JMP 7 heralds the first release of the 64-bit version of JMP for the Windows and Linux operating systems. This is exciting because, although a 64-bit application such as JMP may not necessarily run faster than its 32-bit counterpart, the amount of data that JMP can analyze is substantially increased. We believe that this will have a significant impact on our users, many of whom continue to place greater demands on JMP in terms of the sheer size of the data set they expect JMP to handle.

Unlike the SAS System, JMP must load the entire data table into memory to analyze the data. Hence, JMP faces a fundamental constraint in terms of the size of the data table it can analyze—a constraint imposed by the size of the address space defined by the operating system itself.

In a 32-bit operating system, there are $2^{32}$ bits to form the address of a location in memory where a piece of data can be stored; thus, in theory, when JMP runs on a 32-bit operating system, there are $2^{32}$ theoretical addresses to store or access data. You might think that this would be more than enough room for data but some data analysts, particularly those who are using JMP Genomics, are already experiencing the pain of this limitation. Indeed, some of their data tables are so large that 32-bit JMP cannot load the data at all.

However, in a 64-bit operating system, there are $2^{64}$ bits to form an address for a memory location. This constitutes a theoretical limit of 16 exabytes; in terms of data table size, this is roughly two billion billion cells of data. In other words, 64-bit JMP can load a substantially larger data set into memory than its 32-bit predecessor. Of course, the limitation is still a function of how much memory is available. JMP data tables must fit in memory.

The description above is somewhat of a simplification. It is not really true, for example, that an application running on a 32-bit operating system can access all $2^{32}$ theoretical memory locations because the operating system restricts the range of addresses that an application is allowed to access. Some addresses are off limits to all applications because they are reserved for the operating system itself. A similar qualification applies to a 64-bit operating system—not all $2^{64}$ addresses are accessible to an application such as JMP. Nevertheless, in practice a 64-bit operating system offers a tremendous amount of extra addressing space when compared to a 32-bit operating system. This in turn means that you can load more data into the 64-bit version of JMP running on a 64-bit operating system than you ever could before running the 32-bit version of JMP.
The Status of Linux in the Marketplace
Many industry experts predicted that 64-bit operating systems would increase in popularity and that Linux would gain ground on the desktop as more users adopted the 64-bit Linux operating system. However, neither prediction has come true. In fact, the adoption rate of Linux on the desktop has tapered off. One reason that users may have refrained from migrating to Linux is because the applications they’ve come to depend upon still aren’t available—at least, not applications of quality comparable to those found on Windows and the Mac. The same reasoning probably applies to the slow adoption rate of 64-bit computing. As long as a user has relatively small data sets to analyze, there is little urgency to move to a 64-bit platform. But even if the size of the user’s data is overwhelming, there is little point in adopting a 64-bit operating system if the 64-bit versions of the applications needed aren’t there!

Fortunately, if you are someone who faces the challenge of analyzing increasingly large data sets, you can now migrate to a 64-bit Windows or Linux operating system and know that at least one powerful application will be there: the 64-bit version of JMP 7.

A Word about the Macintosh
Although a 64-bit implementation had also been planned for the Macintosh version of JMP 7, our strategy for creating it was based on the assumption that Apple would provide support for migrating 32-bit Carbon-based applications (such as JMP) to its 64-bit Mac OS X. Shortly before the release of JMP 7 (and after considerable effort on the part of our Mac development team to port our Carbon-based application to 64-bit Mac OS X), Apple dropped its support for this migration path. Instead, Apple announced that if you want to create a 64-bit application for Mac OS X, your application must be written to target the Cocoa API. Creating a 64-bit version of JMP 7 for the Mac that targets Cocoa requires a massive rewrite of the host level code within JMP.

However, within the JMP 8 development cycle, we plan to completely rewrite the Macintosh host layer of JMP in Objective-C and to target the Cocoa API, thereby converting JMP into a truly native Cocoa-based application. This will enable us to produce a 64-bit version of JMP for the Macintosh in time for the release of JMP 8.
Innovators Unite at JMP-sponsored Summit

Diana Levey and Ann Lehman, SAS Institute

The inaugural JMP Innovators’ Summit met in Traverse City, Michigan on October 14-16. This unique conference brought together business and academic professionals for two days of idea sharing. A mix of plenary addresses, panel discussions and small-group sessions offered a variety of opportunities for exploring the path to innovation.

The summit began with an outstanding panel of presenters representing diverse disciplines. By design, a variety of industries was represented in both the speaker line-up and the attendee list. Unlike a conventional conference, the Innovators’ Summit was created to facilitate the exchange of ideas and the benchmarking of best practices. This concept of creating an event to talk about common strategies from industry to industry was inspired by Dimitri Mavris, Director of Georgia Tech’s Aerospace Systems Design Lab.

The initial keynote speaker, Michael Schrage from MIT, is a researcher and author of *Serious Play: How the World’s Best Companies Simulate to Innovate* (Harvard Business School Press, 1999). He is one of the world’s leading experts on the economics of innovation and a pioneer in the economic sociology of modeling, simulation and experimentation in organizations.

Schrage is a specialist in *rapid prototyping* and *speedy simulations* to more effectively manage innovation risks and on managing *innocentives*—incentives for innovation.

He challenged the audience by saying “There is nothing more overrated than a good idea! You are not truly innovative just because you innovate new products and processes. It doesn’t matter how many new features,

functionality, and power, you put into something,” he said. “It’s about the value that your customers find in your offerings. Innovation isn’t what innovators do. It’s what customers adopt!”

Schrage’s talk was part of a four-person panel (shown below) on ‘Extending the Reach of Analytic Excellence.’

The second JMP Innovators’ Summit is already in the planning stage for next year. Watch the JMP Web site for more information about upcoming events, and to read more about the presentations given at the summit this year.

www.jmp.com/summit

Opening Session panel members from left to right:

- *Michael Schrage*, Co-director of the MIT Media Labs eMarkets Initiative
- *J. Stuart Hunter*, Professor Emeritus, Princeton University
- *Chris Nachtsheim*, Chair, Operations and Management Sciences, Carlson School of Management, University of Minnesota
- *Dimitri Mavris*, Boeing Professor of Advanced Aerospace Systems Analysis, Guggenheim School of Aerospace Engineering, Georgia Institute of Technology
Unit Tests: Automated JSL Testing

Joseph Morgan and Xan Gregg, SAS Institute

Software development practitioners use the term unit to refer to the smallest testable component of a software product and the phrase unit testing to refer to the process of validating that a unit operates as expected. Over the past ten years, these practitioners have become increasingly interested in automated unit testing and the phrase unit testing framework has come to refer to the mechanism used to facilitate automated unit testing.

Purpose
Software developers, including JSL developers, have a vested interest in ensuring that the software they develop operates as expected. Apart from this obvious motivation, there are two additional reasons:

1. Software evolution: Software evolves over time. Typically, this is because features are added or changed but sometimes software evolves due to refactoring. A properly maintained suite of unit test cases, routinely used to validate the product, can ensure that the software product retains its expected behavior as it evolves. Units that regress (break or stop working) as the product evolves are quickly identified.

2. Executable documentation: Unit test cases document the expected behavior of software products. One of the benefits of automated unit testing is that the test cases act as executable documentation. As a result, as long as the test cases are routinely used to validate the product as it evolves, these test cases are likely to reflect the correct expected behavior of the software product, unlike static written documentation.

In recent years, the chief advocates of unit testing in general, and automated unit testing in particular, have been the eXtreme Programming (XP) community. Partly due to the enthusiasm of this community, unit testing frameworks are now available for every major programming language.

Features You Expect in a Framework
Frameworks should facilitate automated unit testing and so, at a minimum, should have the following features:

1. Easy to use: Not only should the framework make it easy to run tests, it should make the writing of unit tests simple and straightforward. There are several aspects to this:
   a) Expected behavior specification capability: A unit test specifies expected behavior for a given input.

   A framework should allow for this by providing some way to make an assertion about the output of the unit, or its state, given a particular input.

   b) Expected error support: A suite of unit test cases should include cases with invalid inputs as well as those with valid inputs. For invalid inputs, a framework should provide some way to trap the expected error behavior and treat it as a success.

   c) Fuzzy comparison capability: When the output from a unit is numeric, a value that is close enough to the expected value should be considered a success. The framework should therefore provide some way to specify a close enough level of accuracy.

   d) Setup and cleanup support: Often, a unit test requires that some setup operation be done prior to the test and also that the test be followed by some cleanup operation. For example, the unit test may require that the input data for the test be obtained from an external data source, such as a relational database. This is an example of a setup operation. Closing the database connection would be a cleanup operation. The framework should provide an easy mechanism to do these operations.

   e) Unit test management: For anything but trivial software products, the unit test suite will likely consist of thousands of test cases. A framework should provide some way to easily retrieve test cases for execution or for maintenance.

2. Abstract away complexity of automation: A framework should provide the superstructure needed to write automated unit tests but, at the same time, abstract away the complexity of this superstructure from the user. Furthermore, the framework should make it easy to execute a single unit test or a group of unit tests. It should also make it easy for the user/tester to query the repository of unit tests to determine which subset of unit tests, if any, provide coverage for a particular feature.

3. Reporting: A framework should, at a minimum, provide enough details to allow the user to easily identify the test cases that fail and the cause of failure. Also, the user needs to see a summary report with counts of successes and failures, as well as build and version information (if appropriate), plus the date of execution.
Why Unit Testing in JMP?
JSL has evolved into a full-featured scripting language that is increasingly being used to build non-trivial applications. We believe that as this phenomenon continues, JSL programmers will need more sophisticated development tools. The JMP 7 JSL editor is one step towards that end, and we believe that JSL-Unit is another. JSL-Unit refers to a unit testing framework that may be used to test JSL applications as they evolve.

JSL-Unit is completely written in JSL and so, for the JSL programmer, it is readily accessible. In addition, since it is written in JSL, it can easily be extended by JSL programmers. Furthermore, one of its main advantages when compared to third party unit testing frameworks is that it is guaranteed to work on any host for which JMP is available.

JSL-Unit is particularly suited to a JSL development philosophy that views JSL applications as a collection of loosely coupled functions. Such functions may be thought of as units, and so fit naturally into the JSL unit testing framework. Notice, however, that the framework is not limited to validating simple functions that return a simple value. As long as the unit to be tested has a well defined expected output (e.g., a JMP report or a JMP data table) then the framework can be used to validate it.

JSL-Unit Implementation
The architecture is as follows:

Unit Tests  GUI Driver  Report

Unit Tests
Unit tests may be specified as either JSL scripts or JMP data tables. Expected behavior is specified by way of a JSL function defined by the GUI driver. The prototype of this function is:

\[
\text{ut assert( expression, expected value )}
\]

This is the mechanism that test scripts use to make assertions about the behavior of a JSL application (i.e., to define test cases). An individual test case is an invocation of \text{ut assert} where the \text{expression} argument specifies the actual result and the \text{expected value} argument the expected result. A test case is considered a success (or a pass) if the actual and expected results are equal. The test case also passes if the expected and actual values are both missing. Consider the following example:

\[
\text{ut assert( Expr(Sqrt(1.21)), 1.1 );}
\]

For this example, the unit is the JSL internal \text{sqr} function. The expression \text{Sqrt(1.21)} is evaluated and its result compared to 1.1. Since both values are numbers, a fuzzy comparison is done.

The \text{ut assert} function understands JMP data types and so knows when to do fuzzy comparisons and when to do exact comparisons. For fuzzy comparisons, it allows the user to override the default level of accuracy. Furthermore, it allows the user to mark selected test cases as stress test cases. These are often edge test cases that are expected to fail for a variety of reasons (e.g., floating point round off error).

Now consider the following example:

\[
\text{ut assert( Expr(Sqrt()), ut_error );}
\]

The expected value \text{ut_error} is a special expected value that allows JSL-Unit to provide expected error support.

GUI Driver
The GUI driver
- provides the user interface
- defines the superstructure to automate unit tests
- provides the reporting capability
- provides the unit test management mechanism
- defines the \text{ut assert} assertion function.

As a result, the driver is the core of the framework. It makes the framework easy to use and abstracts away the complexity of automation (see Figure 1).

Figure 1  Unit Test Framework GUI
JSL-Unit Usage at SAS

JSL-Unit was developed to automate the testing of JMP itself and has been heavily used by JMP development over the past two years. The repository of unit tests currently contains over 15,000 test cases organized into approximately 300 unit test suites. These test cases validate JMP internal functions and JMP platforms. The entire suite of test cases is run on each host (Linux, MacOS, and Windows) for each new build of JMP.

Example: Geometric Mean

The geometric mean of a data set \(a_1, a_2, ..., a_n\) is the \(n\)th root of the product of all the values. The geometric mean is useful for finding the average multiplier.

For example, if an investment returns 7%, -5% and 28% in consecutive years, the average return is the geometric mean of the multipliers, 1.07, 0.95, and 1.28, which is 1.09, or a 9% return. Using the arithmetic mean would show an 11% average return.

Our first step is to write some tests to define how our new function should behave, taking the geometric mean of numbers that are elements of a matrix. The initial tests are simple cases where we can compute the answer directly.

Next we write our simple JSL function to compute the geometric mean using the Function operator and the Power (^) operator.

```
gmean = Function( {m},
    Local( {p, i, j},
        p = Product( i = 1, N Row( m ), m[i] );
        p ^ (1 / N Row( m ));
    )
);
```

When we run the tests with the test driver, the output report shows that all tests pass.

Running all tests again shows that all new tests fail.

The failures are because the : operator produces a row vector, but our \(gmean\) function only handles column vectors. We update \(gmean\) to cycle over rows and columns by using nested Product operators.

```
gmean = Function( {m},
    Local( {p, i, j},
        p = Product( i = 1, N Row( m ),
            Product( j = 1, N Col( m ), m[i, j] )
        );
        p ^ (1 / N Row( m ));
    )
);
```

This test also fails, though with a different result.

It looks like we forgot to consider the number of columns when taking the root of the product. A new version of our function fixes that.

(continued next page)
Now all tests pass again. 😊
Next we add a test for really large values. Thanks to the framework’s consideration of relative errors, expected rounding errors don’t cause problems and the test passes.

For more thorough testing, we write a loop that generates data sets and checks that `gmean` computes the right answer. So that we can know the answer without using `gmean`, we create the data so that each value is raised to the nth power so when `gmean` takes the nth root, it should just get back the product of the original values.

All tests still pass. 😊 Notice there are now 60 tests since our added test was in a loop and was applied 50 times.

Finally, now that we have a good suite of test, we can confidently change the implementation of our function, knowing that any regression will trigger a unit test failure.

We change the implementation by reverting the nested product to a simple product after using the `Shape` operator to convert the matrix into a row vector.

The final test passes; the function is complete. 😊

Note: You may download JSL-Unit from the JMP Extras section of the JMP User community site: www.jmp.com/community

---

**Book Discussion**


The fourth edition of *JMP Start Statistics* by John Sall, Lee Creighton, and Ann Lehman is fresh off the press. Based on comments and suggestions by teachers, students and other users, *JMP Start Statistics* has been expanded to include new features in JMP 7.

This book is a mix of software manual and statistics text. The fourth edition continues to focus on using JMP to learn about statistics. It is designed to be a complete and orderly introduction to analyzing data. It is also a teaching text, but is especially useful when combined with a standard statistical textbook.

Chapters have been rearranged to streamline the flow of information and make the book easier to use in the classroom. New sections and chapters have been added where needed to illustrate new features in JMP 7, such as the 3-D scatterplot.

The “Design of Experiments” chapter has been completely re-written to reflect the popularity and utility of optimal designs. Optimal screening designs are created and modified interactively with the JMP Custom Designer. Data from screening designs is addressed by the new Screening analysis platform, which uses calculations and graphical presentations especially suited for finding active effects.

Each topic is supported with hands-on examples of varying levels of complexity. JMP’s interactive statistical platforms help clarify and simplify what are often perceived as difficult and tangled-up statistical concepts. Chapters conclude with a set of lab or homework exercises.

Last (but not least), to meet ongoing requests, particularly from teachers and students, the fourth edition of *JMP Start Statistics* includes an appendix with answers to selected exercises, organized by chapter.

To order a copy of *JMP Start Statistics*, go to http://www.jmp.com/academic/books.shtml or contact SAS Publishing at sasbook@sas.com
Polynomial models have become the workhorse of experimenters. Regardless of the purpose of your experiments, whether it is to screen factors or to optimize factor levels, a polynomial model often fits the data well.

Experimenters have learned the distinct roles of each type of model term—main effects, interaction effects, and quadratic effects—are all part of the common parlance. With experience comes comfort using higher order polynomials, but the principle of parsimony always points toward the simplest model that is adequate for the purpose at hand.

When an inferential test indicates that a higher order term is statistically significant, is there ever a reason to remove it? Is it possible that a non-additive component of the response is a side effect of the way that the data is measured or expressed? Sometimes, when you transform the response, it no longer exhibits this component.

A generalized power transformation developed by Box and Cox (Box, Hunter, and Hunter (2005)) has been used to achieve well-behaved data with regard to model assumptions such as homoscedasticity (homogeneity of variance). Data transformation often provides other benefits as well, one of which is that transforming might enable a simpler model to fit the data.

The transformed response, $y'$, is computed by re-expressing each data value, $y$, as shown here,

$$ y' = \frac{y^\lambda - 1}{\lambda \mu} \quad \mu \neq 0 \quad \mu = \frac{\prod_{i=1}^{n} y_i}{n} $$

where log is the natural logarithm and the geometric mean of the original data, $\mu$, is employed as a scaling factor. The Box-Cox transformation is available in the Fit Model platform or it could be implemented in JMP as a column formula.

This article explores the above transform with an example from Box, Hunter, and Hunter (2005). The durability of woven thread is measured as the number of repeated loadings that a sample survives. In this study, the length of the thread specimen, the amplitude of the loading cycle, and the load are varied at three levels, according to a $3^3$ full factorial design.

The typical model used in response surface methodology (RSM) is the full quadratic model, including all of the two-way interaction effects. Figure 2 shows these results, as produced by the M1 script in the data table.

**Figure 2 Results of Response Surface Fit**

<table>
<thead>
<tr>
<th>Term</th>
<th>Estimate</th>
<th>Std Error</th>
<th>t Ratio</th>
<th>Prob&gt;</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intercept</td>
<td>550.69267</td>
<td>126.4379</td>
<td>3.99</td>
<td>0.001*</td>
<td></td>
</tr>
<tr>
<td>Specimen Length (200, 300)</td>
<td>490</td>
<td>64,985.21</td>
<td>16.30</td>
<td>&lt;0.001*</td>
<td></td>
</tr>
<tr>
<td>Load Cycle Amplitude (147, 16)</td>
<td>-561.9999</td>
<td>64,985.21</td>
<td>-8.75</td>
<td>&lt;0.001*</td>
<td></td>
</tr>
<tr>
<td>Load 49 (50)</td>
<td>-316.7768</td>
<td>64,985.21</td>
<td>-4.85</td>
<td>0.0002*</td>
<td></td>
</tr>
<tr>
<td>Specimen Length x Load Cycle Amplitude</td>
<td>236.6667</td>
<td>116.9907</td>
<td>2.05</td>
<td>0.0462*</td>
<td></td>
</tr>
<tr>
<td>Load Cycle Amplitude x Load Cycle Amplitude</td>
<td>757.6667</td>
<td>116.9907</td>
<td>6.53</td>
<td>&lt;0.001*</td>
<td></td>
</tr>
<tr>
<td>Specimen Length x Load Cycle Amplitude</td>
<td>-256.6667</td>
<td>76,490.59</td>
<td>-3.30</td>
<td>0.0010*</td>
<td></td>
</tr>
<tr>
<td>Load Cycle Amplitude x Load</td>
<td>-4.145</td>
<td>76,490.59</td>
<td>1.92</td>
<td>0.0581</td>
<td></td>
</tr>
<tr>
<td>Load</td>
<td>-49.351</td>
<td>116.9907</td>
<td>-4.24</td>
<td>0.0581</td>
<td></td>
</tr>
</tbody>
</table>
All of the effects except for Load*Load are deemed to be statistically significant. The fit is adequate, but difficult to interpret. A transformation that might improve the fit or simplify the model can be identified using the **Factor Profiling > Box Cox Y Transformation** platform command. Click the red triangle on the Response Life Times title bar to select this command, as shown in Figure 3.

**Figure 3 Box Cox Y Transformation Command**

The model is repeatedly fit while lambda is varied from −2 to 2 in 0.2 increments. For each value of lambda, the error sum of squares (SSE) is recorded and displayed in a lambda plot for the SSE (see Figure 4).

**Figure 4 Box Cox Transformation Plot**

The minimum SSE value indicates the appropriate choice of lambda. Experts recommend rounding lambda to a value that may imply some physical meaning. This principle usually means selecting a value that corresponds to a common power such as 0 (logarithm), 2 (square), 1/2 (square root), or −1 (reciprocal). A lambda value of 1 corresponds to the metric of the original data, which indicates that no transformation is necessary. In this example a lambda close to 0 seems to provide a good transformation. This lambda value corresponds to a logarithm transform.

Note: An automatically chosen transformation, in this case the reciprocal fifth root (lambda of −1/5 or −0.2), can be saved as a new column with its formula. To do this, click the red triangle on the Box-Cox Transformations title bar and select **Save Best Transformation** from the menu. The new column name is the original response name with a suffix “X”.

If you want to implement the log transformation,
- return to the Fit Model dialog,
- select the response, Y
- click on the Transform hot spot
- select Log from the hot spot menu

For this demonstration, make sure the full quadratic model is still specified. Then click Run Model. The M2 script in the data table automatically runs this log model for you. Figure 5 shows the results.

**Figure 5 Analysis of Log Transformation on Response**

The script in the data table automatically runs this log model for you. Figure 5 shows the results.

**Table 1 Comparison of Original and Transformed Results**

<table>
<thead>
<tr>
<th>Statistic</th>
<th>Original</th>
<th>Transformed</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSE</td>
<td>271.8867</td>
<td>0.193833</td>
</tr>
<tr>
<td>$R^2$</td>
<td>0.937789</td>
<td>0.972472</td>
</tr>
<tr>
<td>F-Ratio</td>
<td>28.5176</td>
<td>66.7326</td>
</tr>
<tr>
<td>$p &gt; F$</td>
<td>1.56x10^{-8}</td>
<td>1.73x10^{-11}</td>
</tr>
</tbody>
</table>
By all measures, the transformation greatly benefits the fit. Now inspect the individual effect tests. Notice that only the main effects remain significant. That is to say, the purely additive model for the transformed response is sufficient to fit the data. This reduction in the complexity of the model is remarkable. How does it work?

For each of the fits that were explored during the evaluation of $\lambda$, other regression statistics, such as the $t$-ratio for each effect, could have been recorded instead of the SSE. The $t$-ratio is the ratio of the difference between the hypothesized parameter and the parameter estimate to the standard error of the estimate. To test the hypothesis that a parameter is zero, the $t$-ratio is simply the estimated coefficient divided by its standard error. A JMP script can produce this plot in a separate window, as shown in Figure 6.

The lambda plot in Figure 6 shows how the $t$-ratios change in strength and direction as the data are transformed by different values of lambda. The curves for all of the interaction effects have been selected by the check boxes on the right, and are highlighted in the plot. You can see that when lambda is 0 (indicated by the dashed vertical line) their $t$-ratios shrink and become insignificant, while at the same time, the $t$-ratios for the main effects become stronger.

Note: The script that produces the $t$-ratio plot provides similar lambda plots for the F-ratio, SSE, and $R^2$. It is available with this issue of JMP Per Cable at: wwwjmp.com

In contrast, the lambda plot for SSE shown previously in Figure 4 shows a broad, flat region, and so it is less discerning about the best choice of lambda for the quadratic model. The reason is that the RSM model is able to fit the data well even without transformation. The same plot would exhibit a sharper curve if only the main effects model were profiled.

To summarize, if you start with a screening design and use a first order model, then you might prefer the lambda plot of SSE, F-ratio, or $R^2$ to select lambda. If you start instead with a second-degree polynomial, then you might find the lambda plot of $t$ ratios more informative. The Box-Cox transformation simultaneously simplifies the model to first order and increases the power of the regression analysis. So, what model are you going to simplify today?

Contact: mark.bailey@sas.com or cfung@earthlink.net for further information.


---

**Figure 6** Lambda Plot for Comparison of $t$ Ratios, for all Model Effects

---

**Comments from the innovators’ Summit, Oct 16, 2007, Traverse City, Michigan**

“Using JMP, our company saved $6 million in the first six months of this year. Even for a company that brings in $100 million a month, that’s a lot of money.”
- Attendee, Aera Energy, LLC Bakersfield, CA

“The ultimate purpose of collecting data is to provide a basis for action or a recommendation for action.”
- J. Stuart Hunter, PhD, Princeton University, Plenary Session Speaker

“The speakers are like flowers, and we’re like bees.”
- Attendee from a major consumer products company referring to the value of interaction at the conference
A Problem with an Incorrect Split Plot Analysis
Annie Dudley Zangi, SAS Institute

The following example shows problems that can occur when you incorrectly analyze split-plot data as a crossed model.

Consider the following scenario. A semiconductor company is having difficulty with the insulating ability between the channels (wiring) on wafers. A measurement, $k$, indicates low leakage across wires or channels. Ideally, $k$ is as low as possible. The company wants to find out if different types of diffusion furnaces affect the insulating ability.

To address this problem, the company runs an experiment with their three standard etch methods, their three old diffusion furnace types, and a new type of diffusion furnace.

Note: There are many steps in the semiconductor building process. This analysis is a simplified example that only analyzes two possible factors, diffusion furnace and etch method. Often, batches (or lots) of four boats at a time, each holding twenty wafers, are in the furnace for each application.

The data table SplitPlotMistake.jmp, shown in Figure 1, has the data from this experiment.

![Figure 1: Partial Listing of the Split Plot Table](image)

The first script saved with the data (Fit Model - Crossed) runs the analysis for a standard two-factor crossed design. These results (Figure 2) show that the Diffusion Furnace effect was significant. Most notably, the furnaces with the lowest insulating ability were the new diffusion furnaces. You can see this especially well looking at the LSMeans plot for Diffusion Furnace.

In this scenario, an expensive new facility was built based on the crossed analysis, using the new diffusion furnaces. However, after the plant was up and running, the insulating ability problem did not improve.

The next step was to hire a statistical consultant to examine the analysis in more detail. The Statistician noticed the split-plot structure in the data, where the lots were whole plots and the individual wafers were the subplots. The etching was applied to individual wafers, rather than the whole lot, making the error structure different from the error structure for Diffusion Furnace.

The second script saved with the data (Fit Model - REML) analyzes the data as a split-plot model instead of the standard two-way crossed model. This split-plot analysis shows that Diffusion Furnace was not a significant effect. Instead, both the Etch method and its interaction with Diffusion Furnace were significant. This meant that the insulating ability could be improved using an existing combination of diffusion furnaces and etching methods—there had not been a need to purchase any new furnaces at all.

![Figure 2: Results from Analysis of Crossed Data](image)

![Figure 3: Split-Plot Analysis Shows Different Results](image)
Questions, comments, or for more information about JMP, call
1-877-594-6567
or visit us online at
www.jmp.com

To Order JMP Software
1-877-594-6567