	0.001385357	0.000284402
0.847555094	14.34339255	Z	T	P^2	V	T*P	T^2*P^2*V	450.7362975	-60.08712862	-0.032161169	7.902384053	1.434826536	0.000226458
0.846707684	14.25636201	Z	T	P^2	V	T*P	T^2*P*V	505.1357466	-91.61664979	-0.047463719	6.928170944	2.024373901	0.011165129
0.846352502	14.22016972	Z	T	P^2	V^2	T*P	T^2*P*V	661.003612	-88.0591944	-0.074082443	-0.262519265	1.871380853	0.009763006
0.845293618	14.11325789	Z	T	P^2	V^2	T*P	T^2*P^2*V	616.651467	-60.60064458	-0.060841823	-0.206716	1.354122304	0.000199697
0.844701242	14.05408366	Z	T	P^2	V	T^2*P^2	T*P*V	781.3207212	-61.47375179	-0.057781319	-21.22911808	0.001939696	0.152839118
0.844484542	14.03254947	Z	T	P	V	T^2*P^2	T^2*P*V^2	-2659.956894	206.9751254	34.03138253	140.5746463	-0.002416005	-0.004589761

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. When the calculations are done, you see the results of the best regression model—assuming it the one you most likely want to see. 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	Z	T	P^2	V^2	T^2*P^2	T^2*P^2*V
	721	11	3025	2.25	366025	549037.5

Regression Statistics	
Multiple R	0.98227035
R Square	0.964855041
Adjusted R Square	0.9397515
Standard Error	27.91036355
Observations	13

ANOVA

	df	*SS*	*MS*	*F*	*Significance F*
Regression	5	149702.1582	29940.4316 3	38.4350163 4	6.05861E-05
Residual	7	5452.918754	778.988393 5		
Total	12	155155.0769			

	Coefficients	*Standard Error*	*t Stat*	*P-value*	*Lower 95%*	*Upper 95%*	*Lower 95.0%*	*Upper 95.0%*
Intercept	682.140109	98.46679237	6.92761582 4	0.00022566 8	449.3031437	914.977074 2	449.303143 7	914.977074 2
T	-55.62482386	7.703506222	-7.22071512 3	0.00017429 -73.84072149		- 37.4089262 2	- 73.8407214 9	- 37.4089262 2
P^2	-0.042941808	0.021671804	- 1.98145977 7	0.08799224	-0.094187481	0.00830386 5	- 0.09418748 1	0.00830386 5
V^2	-0.044230979	0.491510367	- 0.08998992 1	0.93081599 6	-1.206468313	1.11800635 5	1.20646831 3	1.11800635 5
T^2*P^2	0.001691435	0.000177908	9.50734638 8	2.98196E-05	0.001270749	0.00211212 1	0.00127074 9	0.00211212 1
T^2*P^2*V	0.000315937	6.70071E-05	4.71497460 9	0.00217023 3	0.0015749	0.00047438 4	0.00015749	0.00047438 4

RESIDUAL OUTPUT

Observation	*Predicted Z*	*Residuals*	Y Obs.	%Err
1	732.8372632	-11.83726325	721	- 1.64178408 4
2	435.0904299	30.90957009	466	6.63295495 4
3	425.1169052	-45.11690516	380	- 11.8728697 8
4	633.3791177	28.62088231	662	4.32339612 0.90267483
5	652.0603996	5.939600413	658	5 -
6	608.5661272	-4.566127166	604	0.75598131 9 -
7	577.7631582	-14.76315822	563	2.62223058 9
8	506.5477992	27.45220079	534	5.14086157 1
9	508.1057644	-15.10576443	493	-3.06404958 -
10	490.1404215	-10.14042146	480	2.11258780 3
11	329.4558993	10.5441007	340	3.10120608 8 -
12	471.9014285	-4.901428502	467	1.04955642 4
13	436.0352861	2.964713878	439	0.67533345 7

46		422			
6	4	5	4	67600	135200
38		448			
0	2	9	9	17956	53868
66		230			
2	9	4	16	186624	746496
65		250			
8	8	0	25	160000	800000
60	1	115			
4	3	6	36	195364	1172184
56	1	108			
3	2	9	49	156816	1097712
53		302			
4	1	5	64	3025	24200
49		168			
3	3	1	81	15129	136161
48		115			
0	5	6	100	28900	289000
34	1				
0	0	441	121	44100	485100
46					
7	6	841	144	30276	363312
43					
9	7	625	169	30625	398125

Figure 8. the 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 retrieved 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

"Working nine to five, what a way to make a living?!"
From the theme song of the movie "9 to 5"

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 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 range of transformations and their increments. Select the *Models* sheet and populate columns A to E with data that select the variable(s) for each term, specify the range and 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.

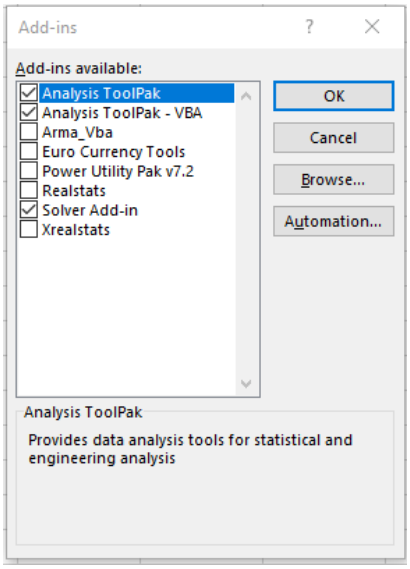

Figure 10. The Add–ins dialog box.

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) or break them down in two stages using command buttons 2 and 3.

You can also run the subroutines 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.

Subroutine Name	*Button Number*	*Purpose*
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.
BuildSuperRegressionModels	2	Build the models list. You can then inspect and edit the models list.
GoDoMultiplRegressionCalculations	3	Perform the regression calculations after using the above subroutine.
ExamineARegressionModel	4	Recalculate the regression ANOVA table of a specific model in the Results sheet.
RedoLastRegressionModel	5	Recalculate the regression ANOVA table after deleting one or more data columns in the *Scratch* sheet.
SetDefaultNormalization	7	Setup the default *Normalization* sheet with all normalization switches off.
RestoreData		Copies all the values from sheet *DataOriginial* to sheet *Data*.
restoreVarNames	6	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 transformation ranges.
3. Click on button 1 of the Main Menu form to execute subroutine *DoMultipleRegression*.

You can also divide the process into two stages. You can first build (and then inspect and/or edit) the list of models by clicking on button 2 in the Main Menu. This action launches subroutine *BuildSuperRegressionModels*. If you did not get any error messages and were **VERY** careful with any list edits, you can trigger the regression calculations by clicking on button 3 of the Main Menu form to invoke subroutine *GoDoMultiplRegressionCalculations*. This subroutine sets the GREEN_LIGHT global variable to True and then calls the subroutine *DoMultiplRegressionCalculations*. The latter subroutine does the "heavy lifting" regression calculations and sorting of results. The subroutine prepares the data in the *Scratch* worksheet and then invokes the subroutine *DoMLR2* in the module Module1. The subroutine *DoMLR2* invokes *ATPVBAEN.XLAM!Regress* (you must have the Add–ins option of *Analysis ToolPak – VBA* checked for VBA to be able to invoke it) to yield the familiar ANOVA regression table in the *Scratch* worksheet.

Notes on the VBA Code in the Data Sheet

"The devil is in the detail" -- Friedrich Wilhelm Nietzsche

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 image 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.

☛	
	The VBA function *Evaluate* 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 *Evaluate* must match the ones supported by the Excel spreadsheets and not VBA itself. For example, the function names *LN*, *LOG*, and *SQRT* 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 *Log* to evaluate the natural logarithm and *Sqr* 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 *WorksheetFunction*.fx_name to access an Excel's function. The good news here is that there are no character-case restrictions for function names, as long as function *Evaluate* can call these functions.

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 relational 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
    Col = 1

    For I = 1 To N
      If idx(I) = 0 Then
        s = "LN(" & sVarNames(I) & ")"
      ElseIf idx(I) = SCALE_POWER Then
        s = sVarNames(I)
      ElseIf idx(I) = -SCALE_POWER Then
        s = "1/" & sVarNames(I)
      ElseIf idx(I) > 0 Then
        s = sVarNames(I) & "^" & CStr(idx(I) / SCALE_POWER)
      ElseIf idx(I) < 0 Then
        s = "1/" & sVarNames(I) & "^" & CStr(Abs(idx(I)) / SCALE_POWER)
      Else
        s = sVarNames(I)
      End If

      ' no math expression or cross-product operator?
      If Not bCMEfound And Len(sTermMathExpr(I)) = 0 Then
          Sheets(sModelsListSheet).Cells(Row,              Col)              =
Sheets(sModelsListSheet).Cells(Row, Col) & s
          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) & s & 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, s)
          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_VARNAMES, sTransformedVar)
      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

"I'm not *buggy*. I'm just *coded* that way."
Paraphrased quote from the movie "Who Framed Roger Rabbit."

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.
- Custom transformations that use the VBA functions and user-defined functions.
- 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:

$$f_1(X1)+f_2(X2)/(f_3(X3)+f_4(X1)+f_5(X2)$$

Where f_1 through f_5 are transformations of the different regression variables.

The General Approach to Working with Custom Transformations

The default transformations supported by the VBA code include negative powers (both integer and fractional), the natural logarithm, and positive powers (both integer and fractional). What if you want to add, say a sine and a cosine transformation? What if you want your own user-defined functions? These additions must piggyback on the current power numbering scheme. As an example, let's say that you do not plan to use positive powers beyond 4 (with the power step of 1). Then you can assign the sine and cosine transformations to what would be powers 5 and 6. The values in the *To Power* column (of sheet *Models*) would read 6. Here is what the updated code looks like:

```
Do
    DoEvents
    Col = 1
    For I = 1 To N
      If idx(I) = 0 Then
        s = "LN(" & sVarName(I) & ")"
      ElseIf idx(I) = SCALE_POWER Then
        s = sVarName(I)
      ElseIf idx(I) = -SCALE_POWER Then
        s = "1/" & sVarName(I)
      ElseIf idx(I) = 5 * SCALE_POWER Then
        s = "SIN(" & sVarName(I) & ")"
      ElseIf idx(I) = 6 * SCALE_POWER Then
        s = "COS(" & sVarName(I) & ")"
      ElseIf idx(I) > 0 Then
        s = sVarName(I) & "^" & CStr(idx(I) / SCALE_POWER)
      ElseIf idx(I) < 0 Then
        s = "1/" & sVarName(I) & "^" & CStr(ABS(idx(I)) / SCALE_POWER)
      Else
        s = sVarName(I)
```



```
      End If
      If Len(sTME(I)) = 0 Then
        Sheets(sModelsListSheet).Cells(Row, Col) =
Sheets(sModelsListSheet).Cells(Row, Col) & s
        Col = Col + 1
      Else
        Sheets(sModelsListSheet).Cells(Row, Col) =
Sheets(sModelsListSheet).Cells(Row, Col) & s & sTME(I)
      End If
    Next I
```

Notice that all the new ElseIf clauses must be inserted **after** the If clause and **before** the clause *ElseIf idx(I) > 0 Then*. Notice that the names of the sine and cosine functions can appear in lowercase, uppercase, or mixed case. I do recommend that you define your own constants (such as SIN_SELECT) to replace 5 * SCALE_POWER and other custom scale multipliers. In addition, such constants make your custom code more readable.

You can also use low–value negative powers (such as –5 or –6, or even –1 if you do not plan to use any negative powers) to trigger special transformations. In that case, the code would look like (with using additional global constants for the custom transformations):

```
' Global constants for the custom transformations
Const SIN_SELECT = -500
Const COS_SELECT = -600
. . .
Do
    DoEvents
    Col = 1
    For I = 1 To N
      If idx(I) = 0 Then
        s = "ln(" & sVarName(I) & ")"
      ElseIf idx(I) = SCALE_POWER Then
        s = sVarName(I)
      ElseIf idx(I) = -SCALE_POWER Then
        s = "1/" & sVarName(I)
      ElseIf idx(I) = SIN_SELECT Then
        s = "SIN(" & sVarName(I) & ")"
      ElseIf idx(I) = COS_SELECT Then
        s = "COS(" & sVarName(I) & ")"
      ElseIf idx(I) > 0 Then
        s = sVarName(I) & "^" & CStr(idx(I) / SCALE_POWER)
      ElseIf idx(I) < 0 Then
        s = "1/" & sVarName(I) & "^" & CStr(ABS(idx(I)) / SCALE_POWER)
      Else
        s = sVarName(I)
      End If
      If Len(sTME(I)) = 0 Then
        Sheets(sModelsListSheet).Cells(Row, Col) =
Sheets(sModelsListSheet).Cells(Row, Col) & s
```



```
        Col = Col + 1
      Else
        Sheets(sModelsListSheet).Cells(Row, Col) =
Sheets(sModelsListSheet).Cells(Row, Col) & s & sTME(I)
      End If
    Next I
```

If you want to have multiple sine and cosine terms, the above new ElseIf clause for the trigonometric transformations would be something like:

```
      ' Global constants for the custom transformations
      Const SIN_SELECT = 500
      Const COS_SELECT = 600
      Const SIN2_SELECT = 700
      Const COS2_SELECT = 800
      . . .
      ElseIf idx(I) = SIN_SELECT Then
        s = "SIN(" & sVarName(I) & ")"
      ElseIf idx(I) = SIN2_SELECT Then
        s = "SIN(2*" & sVarName(I) & ")"
      ElseIf idx(I) = COS_SELECT Then
        s = "COS(" & sVarName(I) & ")"
      ElseIf idx(I) = COS2_SELECT Then
        s = "COS(2*" & sVarName(I) & ")"
```

If you are using power increments of, say, 0.5, the above new ElseIf clause for the trigonometric transformations would be something like:

```
      ' Global constants for the custom transformations
      Const SIN_SELECT = 500
      Const COS_SELECT = 550
      . . .
      ElseIf idx(I) = SIN_SELECT Then
        s = "SIN(" & sVarName(I) & ")"
      ElseIf idx(I) = COS_SELECT Then
        s = "COS(" & sVarName(I) & ")"
```

If get a bit more creative and use (1 + sin(x)) and (1 + cos(x)), the above new ElseIf clause for the trigonometric transformations would be something like:

```
      ' Global constants for the custom transformations
      Const SIN_SELECT = 500
      Const COS_SELECT = 550
      . . .
      ElseIf idx(I) = SIN_SELECT Then
        s = "(1 + SIN(" & sVarName(I) & "))"
      ElseIf idx(I) = COS_SELECT Then
        s = "(1 + COS(" & sVarName(I) & "))"
```

The Steps for Setting Up Custom Transformations

There are three types of functions that you can use in custom transformations:

1. Simple VBA functions like Sin, Cos, Tan, and so on. You simply include the names of these functions in the new ElseIf clauses that you add.
2. VBA functions accessed using the WorksheetFunction.*function_name* syntax. You need to first write a wrapper function in a module and then include the names of these functions in the new ElseIf clauses. For example, you need to write the wrapper function Sinh to access WorksheetFunction.Sinh:

```
Function Sinh(ByVal X As Double) As Double
  Sinh = WorksheetFuncion.Sinh(X)
End Function
```

3. Your very own user-defined functions. Declare these functions in Module1 or any other module inserted in the VBA project. You then include the names of these user-defined functions in the new ElseIf clauses.

There is no character-case restriction for the function names used for custom transformations. The steps involved in working with custom transformation functions are:

1. Your user-defined functions and wrapper functions must reside in a module (Module1 would be good). Code your functions in a module. For example, you write code for the user-defined functions MyFx1 and MyFx2.

```
Function MyFx1(ByVal X as double) as Double
  <declarations of variables>
  <statements>
  MyFx1= <expression of X>
End Function

Function MyFx2(ByVal X as double) as Double
  <declarations of variables>
  <statements>

  MyFx2 = <expression of X>
End Function
```

2. Code wrapper functions in a module. For example, to code a wrapper for hyperbolic sine, Sinh, function that is supported by Excel:

```
Function Sinh(ByVal X As Double) As Double
  Sinh = WorksheetFuncion.Sinh(X)
End Function
```

3. Switch to the global declarations in the VBA code of the *Data* sheet
4. You have the option to declare global constants that clearly identify your user-defined transformations. The following sample constants use the powers 3 and

4 to call the user-defined functions MyFx1 and MyFx2. Use power 5 to call the wrapper function Sinh:

```
' Global constants for the custom transformations
Const MYFX1_SELECT = 300
Const MYFX2_SELECT = 400
Const SINH_SELECT = 500
```

5. Locate the For Next loop that writes the transformation expressions for the various terms of the regression models.
6. Insert the ElseIf clauses that generate the required terms. These new ElseIf clauses must be inserted **after** the If clause and **before** the *ElseIf idx(I) > 0 Then* clause:

```
ElseIf idx(I) = MYFX1_SELECT Then
  s = "MyFx1(" & sVarName(I) & ")"
ElseIf idx(I) = MYFX2_SELECT Then
  s = "MyFx2(" & sVarName(I) & ")"
ElseIf idx(I) = SINH_SELECT Then
  s = "Sinh(" & sVarName(I) & ")"
```

The function *Fx* is coded to evaluate all of the functions and regression variables. The function *Fx* makes the distinction between functions and variable names because the latter end with a punctuation character (the $) which function names cannot use. This difference prevents *Fx* from making fatal mix-up substitution of function names (or parts thereof) with values meant for variables. For example, if you have a variable named nh and are using the user-defined function Sinh, then without the VBA code appending the punctuation character to the variable name, the *Fx* function would replace the letters nh in Sinh with the value meant for variable nh. The result is the letters Si followed by some number. This results in string variable sExpress containing a string that cannot be correctly evaluated by function *Evaluate*. This mix-up would not occur when the variable nh is temporarily named nh$ making it distinguishable from Sinh.

The *Fx* function applies function *Evaluate* to the string variable sExpress to obtain the function's result. The appending of a punctuation character to variable names is a temporary measure performed by the VBA code. When the VBA code completes the regression calculations it will prompt you to restore the original names of the regression variables in all of the worksheets by calling subroutine *restoreVarNames*. If a fatal error occurs and the regression calculations come to a halt, you need to click on button 6 of the Main Menu form to invoke subroutine *restoreVarNames*. This subroutine restores the original variable names in the various worksheets.

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 A*n* 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 A*n* 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.

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.

Math Expression Example	*Appears in Row*	*Covers Last Row*	*Comments*
`'=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 A2, plus one, to the variable named in cells A3. Notice that I have enclosed each quasi-variable in pairs of parentheses to ensure that VBA correctly evaluates the expression.

Math Expression Example	*Appears in Row*	*Covers Last Row*	*Comments*
`'=(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 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**.

Math Expression Example	*Appears in Row*	*Covers Last Row*	*Comments*
`'=(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 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.

☛	
	The math expression transformation is a powerful transformation feature. With power comes responsibility. Plan very carefully the values you enter in the TME cells. I strongly recommend you separately run subroutine *BuildSuperRegressionModels* and examine the output in the *ModelsList* sheet. If need be, edit the math expressions in the TME cells until you get correct terms in the *ModelsList* sheet. Once you get correct models, you can then perform the regression calculations

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

"You've got to ask yourself one question: 'Do I feel lucky?' Well, do ya punk?" – Dirty Harry

Using the transformations to linearize variables and cross–product is a powerful tool. You may need to run alternate sets of regression models replacing the multiplication of multiple transformed variables with the ratios of the transformed variables. This shift from multiplication to division may need to happen in stages, with each stage generating a best selection of regression models! When all is said and done you will have a sizable set of good regression models.

Once you obtain the set of best empirical regression models, you may need to look at the detailed regression ANOVA table of some of the models listed in the *Results* sheet. To perform this task, click No to the prompt that asks you if you want to remove the trailing punctuation characters from the variable names. You still need these characters to correctly evaluate the expressions in the various terms of regression models. The process of examining individual regression models in details involves the following steps:

1. Determine the row number in sheet *Results* whose model you want to examine.
2. Launch/access the Main Menu form and click on button 4. This command button executes the subroutine *ExamineARegressionModel*. The subroutine will prompt you to enter the row number for the selected model. Enter the row number of the selected model and click Ok. The subroutine will perform the regression calculations and switch you to sheet *Scratch* to see the results.
3. First examine the values of the adjusted coefficient of determination, the F statistic, and the significance of that statistic. If the value of the latter is above 0.05 then reject the whole model. If not, continue with the next step.
4. Examine the regression coefficients and the p–values. Pay attention to the following:
    a. Regression coefficients that are exceedingly small (much smaller than the others)
    b. Regression coefficients with a confidence interval ranging from negative to positive values.
    c. Regression coefficients that have p–values greater than 0.05.

    Such regression coefficients may refer to terms that can be dropped out of the current model.

5. If the coefficients (and their p–value) look good, then the selected regression model is acceptable. If not, then resume with the next step.
6. If one or more of the coefficients (and their p–value) do not look good, delete the columns of the data in the *Scratch* sheet that yielded the unacceptable regression coefficients.
7. Click on button 5 in the Main Menu form to invoke the subroutine *RedoLastRegressionModel*. This subroutine recalculates the regression ANOVA table for the reduced model. Examine the results.
8. You may need to remove more data columns and rerun subroutine *RedoLastRegressionModel.*
9. **Once you are done with analyzing the data sets**, click on command button 6 in the Main Menu form to remove the trailing punctuation character from all the variable names in the various worksheets.

Parting Words

"To infinity and beyond!"
--Buzz Lightyear from the movie “Toy Story”

The VBA code in this study is not magic (and certainly is not bug–free and not–crash proof)! I have earnestly anticipated handling several types of runtime errors. The program code has evolved to be a somewhat complex system of multiple software components. The code assumes that each regression model adheres to the basic curve fitting assumptions which states that the errors in the (transformed) dependent variable are normally distributed. The code also adheres to the legacy adage “trash in, trash out!”. Remember that error in your data can prove to be a cunning enemy. Errors in your data may favor certain transformations, cross-products, and math expressions that fit the errors and noise generated by these errors. Dividing and testing your error-prone data into multiple sets may prove to be a good remedy, allowing you to detect best recurring regression models.

You **should know your data** and deal with negative values by normalization. You should also normalize data that have wildly swinging magnitudes. When you normalize data, keep record of the maxima and minima values. You will need them when using the model(s) you select to predict new values.

The VBA code in this study is aimed at empirical modeling. The scheme works best with accurately measured/calculated data, such as:

- Well-established properties of materials.
- Accurately measure data for weather, commerce, transportation, and so on.
- Probability distribution curves.
- Math functions, especially ones that require the evaluation of multi–term series or integrals.

In such a case, you simply seek the best model.

In the case of working with research data, you need to run the VBA code with several data batches (especially if you have a large number of observations) and detect the model that shows some prominence in all or most of these batches. In this kind of study, the absolute best model, that appears just once, is not always the true winner.

Appendix A – Source of Math-Related Runtime Errors

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

Case	*Arguments*	*Comments*
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'll be Back" --The Terminator

Version	*Release Date*	*Comments*
1.0.0	July 17, 2021	Initial release.

