JMP Sales and Marketing
SPRING 2001
ISSUE 8

JMPer Cable®

NEWSLETTER FOR JMP® USERS

IN THIS ISSUE

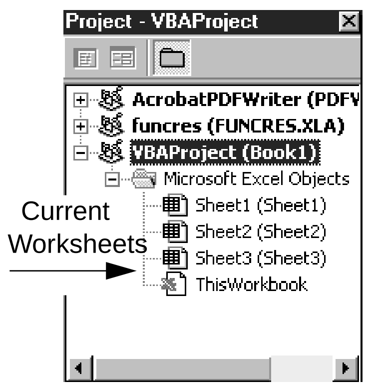
AUTOMATING JMP FROM EXCEL® 2000

Brian Corcoran
JMP Development

One of the priorities for JMP 4 was to provide a better interface with other programs. On Windows, automation is a good way to do this.The following example automates JMP using a macro within an Excel 2000 worksheet. The macro code, shown later, is written in Visual Basic:

1) When the Excel worksheet is initially opened, JMP is visible.
2) The Excel worksheet is then imported into JMP using the ODBC automation interface.
3) Once the worksheet data is in JMP, all changes to individual worksheet cells are sent to JMP and updated in the JMP data table.
4) The first time a row value in Excel changes, JMP generates a Control Chart.
5) Subsequent changes to the Excel worksheet result in updates to the Control Chart. This is because Control Chart output is dynamically linked to the JMP data table, which is dynamically updated by Excel.
6) Every fifth time the Excel worksheet changes, a method is called in JMP to generate a PNG file for the Control Chart. This allows users without JMP to view the output through a web browser.
7) When the Excel worksheet closes, JMP shuts down through automation.

Begin by opening Microsoft Excel. To create a Visual Basic script for an Excel workbook, select **Tools→Macro→Visual Basic Editor** from the Excel menu bar. The Visual Basic editor opens in a separate window. On the left of the Visual basic editor, there is a pane entitled **VBA Project** (shown below), which shows the workbook and the sheets that may have Visual Basic code associated with them. Code written for the workbook usually works for any of the sheets within the workbook.



There are four sections of code involved in the coding for this example. These sections (called **Snippit 1**, **Snippit 2**, **Snippit 3**, and **Snippit 4**) are described next, and shown at the end of this article.

Snippit 1

First, variables that are global in scope are declared in the file called `module1.bas`. These variables can then be referenced in other code modules. To insert a module into the Visual Basic project, right-click in the VBA project icon and select **Insert→Module**. Type or Paste the Snippet 1 code into the module. The Snippit 1 code declares instances of a JMP application, a JMP data table, and a flag to keep track of whether a document is open or not.

Snippit 2

The Snippet 2 code updates JMP when cells in the Excel worksheet change. It is called automatically because Excel generates the `Worksheet_Change` event whenever a cell is changed, deleted, or added.

The Excel VBA Project Browser shows the sheets that are currently part of the workbook. The code snippet should be placed in the sheet that sends data to JMP. Double-click on the sheet icon in the VBA Project Window to bring up the code for that particular sheet.

This code first checks to make sure JMP has a data table open. If the change is happening to the first row, then it is ignored because this is the column header in JMP. So, if a column name is changed in Excel, the corresponding change is not reflected in JMP. Code that would deal with heading changes could be inserted here, but is omitted in this article.

(continued next page).

(continued from page 1)

Next, if the row that has changed is beyond the number of rows that JMP is currently tracking in the data table, then the `AddRows` method is called to create more rows. Finally, if the operation is on a single value and doesn't appear to signal a deletion, the JMP data table cell value is changed to the value that is passed into `Worksheet_Change`.

Snippit 3

This is the main module associated with the workbook. In the VBA Project Browser, the workbook code area is typically assigned the name `ThisWorkbook`. This name can be easily changed. The code of Snippet 3 goes into this area.

The `Workbook_Open` subroutine is called when the workbook is opened in Excel. It tells JMP to open (through ODBC) that same currently loaded Excel file.

The `Workbook_Change` event is generated every time a user changes the data in any cell in any worksheet in the workbook. This sample assumes that there is only one active worksheet in the workbook. The first time that the user changes a cell value in the worksheet, the `Workbook_Change` subroutine creates a Control Chart in JMP using the current data table.

In this sample, the `Workbook_Change` subroutine also creates a PNG graphic file of the Control Chart output and updates it on the disk every fifth time a change is made to the workbook. This just gives some ideas on how Excel events and JMP automation can be used together to create output.

Finally, the `Workbook_BeforeClose` subroutine is invoked when the Excel workbook is closed, but before the window goes away. The code within this subroutine instructs JMP to close down as well.

This example is good if the only activities that occur with the data are additions or changes. The `Worksheet_Change` event in Excel is very limited in the reporting it provides. For example, it doesn't give the type of action (deletion, change, drop) that caused the event. Also, the Excel documentation is incomplete. For instance, it says deletion doesn't cause an event, but it actually does in practice. These problems make it hard to do cell-by-cell updating of a JMP data table in instances where deletion, drag/drop, or block replication need support.

Snippit 4

If there are problem cases, it is probably better to rely on a brute-force approach. One way is to reload the data into JMP every time a certain number of changes occur. An example is shown at the end of this article in Snippet 4.

This sample reloads the data every time there are 10 changes to the Excel Workbook. First, it removes JMP Control Charts and data tables that were previously created. Next, it loads the new data and creates a Control Chart.

This sample works best for small amounts of data. If huge Excel files are involved, this approach isn't efficient because of reloading the table into JMP. These limitations stay in our mind during the JMP development process, and hopefully the Excel event support will be enhanced in the future, making these types of dynamic changes easier.

Snippet 1

```
Public MyJMP as JMP.Application‘The JMP Application Object
Public DT As JMP.DataTable    ‘The JMP Data Table object
Public DocOpen as Boolean     ‘A flag indicating “JMP Table Open”
```

Snippet 2

```
Private Sub Worksheet_change(ByVal Target as Range)
     Dim Col as JMP.Column
     If(DocOpen) Then
           If(Target.Row = 1) Then
               Return
           End If
           If(DT.NumberRows < Target.Row - 1) Then
               DT.AddRows Target.Row - DT.NumberRows, 0
           End If
           If(Not IsArray(Target.Value) And Not IsEmpty(Target.Value)) Then
               Set Col = DT.GetColumnByIndex(Target.Column)
               Col.SetCellVal Target.Row - 1, Target.Value
           End If
     End If
End Sub
```

Snippit 3

```
'Public(Global Variables) that all Workbook subroutines may access
Public Counter As Integer'counter to update Control Chart every 5 changes
Public JMPDoc As JMP.Document        'instance of JMP Document
Public CChart As JMP.ControlChart    'instance of Control Chart
Public ChartOpen as Boolean          'Flag to set if chart is open
'Shut Down JMP before closing the workbook
Private Sub Workbook_BeforeClose(Cancel as Boolean)
    DocOpen = False
    MyJMP.Quit
End Sub
'As soon as the workbook is opened via File Open, load JMP for Automation
Private Sub Workbook_Open()
    Set MyJMP = CreateObject("JMP.Application")'Create an instance of JMP
MyJMP.Visible=True 'Make this instance of JMP visible
    Counter = 0'initialize counter that counts changes
    DocOpen = False'no document open yet
    ChartOpen = False'no charts open yet, either
Set JMPDoc = MyJMP.OpenDocument("C:\BOOK1.XLS")'CHANGE THIS PATH TO POINT TO THE EXCEL WORKSHEET
    Set DT = JMPDoc.GetDataTable'Create data table named DT
    DocOpen = True 'Set flag to say document is open
End Sub
'This is the most important part. After the first piece of data has been changed, generate a 'control
chart. After every 5 changes to Excel worksheet cells, generate a new PNG of the Control Chart.
Private Sub Workbook_SheetChange(ByVal Sh As Object, ByVal Source As Range)
         Counter = Counter + 1
         'Save the control chart to a PNG every time 5 elements get updated
         If (Counter Mod 5 = 0 Or Counter = 1) Then
         'If the Control Chart hasn't been created yet, do so
            If Not (ChartOpen) Then
                Set CChart = JMPDoc.CreateControlChart'create chart
                CChart.LaunchAddProcess "Column 1"'Add column
                CChart.LaunchAddSampleUnitSize 5
                CChart.LaunchSetChartType jmpControlChartVar
                CChart.Launch'launch the chart
                ChartOpen=True 'set flag to remember that a chart is open
            End If
         End If
End Sub
```

Snippet 4

```
Private Sub Workbook_SheetChange(ByVal Sh as Object, ByVal Source as Range)
         Counter = Counter + 1
         If (Counter Mod 10 = 0) Then
            'If there is a previous chart of Table opened, close it first
            If(DocOpen) Then
                JMPDoc.Close False, ""
                CChart.CloseWindow
            End If
            Set JMPDoc - MyJMP.OpenDocument(InstallDir + "C:\BOOK1.XLS")
            Set DT = JMPDoc.GetDataTable
            DocOpen = True
            'Now, create the control chart. This one is keyed to the data in "Column 1". If 5
            'or more values are changed, JMP should generate a new Chart and save it as a
            'PNG file to disk. The PNG file can be viewed with Internet Explorer.
            Set CChart = JMPDoc.CreateControlChart
            CChart.LaunchAddProcess "Column 1"
            CChart.LaunchAddSampleUnitSize 5
            CChart.LaunchSetChartType jmpControlChartVar
            CChart.Launch
            CChart.SaveGraphicsOutputAs "C:\ControlChart.png", jmpPNG
         End If
End Sub
```

OPTIMIZING WITH A FIXED VARIABLE

Bradley Jones and Ann Lehman
JMP Development

The factor profiler is primarily a tool for visualizing the effects of many factors on one or more responses. By using desirability functions with the profiler you can also find the factor settings that optimize the response(s) over the range of the factors.

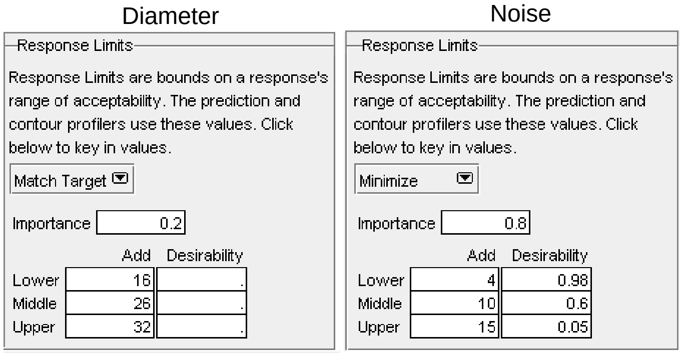
Suppose you have several machines manufacturing optical fibers whose diameter must be between 20 and 35 microns, and must also transmit minimal noise. For each machine (M1, M2, and M3), you want to find values of a control setting and time that give the best results.

	Machine	Setting	Time	Diameter	Noise
1	M1	1	10	45.7	11.4
2	M1	1	20	48.8	8.1
3	M1	2	10	22.3	8.0
4	M1	2	20	25.3	4.2
5	M2	1	10	22.8	12.1
6	M2	1	20	22.0	11.9
7	M2	2	10	28.2	6.0
8	M2	2	20	26.6	4.8
9	M3	1	10	32.3	5.2
10	M3	1	20	34.7	7.1
11	M3	2	10	24.7	1.6
12	M3	2	20	26.1	2.8

The data are shown here. When the response data (Diameter and Noise) were entered into the JMP table, information about these column was stored with the table. Storing columns properties is done using the Response Limits dialogs shown in **Figure A**. To assign or change the response characteristics of a column, select **Response Limits** from the **Properties** menu on the Column Info dialog.

Figure A Characteristics of Responses

Diameter Noise

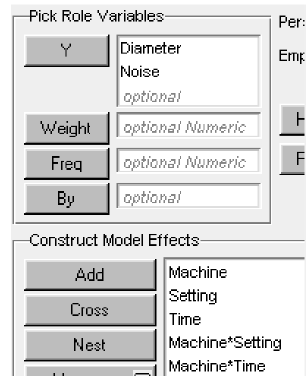


For Diameter, the lower, middle, and upper limits, 16, 26, and 32, were entered into this dialog along with the **Match Target** goal and a relative importance value of 0.2. The goal is to **Minimize** noise. The importance of the Noise factor is 0.8. Limits are 4, 10, and 15, with desirabilities entered as 0.98, 0.6, and 0.05.

- Importance is the relative weight of the response to be considered when searching for the best factor settings. In this example, Noise has 4 times more weight than Diameter.
- The Desirability values, shown for the Noise response, are the points on the desirability function for the given response limits.

Unconstrained Optimum

Fit Model Dialog Specification

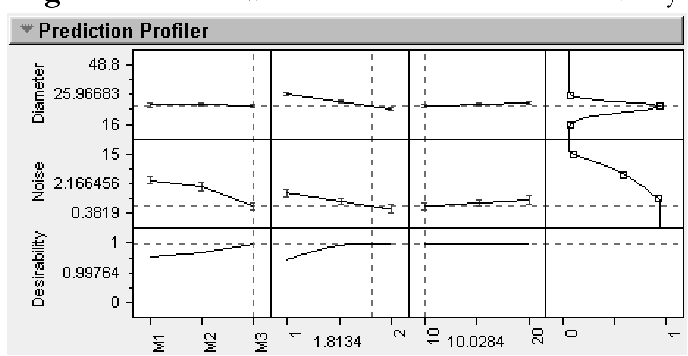


The next step is to fit a model, specifying the effects as shown here. This analysis finds an overall optimum given the properties of the responses as stored in the data table.

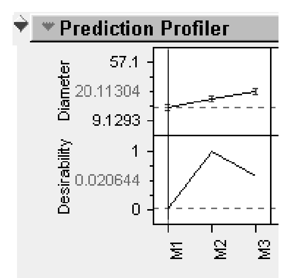
To see the Prediction Profiler in **Figure B**, run the model and select **Profiler** from the menu on any title bar. Then select **Desirability Functions** and **Maximum Desirability** from the menu on the Prediction Profiler title bar. You can see that the overall optimum response occurs for the M3 machine. The desirability at these settings is 0.99764.

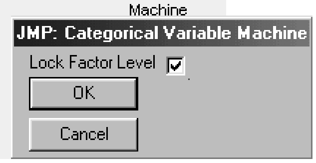
Finding the overall optimum is good, but it only shows where to operate one machine. You can use the profiler to hold each machine constant and find optimum setting again, but for that specific machine.

Figure B Profile with Maximum Overall Desirability



Fixing a Factor Setting





First, drag the vertical line to machine M1. To fix the Machine factor setting, *alt*-Click (*option*-click on the Mac) on the Machine axis and check Lock Factor Setting in the dialog that appears, as shown here. Notice that the vertical line through M1 becomes solid. The

(continued next page)

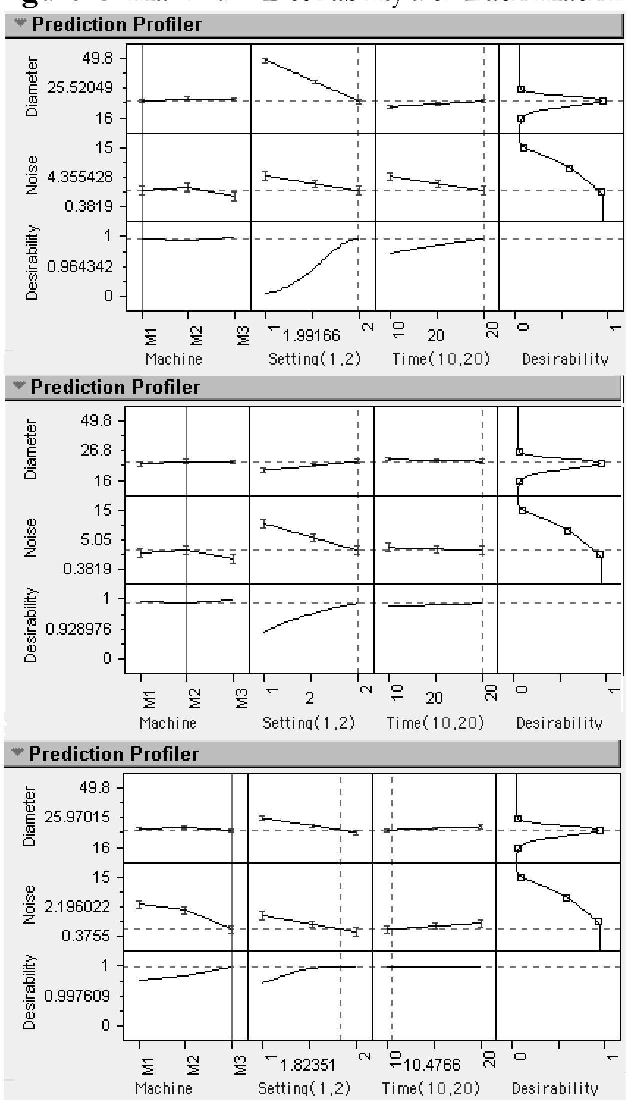
(continued from previous page)

profiler at the top in **Figure C** shows settings that give the optimal response for machine 1 (M1).

Drag the vertical line to machine 2 (M2). Choose **Maximize Desirability** again from the triangle menu on the Prediction Profiler title bar. The middle profiler in **Figure C** shows the optimum response for machine 2 (M2). Repeating these steps for M3 yields the profiler at the bottom in **Figure C**.

In each of the profilers the optimal desirability is close to 1. Each machine is capable of making parts very close to the target. To do so, you need to adjust Setting and Time on a machine-by-machine basis. M1 and M2 perform best when Time is set at 20 and Setting is close to 2. M3 performs best when Time is lowered to 10.4766 and Setting is 1.8235.

Figure C Maximum Desirability For Each Machine



WHAT PEOPLE ARE SAYING...

"I like the excellent capabilities and ease of use that JMP offers. From a wide variety of data analysis needs, JMP is useful for my typical continuous data sets to less common categorical data sets. These analyses involve problem-solving for all business functions. The efforts are expected to provide a large impact to the 'bottom line' of our company"

Andy Paquet
Research Scientist and Six sigma Master Black Belt
Dow Chemical

"...There are many things that make me choose JMP. I think the main advantage is the graphical/data connectivity. The ability to select graphs and the corresponding data selected for manipulation is amazing. Outliers are easily identified and marked, hidden, or manipulated into separate tables for analysis"

Bill Meister
Senior Field Applications Engineer
Excellon Automation

"The outstanding benefits of JMP are its visual approach to data and its ease of learning, on top of a full-featured statistical package. Many others force a compromise between functionality and ease of use, but JMP combines both. Our research assistants pick it up easily and say it's fun! When was the last time you heard THAT for statistical work?"

Paul Teplitz
President
Research Boston Corp.

"Other packages were tedious, and results were difficult to interpret without a Ph.D. in statistics. I watched a JMP user for 10 minutes and was off and running. The graphical representation of results is awesome! Traditionally, the use of statistical packages was only for 'experts,' but because of JMP's ease of use, we make it available to all employees and today are reaping the rewards. JMP is still my statistical software of choice

Damon R. Stoddard
Six Sigma Master Black Belt
Honeywell

"I first used JMP to show confidence intervals and 3-D rotating plots of data to my multiple linear regression students. Then I used it for a multivariate analysis class to show essentially the same thing, because a visual representation of data is much easier to comprehend than series of statistics or tables. I highly recommend it to instructors of statistics at all levels."

Paula L. Woehlke
Professor, Educational Psychology and Special Education
Southern Illinois University at Carbondale

CREATING DIALOG BOXES WITH JSL

Lee Creighton
JMP Development

Version 4 of JMP brought a powerful scripting language, allowing users to extend and customize analyses. In fact, it is possible to create entire new platforms, written generically, that gather information and display results using professional-looking Windows and Macintosh components.

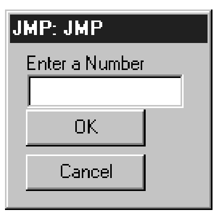
JMP's scripting language (JSL) uses dialog boxes to gather information from a user. JSL provides functions that create dialog boxes and populate them with common controls, such as radio buttons, text boxes, and drop-down lists.

Create a Dialog

Beginning JSL programmers often find dialog boxes challenging to implement. The most common problem (and the one this article tries to address) is getting results out of dialog boxes after the user enters a value or makes a selection.

For example, consider the dialog box generated by the following code snippet. Enter and execute the code example, but make sure the log window is visible so the results of the dialog box actions can be seen after execution. If the log is not showing, select **View→Log** (**Window→Log** on the Macintosh).

```
dlg=Dialog(
    "Enter a Number",
    newvar=EditNumber(),
    Button("OK"),
    Button("Cancel")
);
```



The result of this code is the dialog shown here. Enter a number in the edit box (use 3.14 for this example), click **OK** and note the output in the log:

```
{newvar = 3.14, Button(1)}
```

The results are enclosed in curly braces {}, which means that its contents form a *list*. A list is any collection of items within a set of curly brackets, separated by commas. The first element in this list is '`newvar = 3.14`' and the second element is '`Button(1)`'. If the user clicks **Cancel** instead of **OK**, the result is

```
(newvar = ., Button(-1))
```

Capture the Dialog Entry

The JSL code created the dialog and accepted input but nothing has been evaluated yet. Specifically, the value of `newvar` has not yet been assigned the value 3.14. To actually do the assignment, the elements must be unpacked from the list. Note that the second item in the list, `Button(1)`, is just the result of clicking **OK** on the dialog; nothing needs to be evaluated for this list item.

In order to extract a list item, the entire `Dialog` function was assigned to a variable name:

```
dlg=
```

This assigns the list itself to be the values of `dlg`. Then, to access each individual list member, attach a subscript in square brackets to the variable name `dlg`. For example, `dlg[1]` gives '`newvar = 3.14`' and `dlg[2]` gives '`Button(1)`'.

Next, to capture the dialog entry, the assignment in `dlg[1]` must be evaluated. One way to accomplish this is to enclose `dlg[1]` inside an `eval()` function. So, the complete snippet to draw a dialog box and unpack the results of the user entry is

```
dlg=Dialog(
    "Enter a Number",
    newvar=EditNumber(),
    Button("OK"),
    Button("Cancel")
);
eval(dlg[1]);
```

The final statement evaluates the first element of the list (`newvar = 3.14`), which assigns the value entered into the dialog to the variable called `newvar`. The variable `newvar` can now be used in subsequent JSL statements.

Alternative Methods to Capture a Dialog Entry

There are alternatives to this method of unpacking a list. JSL has a function to evaluate an entire list of assignments, which can be useful if there are many elements in the dialog box results list. In that case, the use of one `eval()` for each element can make a script larger than it needs to be. Instead, a single `evallist()` can be used.

A problem with `evallist()` is in the last element of the dialog box results list. Typically, this result is `Button(1)`, a command that is not understood by the JSL parser. So, simply evaluating the list from a dialog using `evallist()` gives an error. To see this error, replace `eval(dlg[1])` in the above script with `evallist(dlg)`.

One way around this problem is to always remove the last element of the list from the results of the dialog box before evaluating the list with `evallist()`. This is quite easy because the function to remove an element from a list, `remove from()`, removes the final list item

(continued next page)

(continued from previous page)

by default. Using this technique, the complete example script would be:

```
dlg=Dialog(
  "Enter a Number",
  variable=EditNumber(),
  Button("OK"),
  Button("Cancel")
);
removefrom(dlg);
evalist(dlg[1]);
```

A third alternative is to dig the user input out of the list by specifying a variable name with an equal (=) on the left of each element in the list.In this example, `dlg["newvar"]` has the value 3.14. The expression

```
extract=dlg["newvar"]
```

extracts the user input associated with the variable `newvar` in the results lists and assigns it to `extract`. This method is particularly useful if the results need to be assigned to a variable other than the one named in the dialog box code.

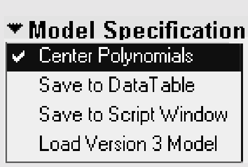
? ?

A Frequently Asked Question

Q: Why is polynomial centering the default in the Fit Model platform? It makes the answers different than in JMP Version 3.

A: Because if you don't use it, the parameters on lower-order terms are less meaningful. This is perhaps most easily seen if you introduce a location shift in a regressor. Suppose you changed from Celsius to Fahrenheit in a regressor called temperature that you use to fit a second-degree polynomial. With polynomial centering, all your tests would be unaffected by the change in location and scale. If you didn't use polynomial centering, then the test for the higher-order effect would be unaffected by a change in location, but the test for the lower order effect, temperature, would change. The benefit of centered polynomials is that they make the test for the main effect independent of the test for the squared term.

Note: If you don't want polynomial centering, use the menu on the Fit Model dialog title bar to turn it off.



Summation Quiz

The **Summation** function sometimes causes confusion because it behaves differently depending on whether or not you use a subscript on its argument.

- If there is no subscript on the summation argument, the **Summation** function sums across its argument(s) the number of times indicated by its indices.
- If there is a subscript on the argument, the **Summation** function sums down the rows of its arguments as indicated by its indices.

For example, each of the following formulas produces a different result. *Which one gives a cumulative sum of the variable called count?*

$\sum_{i=1}^{NRow()} count$ $\sum_{i=1}^{NRow()} count_i$ $\sum_{i=1}^{Row()} count$ $\sum_{i=1}^{Row()} count_i$

The results of these four formulas for a variable called count, with values one to five, are shown in **Figure A**.

The first computed column, called NRow( ) summation no subscript, sums the values of count without a subscript. Since there is only one argument (count), its value is summed **NRow** times. **NRow** is 5, so the result is the same as would be given by the formula, **NRow()*** count, or 5* count in this example.

The second computed column, called NRow( ) summation with subscript, uses the Summation index, i, and sums down the count column for each row, giving the constant value 15.

The third formula behaves the same as the first formula but sums each value the number of times equal to the row number. It is equivalent to the formula **Row()** * count.

The fourth computed column sums from 1 to the current row for each row; it produces the cumulative sum of count.

Figure A Examples of Summation Functions

count	Nrow() summation no subscript	Nrow() summation with subscript	Row() summation no subscript	Row() summation with subscript
1	5	15	1	1
2	10	15	4	3
3	15	15	9	6
4	20	15	16	10
5	25	15	25	15

COMPLEX SPLITS

Lee Creighton
JMP Development

Rearranging a data table is an everyday necessity in the world of statistics. Often, data are entered into a table before an analysis is well-defined. A typical example occurs when there is a matched-pairs situation but responses are entered into a single column as shown in the table here.

	Wash Method	Lot Number	Sand Blast	Starch Content
1	CS	1	measure	34.0
2	CS	2	measure	26.8
3	CS	3	measure	41.3
4	CS	4	measure	36.3
5	CS	5	measure	18.2
6	CS	1	repeat	35.4
7	CS	2	repeat	28.8
8	CS	3	repeat	35.1
9	CS	4	repeat	22.4
10	PS	1	measure	30.0
11	PS	2	measure	32.4
12	PS	3	measure	28.0
13	PS	4	measure	29.0
14	PS	1	repeat	18.4
15	PS	2	repeat	19.4
16	PS	3	repeat	28.3
17	PS	4	repeat	13.9
18	PS	5	repeat	22.4

This file has information from an experiment on blue jeans. When jeans are manufactured, they contain starch that adds stiffness to the fabric. Often, companies wash the jeans to remove some of the stiffness. In addition, jeans are sometimes sand-blasted. The data show the starch content that results in several lots of jeans (Lot number) that were initially washed by one of two methods (Wash Method) with values CS and PS. The purpose of the experiment is to determine whether sand blasting jeans after washing removes a significant amount of starch. For each combination of wash method and lot number, the percent starch content was measured before sand blasting ("measure") and again after ("repeat").

The best way to compare before-and-after measurements is usually a paired t-test. However, to use the Matched Pairs platform in JMP, the response (Starch Content) must be arranged in two columns that will be called measure and repeat in the new table, according to the Sand Blast variable.

A Simple Mistake

At first glance, this might look like a simple task for the **Split** command in the **Tables** menu, where the Split variable is Starch Content and the Col ID variable is Sand Blast. However, a problem occurs because the data are unbalanced—there are not complete before/after pairs for all combination of lot number and wash method. A simple split produces the incorrect matching.The values of Starch Content for the first alphabetical value of Sand Blast ("measure") form the new column called measure. The split process then gathers values for the next alphabetic value of Sand Blast ("repeat") into a second new column and joins the new columns without regard for the values of any other variable. The result is the incorrect pairing shown here; the value for lot 5 of wash method CS is paired with lot 1 of wash method PS.

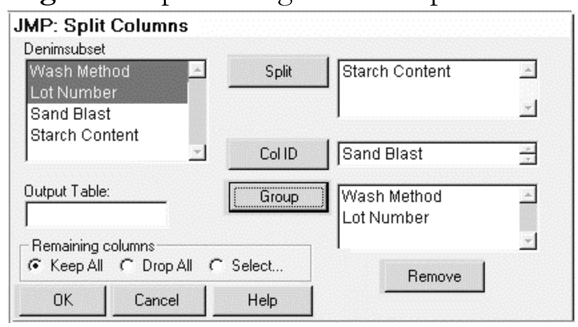
Incorrect Matching

	Wash Method	Lot Number	measure	repeat
1	CS	1	34.0	35.4
2	CS	2	26.8	28.8
3	CS	3	41.3	35.1
4	CS	4	36.3	22.4
5	CS	5	18.2	18.4
6	PS	1	30.0	19.4
7	PS	2	32.4	28.3
8	PS	3	28.0	13.9
9	PS	4	29.0	22.4

A Column Split with Group Variables

To produce a table with the correct pairs in two columns in this example, any variable for which there is a missing part of a pair must be identified as a Group variable (**Figure A**)

Figure A Split Dialog with Group Variables



The split process assembles two new columns as before. But now, joining the columns to produce the new table is done by matching the Group Variables' values associated with each row, which gives the table shown here, with missing values where there are incomplete pairs. Wash method PS does not have a measure value for lot 5, and wash method CS has no repeat value for lot 5.

Correct Matching

	Wash Method	Lot Number	measure	repeat
1	CS	1	34.0	35.4
2	CS	2	26.8	28.8
3	CS	3	41.3	35.1
4	CS	4	36.3	22.4
5	CS	5	18.2	•
6	PS	1	30.0	18.4
7	PS	2	32.4	19.4
8	PS	3	28.0	28.3
9	PS	4	29.0	13.9
10	PS	5	•	22.4

Note: You might wonder why the values of Sand Blast are "measure" and "repeat" instead of the more obvious "before" and "after". The **Split** command in JMP processes the ID values in alphabetic order and arranges the columns that way in the new table—so (from left to right) 'before' shows after 'after'. Besides, we were trying to make a point about 'repeat'(ed) 'measure'(s).

SATURATION CURVES

Annie Dudley Zangi
JMP Development

In the development phase of drug research, saturation curves are often used to examine characteristics of a compound. For example, saturation curves can track rate of uptake, time, and chemical behaviors of a compound dissolving in the mouth or the stomach, with and without certain foods.

One commonly used curve is,

(*Vmax***Saturation*)/(*km* + *Saturation*), where

Saturation represents the current saturation level of the solution.

Vmax is a parameter representing the maximum flow rate that this particular combination of solution and compound exhibit.

km is a parameter representing the constant saturation point for this solution, used for comparing compounds.

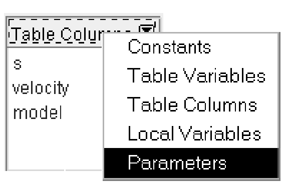
Consider adding sugar to iced tea. As you continue to add sugar the rate that the sugar dissolves slowly decreases until it no longer dissolves at all. We would expect *Vmax* (the maximum flow rate) to occur at the lowest saturation.

However, many solutions involving acids have the opposite effect. As more of a compound is added, the rate of uptake increases.

Figure A shows example data for a saturation study where s is the current saturation level of the solution and velocity (the response) is the resulting flow rate or rate that the compound is dissolving. To estimate the parameters *Vmax* and *km*, we first need to build the nonlinear formula in a data table column that is a function of s. Then use the Nonlinear Fit platform.

We use the formula editor to create the nonlinear formula as follows:

- Create a new column (called model in this example).
- Select **Formula** from the **New Property** menu in the Column Info dialog, and click **Edit Formula** when it shows on the New Property dialog.
- When the Formula Editor appears, create two new parameters called *Vmax* and *km* by choosing **Parameter** from the menu at the upper-left of the Formula Editor.



- Using the list of Table Columns and the list of Parameters, build the formula shown to the right of the table in **Figure A**.

Figure A Saturation Study Data With Model Column

	s	velocity	model
1	0.02	1	0
2	0.05	2.2	0
3	0.1	4.3	0
4	0.5	20	0
5	1	34	0
6	2	50	0
7	4	60	0
8	8	70	0
9	16	71	0

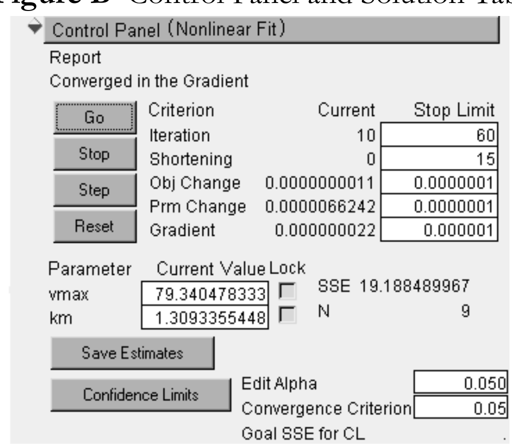
$$\frac{vmax}{km + s} * s$$

Use the Formula Editor to build the saturation model formula. Set initial parameter values to zero .

To run the model, choose **Analyze→Nonlinear Fit**. In the role assignment dialog select velocity as Y and model as X. Click **OK** to see the Nonlinear Fitting Control Panel. Initially the values for the parameters are zero, which works in this example. For other models you may need to edit the current parameter values to specify different starting values.

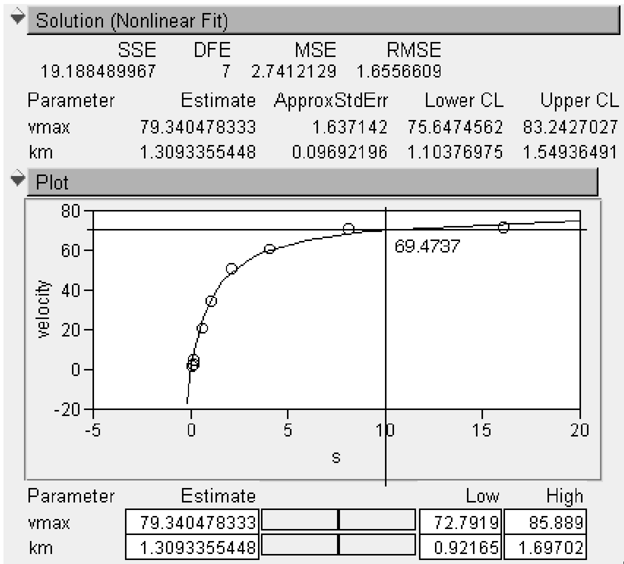
Click **Go** to start the analysis. You can watch the nonlinear fitting process iterate until it finds solutions for the parameters that minimize the SSE. Results appear in the Control Panel and also in the Solutions table as shown in **Figure B**.

Figure B Control Panel and Solution Table



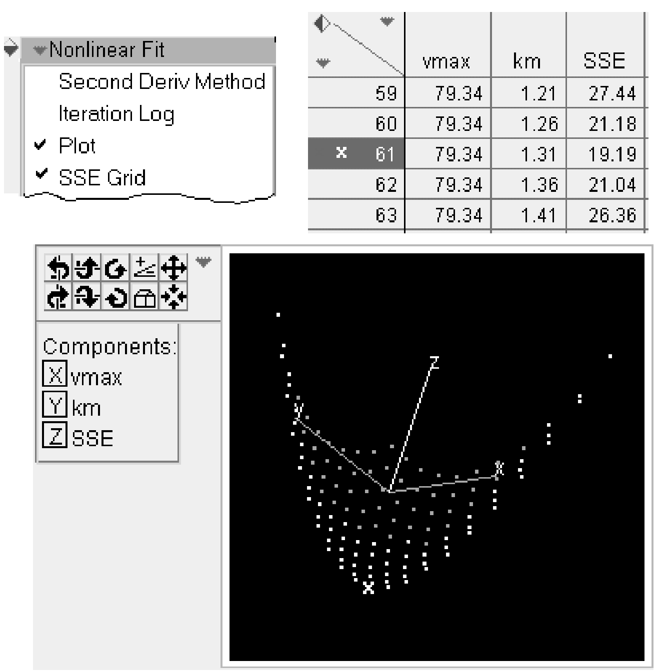
When you have only one X and one Y, as in this example, the Nonlinear Platform displays a plot showing the shape of the curve. There are sliders beneath the plot for each parameter. These sliders can be used to adjust the fit of the curve. You might want to use the sliders to experiment with the fit when there appear to be points that are outliers. You can use the crosshair tool to estimate predicted values from the saturation curve as shown in the plot in **Figure C**.

Figure C Plotted Values of Fitted Response



Also, when there are only two parameters, the SSE Grid option in the popup menu on the Nonlinear Fit title bar constructs a grid of parameter values, computes the SSE for each grid point, and saves the results in a new data table. The grid includes the solution found by the nonlinear fit process, which is selected in the table and indicated by an ‘x’ marker. **Figure D** shows how the Spinning Plot platform can be used to display the parameter surface and solution.

Figure D Plotted Values of Fitted Response



ASSESSING THE NUMERICAL ACCURACY OF JMP

Lee Creighton and Jianfeng Ding
JMP Development

There are many sources of error in statistical computation, including rounding error, truncation error, and the finite-precision inaccuracy involved in representing a number in binary form.

To measure these sources of error, the numerical accuracy of JMP has been compared to standards set by the National Institute of Standards and Technology (NIST). There are 58 data sets created by NIST, called the Statistical Reference Data Sets (StRD). The data sets fall into four categories: Univariate Statistics, Analysis of Variance, Linear Regression, and Nonlinear Regression. These data sets are available at

`www.nist.gov/itl/div898/strd`

Each set comes with a documented source, an explanation that includes a level of difficulty, and data in ASCII format. Included in the data set are certified values for the parameters that are accurate to 15 decimal places.

JMP analyzed each of the data sets, and the number of significant digits given by JMP calculations was recorded. The significant digits computed by JMP were then compared to the StRD standard results.

A frequently-used measure of the number of correct significant digits is the common logarithm of the relative error (LRE), calculated as

$$\mathrm{LRE} = -\log\frac{|q-c|}{|c|}$$

where q is the value given by JMP and c is the correct value as provided by NIST. When $q = c$, this quantity is not defined and the LRE is given the value of the number of significant digits in c. Also, if the certified value is zero then $\mathrm{LRE} = -\log|q|$.

An LRE of 15 means the reported JMP results perfectly agree with the StRD results. Whenever q differs from c by a factor of 2, the LRE is set to zero.

The complete report of the numerical accuracy evaluation of JMP, based on the LRE as described here, can be found at

`www.jmpdiscovery.com`

in the section **Our Quality Statement**. This report also includes graphs that compare the LRE computations for JMP with those for SAS 6.12 and Excel® 97.

JMP EDUCATION

Education Division
SAS Institute

Public or Onsite Training

Let the SAS Education Division guide you through the unique capabilities of JMP at your location or at one of our training centers across the United States, whether you are a novice or expert JMP user. Courses taught by SAS training specialists combine lectures, software demonstrations and hands-on computer workshops utilizing the latest JMP technology.

JMP Software: New Features and Enhancements in Version 4—shows how to navigate the JMP interface and use new features. This course is designed for experienced Version 3 JMP users who want to learn about the new features and enhancements in JMP Version 4.
1 day, $325

JMP Software: Statistical Data Exploration—builds the necessary background in JMP and statistics for anyone who wants to manage data, analyze data, and perform data exploration. Students participate in interactive demonstrations of JMP software in realistic scenarios.
1 day, $325

JMP Software: ANOVA and Regression—for analysts and researchers with some statistical training who want to analyze continuous response data using analysis of variance and regression methods. Topics include basic statistical concepts, multiple regression, N-way analysis of variance, and analysis of covariance.
2 days, $650

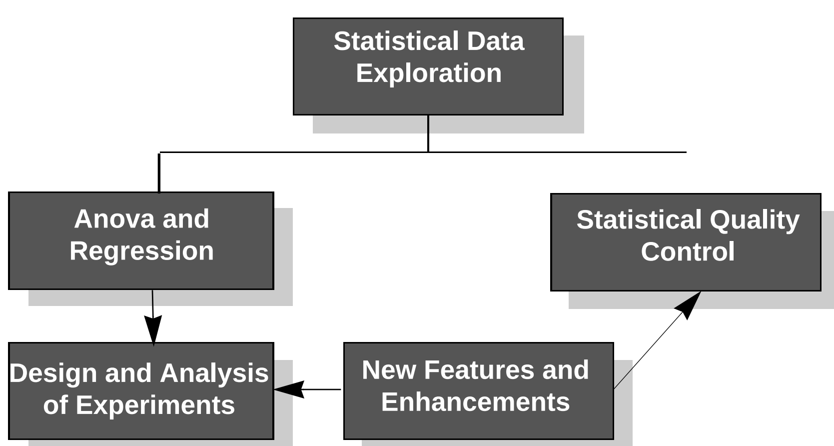
JMP Software: Statistical Quality Control —introduces the principles of quality control using control charts to detect process signals, shows how to respond to types of process variation, covers how to determine measurement error through Gage R&R analysis, capability analysis and the proper use of Pareto charts.
1 day, $325

JMP Software: Design and Analysis of Experiments—teaches how to design and analyze experiments to find the vital few factors or to optimize the process response, with single or multiple continuous responses. Both classical designs and newer custom design approaches to design using realistic examples are covered.
2 days, $650

For More Information or To Register:

- Call 1.800.333.7660 to register for a course or to speak with a public course consultant.
- For onsite training, call 919.531.7312
- E-mail: training@sas.com
- Visit us on the Web at
  `www.sas.com/training`
  for information about course dates and locations.

JMP® 4 Software Curriculum



Coming soon! New courses on:

- Time Series Analysis
- JMP Scripting Language
- Reliability

ISSUE 8 SPRING 2001

EDITOR
Ann Lehman

CONTRIBUTORS
Brian Corcoran
Lee Creighton
Jianfeng Ding
Ann Lehman
John Sall
Annie Dudley Zangi

PRINTING
SAS Print Center

JMPer Cable is sent only to JMP users who are registered with SAS Institute. If you know of JMP users who are not registered, pass them a copy of JMPer Cable and let them see what they are missing!

If you have questions or comments about JMPer Cable, or want to order more copies, write to

JMPer Cable
SAS Campus Drive
Cary, NC 27513

For more information on JMP, or to order a copy, contact JMP Sales

phone: 919-677-8000 x 5071
FAX: 919-677-8224

FLASH

JMPer Cable is on the Web

You can now see JMPer Cable and download data tables used in articles.

Go to the JMP Web site:

`<http://www.jmpdiscovery.com>`

If you don't keep JMPer Cable for reference, please recycle it!

To serve your information needs more efficiently in the future, JMP will distribute the JMPer Cable newsletter via e-mail. Please send your e-mail address and contact information to `jmp@sas.com` so you can receive the latest information from JMP, A Business Unit of SAS. Also, we urge you to register your software at our website:

`http://www.jmpdiscovery.com/product/regrequest_form.shtml`

If you have not yet upgraded to the latest version of JMP 4, please call us at 1-800-727-3228 to upgrade.

JMP™ A BUSINESS UNIT OF sas

JMP Sales and Marketing
SAS Campus Drive
Cary, NC 27513 USA
Tel: (919) 677 8000