A Technical Publication for JMP® Users
Issue 26 Winter 2010

JMPer Cable

Inside This Newsletter

Slicing Solar Array Data

Xan Gregg., JMP Division of SAS Institute

As you may have read, as of January 1, 2009, the SAS world headquarters in Cary, NC has a 1-megawatt photovoltaic solar farm in operation. The 5-acre farm is one of the largest solar farms in the southeastern United States. A second solar farm is scheduled for completion by late March 2010 and will generate an estimated 1.9 million kilowatt-hours (kWh) annually, enough to power more than 200 homes.

SAS Solar Farm photo by Dave Horne

The output is measured separately for two halves of the farm, called Array A and Array B. You can see the arrays, along with the sheep that maintain the grass at the solar farm, in the photo.

Fun Facts

The projected production of the existing farm is 1.7 million kWh/year with efficiency enhanced by using a directional system that tracks the path of the sun. The energy generated should (or could) have the following annual effect.

- reduce carbon dioxide emissions by more than 1,600 tons
- reduce carbon dioxide emissions equivalent to the consumption of more than 167,000 gallons of gasoline
- produce enough energy to power 160 average-size North Carolina homes
- operate an average 15W compact fluorescent light for 120 million hours
- power 616,438 LED night lights used eight hours per day

Exploring the Solar Farm

This article looks at the production of the solar arrays from January through August. You can play with the SAS solar farm data yourself by downloading it from the JMP File exchange on the JMP web site. The data table has power output recorded every 15 minutes beginning January 1, 2009 for each of the two farm arrays. The table also records ambient temperature, wind, and irradiance (used to indicate sun shininess). Using column formulas, variables are extracted from the date-time variable to give measurements for individual months, days, and times. A season variable was created by grouping the observations into two-month categories.

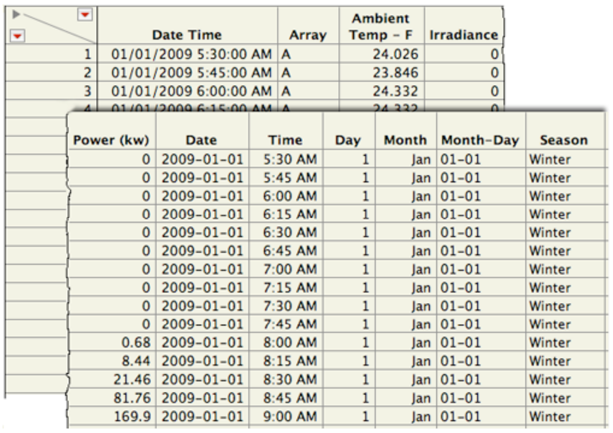
JMP is 20 Years Old

JMP Version 1 shipped on October 5, 1989 — or as we claimed at the time, September 35 — so that we could say we shipped in the third quarter of 1989. The big driving force has been meeting the needs of those users we talk to, who correspond with us, and who sometimes invite us into their sites. We have a very dedicated group of users who keep us directed, and help us serve more and more researchers every year.

John Sall
Executive Vice President
SAS Institute, Inc.

We used a subset of the data that eliminated nighttime hours when there was no sunlight, and brief periods of time where one or the other array was off-line. The resulting data table has 22,174 observations, divided equally between the two arrays. *Figure 1* shows the first few observations in the solar array data table. The table also has scripts that create the graphs in this article.

Figure 1 Example of Solar Array JMP Data



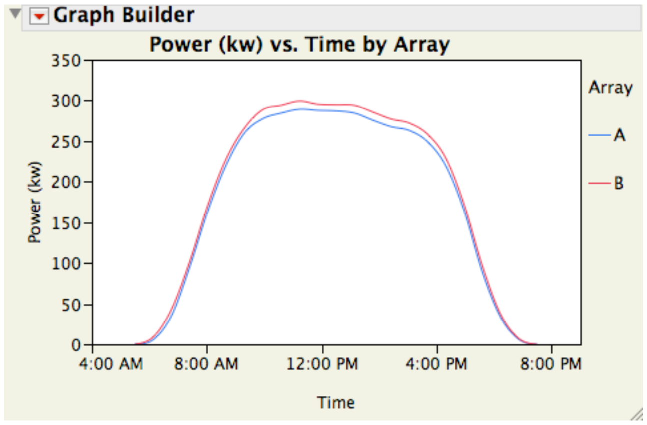
	Date Time	Array	Ambient Temp – F	Irradiance
1	01/01/2009 5:30:00 AM	A	24.026	0
2	01/01/2009 5:45:00 AM	A	23.846	0
3	01/01/2009 6:00:00 AM	A	24.332	0
4	01/01/2009 6:15:00 AM	A	24.332	0

Power (kw)	Date	Time	Day	Month	Month-Day	Season
0	2009-01-01	5:30 AM	1	Jan	01-01	Winter
0	2009-01-01	5:45 AM	1	Jan	01-01	Winter
0	2009-01-01	6:00 AM	1	Jan	01-01	Winter
0	2009-01-01	6:15 AM	1	Jan	01-01	Winter
0	2009-01-01	6:30 AM	1	Jan	01-01	Winter
0	2009-01-01	6:45 AM	1	Jan	01-01	Winter
0	2009-01-01	7:00 AM	1	Jan	01-01	Winter
0	2009-01-01	7:15 AM	1	Jan	01-01	Winter
0	2009-01-01	7:30 AM	1	Jan	01-01	Winter
0	2009-01-01	7:45 AM	1	Jan	01-01	Winter
0.68	2009-01-01	8:00 AM	1	Jan	01-01	Winter
8.44	2009-01-01	8:15 AM	1	Jan	01-01	Winter
21.46	2009-01-01	8:30 AM	1	Jan	01-01	Winter
81.76	2009-01-01	8:45 AM	1	Jan	01-01	Winter
169.9	2009-01-01	9:00 AM	1	Jan	01-01	Winter

One fundamental question is whether the arrays have been behaving alike and are producing the same amount of power. But rain, seasons, and cloudiness make things messy—the perfect data exploration terrain for JMP.

A broad look at power production by time of day over the eight month span for the two sets of panels (A and B) can be seen using **Graph** > **Graph Builder**, where the variable Power is Y, Time is X, and Array is the Overlay variable. The Graph Builder display in *Figure 2* shows that array B performs slightly better on average during midday and early evening hours than array A.

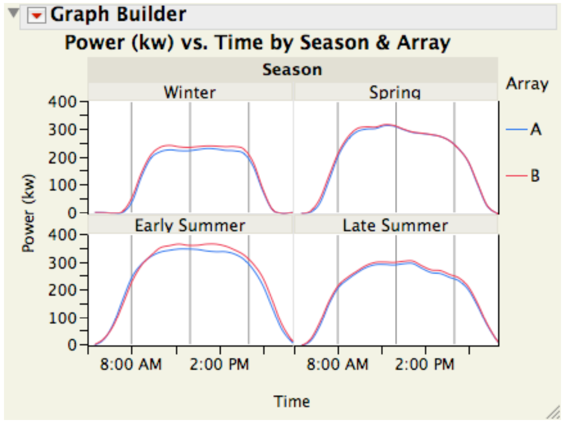
Figure 2 Power by Time for Arrays A and B



A comparison of the arrays in different seasons, different months and even comparing days might shed light on why there are differences in array production.

Figure 3 breaks the plots into two-month groups labeled Season, which is used as the Graph Builder Wrap variable. You can see changes in the shape and height of the curves as the number of daylight hours increases. The Winter (January and February) and Early Summer (May and June) plots clearly show array A under-performing B during the midday hours, but during the spring and late summer production appears nearly the same.

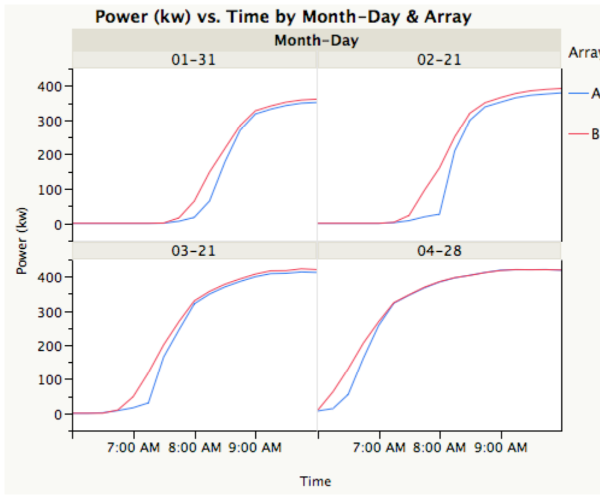
Figure 3 Power by Time and Season for Arrays A and B



A more refined look at power output and behavior of the two arrays is made by looking at selected days. Using the sum of Irradiance for each day as an indication of sunshine, some of the days with highest irradiance were examined.

In the blog posted in April, it was noted that many of these sunny days, especially in winter and early spring, showed a lag in power output between the arrays during early morning hours. *Figure 4* shows an example of 4 days with Array A lagging behind B as the sun comes up.

Figure 4 Power by Time and Season for Arrays A and B



The photograph of the farm gives a clue to why there is a difference in performance. Notice the shadows cast by fence posts and sheep. One blog commenter said, “I assume array A lags B because it is close to the east side and there must be some shadow in the morning.” Another said, “If the solar panels in Array B are aligned on the east side of the farm, then their exposure to the rising sun before those in Array A explains the early morning lag in power output.” The bloggers are correct. Shade from the trees on one side of the farm and rotation of the panels explain the early morning lag in power between the arrays.

Figure 5 shows days selected to illustrate how power output changes on sunny days during the seasons. And, since there is almost always a fair amount of rain in the spring, May 3 is included so that you can see how cloudy weather affects the solar panel readings.

You can clearly see the following behavior.

- Power output is greater in the summer (height of the curves).
- Daylight lasts longer in the summer (width of the curves).
- Spring and summer output is noisier (wiggly curves).

Note that the winter days (January and February) show a midday dip that seems to moderate as the weather gets warmer and the days get longer.

Two blog commenters pointed out that since there's only one axis (north-south), the angle of the arrays against the ground is optimized for morning and evening hours, and the horizontal tracking is sub-optimal around noon because Cary, NC is 35 degrees north of the equator.

The fact that this dip is greater in the winter supports the idea that the horizontal axis of the panel rotation causes the dip, because it is more pronounced when the sun is lower in the sky.

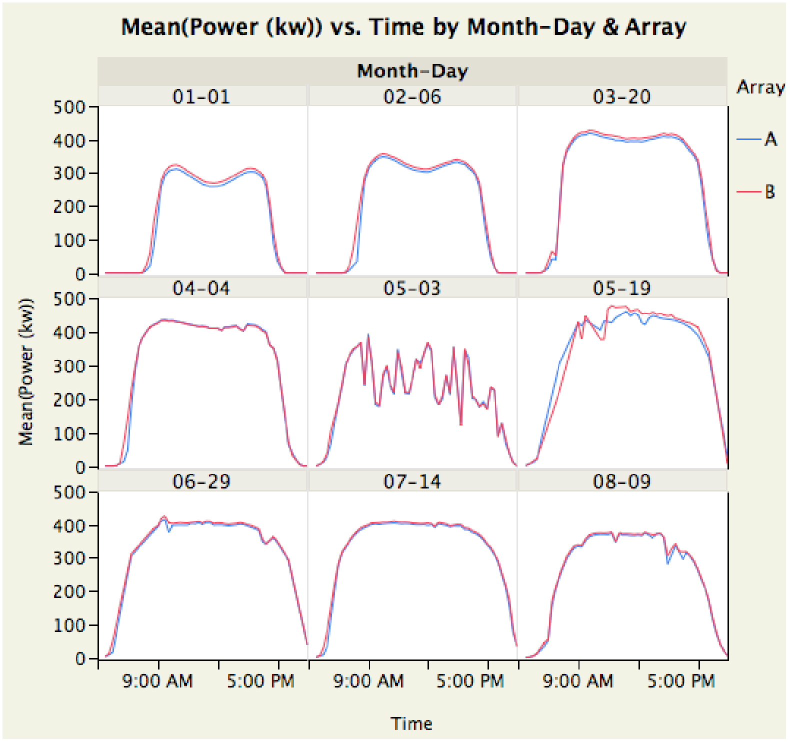
Wikipedia says “Since [single-axis horizontal trackers] do not tilt toward the equator, they are not especially effective during the winter midday (unless located near the equator), but they do add a substantial amount of productivity during the spring and summer seasons when the solar path is high in the sky.”

There could be other factors affecting overall power output of the arrays, such as ambient temperature. Another blog commenter noted that “manufacturer and solar photovoltaic literature clearly demonstrate that solar panels exhibit a loss in efficiency that is inversely proportional to panel temperature—the hotter the solar panel, the lower the output. The solar panels at midday are at or near horizontal, thereby obstructing wind and warm/hot airflow underneath the panels. In other words, there's a slightly greater tendency for hot air to accumulate underneath the panels when they are all laying in a near perfect plane to one another. Also, at midday, the solar irradiation is at its peak maximum, so panel temperatures should be at a maximum peak for the day.” How does this affect power output?

Clearly, more data exploration is needed. You can access the data used here, or the entire set of data, and respond to the blogs on the SAS solar arrays. We welcome further analytical insight on the solar array production.

http://blogs.sas.com/jmp/

Figure 5 Mean Power (kw) by Time for Selected days, Colored by Array



JMP is 20 Years Old

John Sall, JMP Division of SAS Institute

October 5 was the 20th anniversary of JMP's first release, and I want to thank everyone who has helped to make JMP a success.

JMP version 1 shipped on October 5, 1989—or September 35 as we claimed then so we could say we met our goal and shipped in the third quarter of 1989.

JMP started as a research project in the late '80s. In the earlier part of that decade, we had spent several years rewriting SAS completely (but compatibly) to fit on personal computers. But by 1988, we felt three big forces, which can be characterized by:

- The vehicle—cars as well as trucks
- The roles—detectives as well as lawyers
- The technology—pointing as well as writing

The Vehicle

SAS was becoming a large enterprise-scale product—a larger investment than some users, like engineers and scientists, were willing to handle. We were producing analytical trucks, but there was a market for analytical cars, (something with low investment and ease of driving). We needed a more personal-scale tool, one for the desktop project rather than for the enterprise system.

The Roles

Statistics itself saw the opportunities in exploratory techniques, and the value of graphics and interactivity. The statistics profession had been molded as a testing discipline, a role like a lawyer whose job is to prove things we already knew. What was missing was the exploratory role, like a detective, whose job is to discover things we didn't already know. With John Tukey's *Exploratory Data Analysis* and the improvement of statistical graphics, statistics began to serve in the detective role as well as the lawyer role. Graphics was the key enabler of seeing patterns and points that don't fit patterns.

The Technology

The graphical user interface arrived with the Macintosh, and later, Windows. It is a huge difference to point and click rather than look up and type. Applications written for batch computing through languages were not suited for graphical interactivity. It was time for some fresh design.

In response to these three forces, a small group put together version 1 of JMP in a year and a half. This was a very small product compared to the JMP of today, but it had all the basics of statistics and graphics, with many innovative features. We thought "jump" was a name to suggest a big step into a new future, a product that jumps in responsiveness to the mouse, and a tool that enables our customers to do the experiments and make the discoveries to take huge strides in their products and processes.

In the early years, we learned that engineers and scientists were our most important customer segment. These people were smart, motivated, and in a hurry—too impatient to spend time learning languages, and eager to just point and click on their data. We had a product that was nearly as easy as walk-up-and-use with enough delights to hold their loyalty.

We learned that engineers need Design of Experiments (DOE), quality and productivity support (Six Sigma), and reliability modeling. We made a concerted effort to improve in these areas. We thought that engineers should be able to simply ask the computer to custom-make a design that fits their needs rather than attempting to find a pre-built design that fit their circumstance.

In JMP version 3.1 we learned how to port to Windows using the Altura library. Soon we were busy rewriting the whole product in a different implementation language with a portability host-interface layer, which led to a wait of more than three years before Version 4. Version 4 not only switched languages, but also introduced a new nervous system for the product, including the JMP Scripting Language.

In the last few years, JMP has matured considerably. The driving force has been in meeting the needs of those users we talk to, who correspond with us, who sometimes invite us into their sites. We have a very dedicated group of users who keep us directed, and help us serve more and more researchers every year. Recently, I heard the group of passionate JMP users refer to the "JMPerati," analogous to Stephen Baker's term, the "numerati."

JMP has broadened and become more versatile. JMP now supports business visualization in partnership with SAS Business Intelligence (BI). This, in turn, has encouraged us to introduce more visualization platforms, like the drag-and-drop Graph Builder in JMP 8. JMP now handles larger problems because of work we have done to multithread many of the bottleneck methods and to implement JMP on 64-bit systems. We now work with various SAS teams on projects in several areas, notably JMP Genomics, collaborating and sharing efforts.

JMP is 20 years old, but it seems like it is just getting started. We are growing fast. Last year, our business grew faster than ever, and we have geared up and prepared to grow even faster in the future.

Happy birthday, JMP, and thank you, everyone, for your contributions to JMP's success.

See this blog and others about the JMP 20 year anniversary at

http://blogs.sas.com/jmp/

Script Snippet

Q: How can I make a function that evaluates another function given later with a variable number of arguments?

A: You can construct the function by inserting arguments, then calling the constructed function.

```
/* evaluate the function named in the first
   argument with the list of arguments
   specified in the second argument */
subfun = Function({funName,argList},
   ni = nitems(argList);
   afun = nameExpr(funName);
   for(i=1,i<=ni,i++,afun =
       insert(nameExpr(afun),argList[i]));
   afun;
);
expfunction = function({x},exp(x));
divfunction =
   function({arg1,arg2},arg1/arg2);
subfun (expr(expfunction),{2});
subfun (expr(divfunction),{2,3});
```

For more, see page 9 for a detailed discussion of expression handling functions, by Joseph Morgan.

The Buzz About JMP Discovery and Innovators Summit

Arati Bechtel, JMP Division of SAS Institute

While at Discovery 2009 and Innovators' Summit in Chicago last week, John Sall, chief architect of JMP, spread the word about JMP, SAS and the value of analytics. He met with journalists and bloggers, and some of the coverage has already been published online. Take a look:

- In a story for *Computerworld*, reporter Eric Lai asked Sall why JMP chose best-selling author Malcolm Gladwell to headline the Chicago conferences. "Journalists are like detectives. A lot of our customers are like that," said Sall, who is co-founder and Executive Vice President of SAS. "They don't just want to prove things you already know. That's lawyer stuff. They want to look at statistical outliers and figure out new things."
- Greg Burns, Senior Business Correspondent for the *Chicago Tribune*, talked with both Sall and Gladwell during the conferences. It is part of what Burns is calling Big Brain Week, with Sall representing left-brain thinking and Gladwell representing right-brain thinking. Burns' interview with Gladwell appeared in two parts, September 24th and 25th.
- Data visualization guru Stephen Few, Principal of Perceptual Edge, wrote about the conferences and Gladwell's keynote speech in his influential blog. Few applauded Gladwell's insight that modern problems present the challenges of too much information and too little understanding of that data, pointing to a need for good analytics. "More directly related to my work in BI, I've stated countless times that this industry has done a wonderful job of giving us technologies for collecting and storing enormous quantities of data, but has largely failed to provide the data sense-making tools that are needed to put data to use for decision-making," Few writes.

Visit Stephen Few and others at

http://www.perceptualedge.com/blog/

If you didn't get to Chicago for Discovery and the Innovators' Summit, you can get a sense of why some attendees said it was the best conference they'd ever been to by checking the coverage on our web site.

Isn't That Saying the Same Thing?

José G. Ramírez, Ph.D., W. L. Gore and Associates, Inc.
Mark Bailey, Ph.D., SAS Institute Inc.

Savvy users and even regulatory agencies have come to realize that how you ask a question is very important. Crafting the question depends on how you think about your situation, your purpose, and the claim you want to make. The meaning of human language and the logic of statistical tests of significance, however, are somfetimes at odds with each other. For example, the questions "Is there a difference between them?" and "Are they the same?" seem like equal but opposite questions to most people. A negative answer to one of these questions usually means an affirmative answer to the other question. Unfortunately, that is never true for significance tests! Failing to show a statistically significant difference between things is not the same as showing that things are the same.

You Cannot Prove the Null

Significance tests work in only one direction—that of disproving the *null* hypothesis (H_0). You define the *alternative hypothesis* (H_a) in terms of what you are trying to prove by rejecting the null hypothesis. In this context, you 'prove' the alternative hypothesis in the sense of "establishing by evidence a fact or the truth of a statement" (The Compact Oxford English Dictionary). For example, the hypotheses that compare the average performance of a product to a standard, *k*, are written:

H_0: Average Performance (μ) = Standard (κ) (Assume)

H_a: Average Performance (μ) $\neq$ Standard (κ) (Prove)

Suppose you work on an improvement project and want to show that the improvement has a real effect on a measurement (a response). The alternative hypothesis is

H_a: The mean response is different after the improvement

The other possibility, the null hypothesis, is

H_o: The mean is the same in spite of the improvement

You *assume* that H_o is true and base your expectation on it. If statistics do not allow you to reject the null hypothesis then there is not enough evidence to say the mean response is different after the improvement. Only when there is sufficient evidence against the null hypothesis (that is, when your result is highly unlikely if H_o is true), do you reject it in favor of the alternative hypothesis (H_a).

Proving Sameness

In another situation, you may not want to show that things have changed, but that they remain the same. For example, you need to qualify a new piece of manufacturing equipment by showing that it does not change the key performance characteristic of a final product. In this case, the alternative hypothesis is

H_a: the average performance of the final product is the same with the new equipment

versus the null hypothesis

H_o: the average performance of the final product is not the same with the new equipment

This situation demands that H_o and H_a are stated opposite of the way traditional significance tests define them. You now assume that the equipment swap caused an effect unless you have sufficient evidence to reject that assumption. Such hypothesis tests have come to be known as *equivalence tests*.

Either way, they are all significance tests. The distinction just depends on how you define the two hypotheses. Unfortunately, many textbooks do not make this distinction and only address the first example, which can be called a *difference test. Table 1* below summarizes the two views of statistical tests.

Tests of Equivalence

When you say that two means are the same, what you really say is that the difference between them is so small that it is not of practical importance. This acceptance is established in practical terms such that performance is not adversely affected. In other words, you can never show that the means are exactly the same (they will be different in some decimal place if measured with an accurate enough instrument).

Table 1 Two Views of Statistical Tests: Null and Alternative Hypotheses

Test	**H_o**	**H_a**	**Reject**	**Fail to Reject**
Difference	No Effect	Real Effect	Sufficient evidence to reject "No Effect"	Insufficient evidence to reject
Equivalence	Real Effect	No Effect	Sufficient evidence to reject "Real Effect"	Insufficient evidence to reject

Since you cannot prove equality, you show that two sample means are statistically *equivalent* within an acceptable interval. In this situation you are responsible for defining what the acceptable equivalence interval is.

The equivalence hypotheses that compare the average performance of a product with a standard, *k*, are written

H_0: |Average Performance – k| > δ (Assume)

H_a: |Average Performance – k| ≤ δ (Prove)

where δ represents the *equivalence bound*, or practical threshold within which the average performance is considered to be the same as the standard.

The above hypothesis is actually two sets of one-sided hypotheses, namely

H_0: Average Performance > δ + k (Assume)

H_a: Average Performance ≤ δ + k (Prove)

H_0: Average Performance < k – δ (Assume)

H_a: Average Performance ≥ k – δ (Prove)

The first set uses a test of the mean to show that the difference between the average and the standard is significantly less than the upper equivalence limit of the interval and, *at the same time*, the second set tests the means to show that the difference between the average and the standard is significantly greater than the lower equivalence limit. This simultaneous application of a lower-tailed test and an upper-tailed test is often referred to as *Two One-Sided Tests* or TOST.

Example: Is the DC Resistance = 50 Ohm?

For comparing a mean to a standard JMP does not provide the TOST directly. Instead, you simply apply the two tests separately. Let's look at an example. An improved version of a cable will be sent to the customer and you need to demonstrate that it performs according to the required DC resistance target of 50 Ohm. The cable is considered to be equivalent to the specified target if it is within 0.6 Ohm. To demonstrate this equivalence, data was collected from a week of production by randomly selecting 8 cables per day from the production floor, for a total of 40 cables. The resistance of each of the cables is measured and recorded. *Figure 1* shows a histogram, along with summary statistics of the 40 DC resistance measurements. The data table, called Resistance Data.jmp is available with this newsletter on the JMP web site.

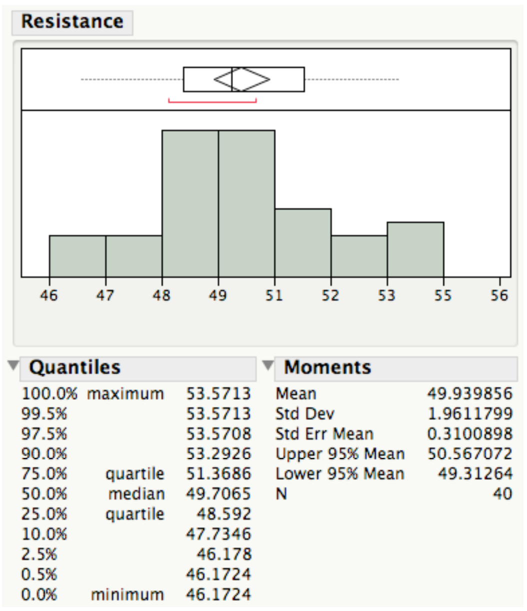


Figure 1. Distribution of 40 DC Resistance Measurements

Quantiles

100.0%	maximum	53.5713
99.5%		53.5713
97.5%		53.5708
90.0%		53.2926
75.0%	quartile	51.3686
50.0%	median	49.7065
25.0%	quartile	48.592
10.0%		47.7346
2.5%		46.178
0.5%		46.1724
0.0%	minimum	46.1724

Moments

Mean	49.939856
Std Dev	1.9611799
Std Err Mean	0.3100898
Upper 95% Mean	50.567072
Lower 95% Mean	49.31264
N	40

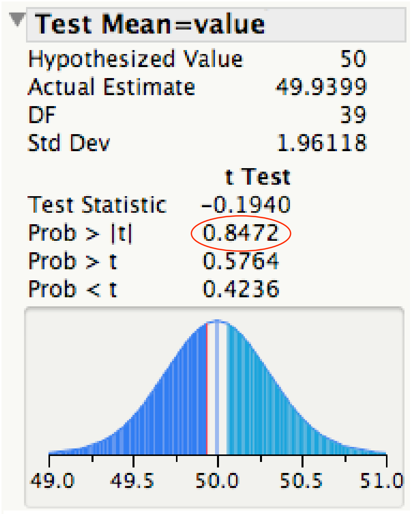
You can see that the average DC resistance of this sample of 40 cables is 49.94 Ohm, which is close to the target of 50 Ohm, with a variation of 1.96 Ohm. The JMP distribution platform can do a traditional significance test to test the hypothesis

H_0: Average DC Resistance = 50 Ohm (Assume)

H_a: Average DC Resistance ≠ 50 Ohm (Prove)

The *p*-value for the two-sided *t*-test in *Figure 2* is 0.8472, which indicates there is not have enough evidence to say that the average DC resistance of the 40 cables is not 50 Ohm.

Figure 2. Significance Test for DC Resistance = 50 Ohm



Test Mean=value

Hypothesized Value	50
Actual Estimate	49.9399
DF	39
Std Dev	1.96118
	t Test
Test Statistic	-0.1940
Prob > |t|	0.8472
Prob > t	0.5764
Prob < t	0.4236

Equivalence Test

Even though there is not enough evidence to say that the average DC resistance is not 50 Ohm, you can see in *Figure 2* that the average DC resistance, being 49.94 Ohm, is only 0.06 Ohm different from 50. A difference of 0.06 Ohm is well within the measurement error of 0.2 Ohm, so for all practical purposes there is no distinguishable difference. Unfortunately, this *t*-test does not help prove equivalence.

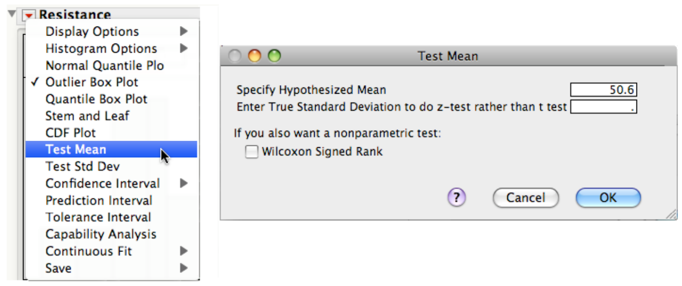
A practical threshold is needed to prove equivalence. Since the given measurement error is 0.2 Ohm, you can decide to define the equivalence window as 0.6 Ohm, or 3 times the measurement error. Then, the two one-sided hypothesis becomes:

H_0: Average Resistance > 50.6 Ohm	(Assume)
H_a: Average Resistance ≤ 50.6 Ohm	(Prove)
H_0: Average Resistance < 49.4 Ohm	(Assume)
H_a: Average Resistance ≥ 49.4 Ohm	(Prove)

You can conduct two one-side *t*-tests to show that the average resistance is less than 50.6 Ohm, and at the same time greater than 49.4 Ohm.

From the Distribution output, click the red triangle on the Resistance title bar and select **Test Mean**, as shown in *Figure 3*. When the Test Mean dialog appears, enter 50.6 for the hypothesized mean, and then click the **OK**.

Figure 3 Fit Mean Option on the Distribution Platform



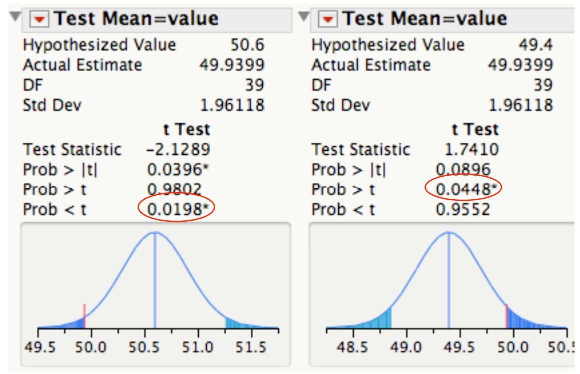
Specify Hypothesized Mean 50.6
Enter True Standard Deviation to do z-test rather than t test
If you also want a nonparametric test:
☐ Wilcoxon Signed Rank

The report on the left in *Figure 4* shows the *p*-value associated with the lower-tailed test, labeled Prob < t, is equal to 0.0198 (< 0.05). This small value is enough evidence to reject H_0 and say that the average resistance is significantly lower than 50.6.

Then, repeat this process but enter 49.4 as the value for the hypothesized mean. This time interpret the *p*-value associated with the upper-tailed test, labeled Prob > t, to decide whether to reject H_0. The report on the right in *Figure 4* shows the *p*-value = 0.0448 (< 0.05), which indicates that there is enough evidence to say the mean value is significantly greater than 49.4.

These two tests together provide enough evidence to make the decision that the observed average DC resistance of 49.94 Ohm is equivalent to 50 Ohm because it is within the ± 0.6 Ohm interval.

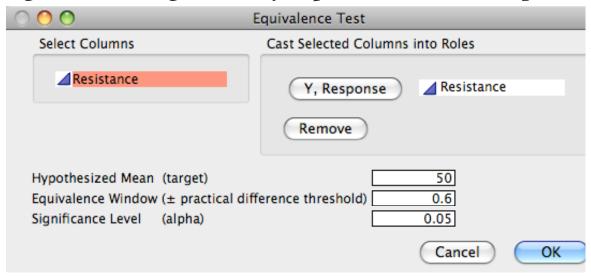
Figure 4 Two Means Tests For Equivalence Text



```
Hypothesized Value      50.6      Hypothesized Value      49.4
Actual Estimate      49.9399      Actual Estimate      49.9399
DF                        39      DF                        39
Std Dev              1.96118      Std Dev              1.96118
                 t Test                            t Test
Test Statistic   -2.1289          Test Statistic    1.7410
Prob > |t|       0.0396*          Prob > |t|        0.0896
Prob > t         0.9802           Prob > t          0.0448*
Prob < t         0.0198*          Prob < t          0.9552
```

Going one step further, there is a script available with the newsletter to perform the one-sample equivalence test. When you execute the script, the dialog shown in Figure 5 prompts you for the inputs to test.

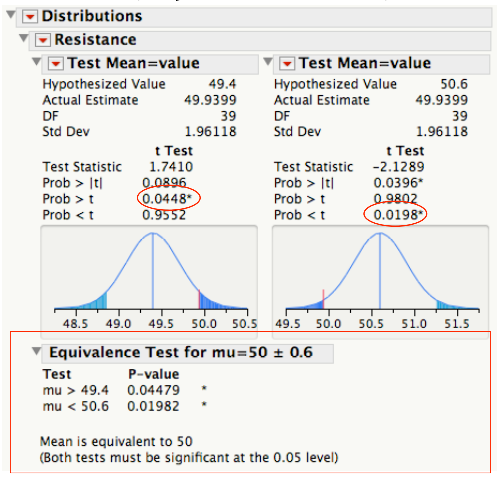
Figure 5 Dialog Given by Equivalence Test Script



Hypothesized Mean (target) 50
Equivalence Window (± practical difference threshold) 0.6
Significance Level (alpha) 0.05

The result (*Figure 6*) shows both tests and a message indicating that the average resistance is equivalent to 50 Ohm.

Figure 6 Results of Equivalence Test Script



```
Hypothesized Value      49.4      Hypothesized Value      50.6
Actual Estimate      49.9399      Actual Estimate      49.9399
DF                        39      DF                        39
Std Dev              1.96118      Std Dev              1.96118
                 t Test                            t Test
Test Statistic    1.7410          Test Statistic   -2.1289
Prob > |t|        0.0896          Prob > |t|       0.0396*
Prob > t          0.0448*         Prob > t         0.9802
Prob < t          0.9552          Prob < t         0.0198*

Equivalence Test for mu=50 ± 0.6
Test         P-value
mu > 49.4    0.04479   *
mu < 50.6    0.01982   *

Mean is equivalent to 50
(Both tests must be significant at the 0.05 level)
```

The equivalence test script launches the JMP Distribution platform and, by default, displays a histogram, and moments and quantiles tables. In addition, the script performs the two means tests for equivalence testing, and prints the conclusion giving by the tests. In *Figure 6*, platform options were used to hide the histogram and moments.

The equivalence test allows us to say, with statistical rigor, that |Average DC Resistance – 50| ≤ 0.6 Ohm. In other words, these simple tests show that the average resistance of the sample of 40 cables is *functionally* equivalent, as defined by the ± 0.6 Ohm window (equivalence bound), to 50 Ohm.

Defining Equivalence

It is up to the experimenter to decide what constitutes equivalence, and thereby define the functionally equivalent window of performance. When defining this window there are several options to consider.

1. Define the equivalence window as ± 3 × measurement error.
   This option takes into account the measurement error by making the window a function of it. The definition of equivalence should stay in line with what measurement systems can detect.
2. The equivalence bound should be much less than the process variation.
   Every process has natural process variation that defines what the performance that can be expected. If the process is stable this variation does not change much and can be used as a guide to define the equivalence window.
3. The equivalence bound should be much less than specifications.
   Ideally, specifications define fitness-for-use and as such, determine the acceptable performance of a product, material or process. The specification window can guide the definition of the equivalence window.
4. The equivalence bound should be related to functional performance.
   How small a difference can be tolerated before the observed performance is considered different?

Notice that this common application can be generalized to other situations: showing that two or more population means are equivalent, two slopes or model parameters, in general, are equivalent, and so on.

Look for more articles about applications of equivalence tests in future issues of the JMPer Cable newsletter. For more information about how to determine equivalence bounds, equivalence tests, and the theory of equivalence, tests see the references listed below.

References

Ramírez, José G., and Ramírez, Brenda S (2009) *Analyzing and Interpreting Continuous Data Using JMP: A Step-by-Step Guide*, Cary, NC: SAS Institute Inc.

Berger, Roger L., and Hsu, Jason C. (1996) Bioequivalence Trials, Intersection-Union Tests, and Equivalence Confidence Sets, Statistical Science 11, 283-319.

About the Authors

We are industrial statisticians each with more than fifteen years of experience working closely with engineers and scientists to help them 'make sense of data'. We view statistics, combined with a powerful visualization soft-ware, as a catalyst for discoveries and insights that help bring new products to market, sustain manufacturing operations, and guide process improvements. We are avid users of JMP and SAS. Visit our blog at

http://statinsights.blogspot.com/

The New Heroes of Commerce?

Chuck Pirrello, JMP Division of SAS Institute

A growing number of heroes are rising up from their cubicles at many companies. They provide that sought after competitive edge in these trying economic times. Yet, only the most enlightened companies fully embrace them. In fact, most people find them intimidating.

Who are these heroes of commerce, praised by some and feared by others? They are The Statisticians. Some have described statisticians as numbers people who "don't have the personality to be accountants." But statisticians are becoming valued for their work in a discipline that causes many an executive's eyes to glaze over.

According to Ian Ayres, author of the book *Super Crunchers* (2007) "there's a war going on between traditional experts, who make decisions based on limited historical data and intuition," and a new breed of statistician, whom he calls Super Crunchers.

Super Crunchers crunch large (VERY LARGE) databases to find trends and patterns in their organizations' data and produce golden nuggets of insight from everything ranging from their customers' buying habits and profitability to waste in their value chain.

Businesses are starting to understand the power of statistics to support their critical decisions. Never before has it been possible to analyze such large databases and quickly discover information that can have a vast impact on an organization and its customers.

One of my favorite examples from Ayres' book is that of an online travel site called Farecast.com. This site was started by a professor who was upset to find that plane passengers sitting right next to him had paid far less for their tickets than he had paid. He decided to set up a web site and use statistics to predict whether fares were trending up or down.

When you go to Farecast.com and search for airfares to your desired destination, it tells you whether the fares are expected to rise or fall over the next seven days. The amount of data it crunches is massive. Not only does it factor in historical prices, but also the time of year you are traveling, whether it's during a holiday, or if there is a big event at your destination like the Super Bowl or Mardi Gras. Farecast can even displays how confident it is in that prediction. Because it can predict where fares are headed, Farecast.com has obtained a superior competitive advantage.

Google is another great example of using predictive statistics on huge amounts of data. If Bill Gates searches using the word "Blackberry," he'll get a list of technology sites about RIM's BlackBerry. But if Martha Stewart enters the same search word, Google is likely to show her food or cooking sites.

So what does it mean for more companies to incorporate statistics in their business decision-making? With help from statisticians, they can to crunch more data than ever before, interpret the results, and predict more confidently. It's possible for a company to make decisions that focus on their operations, increase market share, and make them more profitable. And maybe statisticians will start to be valued for their brains and talent rather than being just a stereotyped personality.

Reference

Ayres, Ian (2007), *Super Crunchers: why thinking by numbers is the new way to be smart*, New York: Bantam Dell, A Division of Random House, Inc.

Meet Chuck Pirrello

Chuck is a Product Manager at JMP, where he drives the development and marketing of JMP to the business community, focusing on JMP's capabilities for interactive data visualization and exploration.

Quick Tip

Q: I am doing a cell plot for dozens of categorical columns from survey data. The columns all have the same categories, but not all columns have all the categories, so the cell plot colors them inconsistently. How can I make the colors consistent?

A: Cell Plot has a **Scale Uniformly** option, but this applies only to numeric variables. For category variables, use Column Properties as follows.

1. Highlight the categorical columns you want to use for the cell plot.
2. Select **Cols** > **Standardize Attributes**.
3. In the window that appears, select **Value Colors** from the **Column Properties** menu.
4. Customize the colors as desired, and click **OK**.

The next Cell Plot on this data presents the columns consistently, using the value colors you specified.

Instrument Measurement Linearity

Tonya Mauldin, JMP Division of SAS Institute

If you work with instruments that take measurements, there can be a myriad of potential problems. Some of these problems are with the instrument itself, such as:

- Need for calibration
- Damaged instrument
- Poor quality instrument
- Need for cleaning

What is a linearity study?

One tool that can help uncover problems like those listed above is a linearity study. A linearity study tracks the variation between measurements to a known standard throughout the expected operating range. A well working and appropriately used gauge (measuring device) should have constant bias throughout the expected operating range. Said another way, you expect that the measuring device can measure small values and large values with the same amount of accuracy.

How to perform a linearity study

To perform a linearity study, you need two basic pieces of information:

- Known standards or truths
- Measurements from the device that cover the full operating range

The *Measurement Systems Analysis Reference Manual* (MSA) third edition (2002) suggests obtaining a minimum of five different parts whose measurements span the operating range. These parts are what you measure for the linearity study. They can be different sized washers, concentration, a pH value, etc. A reference or standard value must be determined for these five parts.

Some possible ways of determining reference or standard values are to

- use devices in your master tool room
- send out the samples to another lab that has better precision than your lab
- obtain reference standards from the National Institute of Standards and Technology (NIST)

Each of the five parts should be measured at least ten times on the gauge of interest by your expert operator. An expert operator is one that uses this gauge on a regular basis and obtains the most accurate measurements. The parts should be randomly selected to minimize bias.

Example

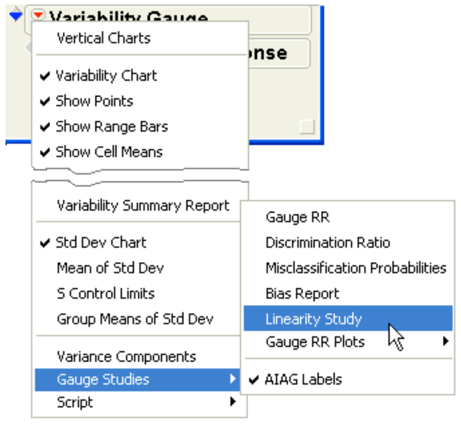
Suppose a new measurement system is under consideration for a manufacturing plant that produces washers. The goal is to determine if the new gauge accurately measures washers (parts) across the operating range. Five differently sized washers were chosen throughout the operating range of the measurement system. A reference value was determined for each washer using a layout inspection. Each washer was then measured twelve times by the expert operator using the new gauge. The washers were randomly selected in order to minimize bias. The old measurement system that is currently in place has a known sigma of 2.75613.

To see the results of this example, open the JMP table named MSALinearity.jmp, found in the Variability folder in the Sample Data installed with JMP. To perform a linearity study, do the following

1. Select **Graph** > **Variability/Gauge Chart**.
2. In the launch window, select Response as the **Y, Response** variable.
3. Select Standard as **Standard**.
4. Select Part as **X**, **Grouping**.
5. Click **OK**.

When the results appear, click the red triangle on the Variability Gauge title bar and select **Gauge Studies** > **Linearity Study**. (*Figure 1*)

Figure 1 Select Gauge Studies > Linearity Study



You are then prompted to specify the *Process Variation*, which is used to compute linearity. The value for process variation is 6*historical sigma (6 * 2.75613 = 16.5368). Therefore, enter 16.5368 into the box for process variation and click **OK**. You should see the results shown in *Figure 2*.

Interpreting the results

A linearity study is a linear regression analysis using the standard (reference) variable as the X variable, and bias as the Y variable. You hope to see a slope of 0 giving a horizontal regression line. A slope of 0 means there is no relationship between the size of the washer (part) and the ability to measure the washer.

The regression line in *Figure 2* does not appear horizontal and the *p*-value (Prob > |t|) for the test that the slope is 0 is less than 0.0001. This means you reject the null hypothesis and conclude that the slope is not 0. In practical terms, this means that there is a relationship between the size of the washer and the ability to measure the washer.

In this example, the absolute bias is greatest at the two extremes of the operating range (2mm and 10mm). The bias for the smallest washer (Part 1, 2mm) is a positive bias of 0.49167 and the bias for the largest washer (Part 5, 10 mm) is the negative bias of –0.61667. The smallest absolute bias occurs in the middle of the operating range (Part 3, 6mm) with a bias of 0.02500.

This new gauge is good at measuring washers in the middle of the operating region (6mm), but not very good at measuring washers that are at the extremes of the operating region (2mm and 10mm). It appears that the measuring device needs to be recalibrated to achieve near zero bias across the operating range before the new measurement system can be implemented.

References

SAS Institute Inc. (2008), *JMP Statistics and Graphics Guide*, Cary, NC: SAS Institute Inc.

ASQC Automotive Division/AIAG (2002), Measurement Systems Analysis Reference Manual, 3rd Edition, AIAG.

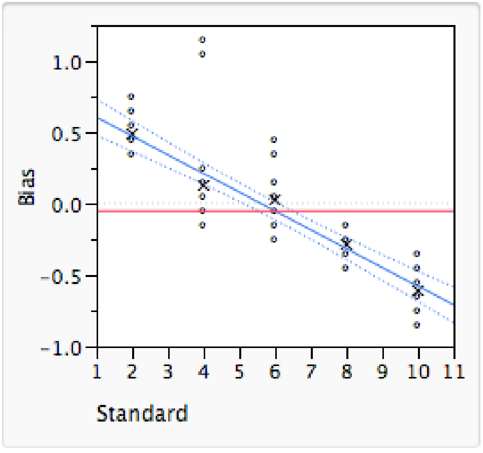
Figure 2 Results of the Linearity Study for the Washer Measurement Data

Variability Gauge

Variability Chart for Response

Measurement Bias Report

Linearity Study



Bias = 0.7366667 - 0.1316667 Standard

Standard	Avg Response	Bias	Lower CL	Upper CL
2.0000	2.49167	0.49167	0.36612	0.58055
4.0000	4.12500	0.12500	0.13419	0.28581
6.0000	6.02500	0.02500	-0.11524	0.00857
8.0000	7.70833	-0.29167	-0.39248	-0.24085
10.0000	9.38333	-0.61667	-0.68722	-0.47278

Source	DF	Sum of Squares	Mean Square	F Ratio
Model	1	8.321333	8.32133	145.0232
Error	58	3.328000	0.05738	**Prob > F**
C. Total	59	11.649333		<.0001*

	Estimate	Std Error	t Ratio	Prob>|t|
Intercept	0.73667	0.072524	10.158	<.0001*
Slope	-0.13167	0.010933	-12.043	<.0001*

		Which equals
Linearity	-0.815	Slope * Process Variation
% Linearity	13.167	100 * abs(Slope)
Avg Bias	-0.053	Bias averaged over all parts
% Bias	0.862	100 * AvgBias / Process Variation
Process Variation	6.191	Entered on dialog
R-Squared	0.714	
Alpha	0.050	

What about 3-D Pie Charts?

Russ Wolfinger, JMP Genomics Division, SAS Institute

Many of us are faithful left-brained statisticians and scientists who make every attempt to adhere to the highest professional standards of data visualization and analysis. We acknowledge that graphics luminaries like Edward Tufte and Stephen Few have made very valuable contributions to the field. We know the commandments from the graphics experts: Keep it simple—Avoid chart junk—Let the data shine through—Favor linear over spatial comparisons—Eschew volumetric distortion—Wield Ockham's Razor—and so on. 3-D pie charts are the worst offenders and have long ago been banished to graph purgatory.

Figure 1 compares a standard bar chart to a 3-D pie chart. The purpose of both charts is to quickly and effectively convey the dominant sources of variation in a microarray experiment. The script to create 3-D pie charts was created by a summer intern, Jong-Seok Lee and is included with this newsletter on our web site.

Without doubt, the bar chart has more detail and nicely uses linear instead of spatial comparison. It's a great graph and in fact is the default one shown for such analyses in JMP Genomics.

Pie charts are considered technically inferior, so why won't they go away? And, why would a 3-D pie chart be appealing to some people? The pie chart below does use color aggregation, has 3-D flair, and labels its major categories. In addition, there is interactivity. It's a spinnable graph that comes complete with slider bars that let you adjust degree of explosion and shininess. You can download the script and try it for yourself.

Perhaps the pie chart appeal has something to do with philosophical presuppositions. Dutch philosopher Herman Dooyeweerd and colleagues have extensively discussed 14 Aspects of Reality arranged in a specific order:

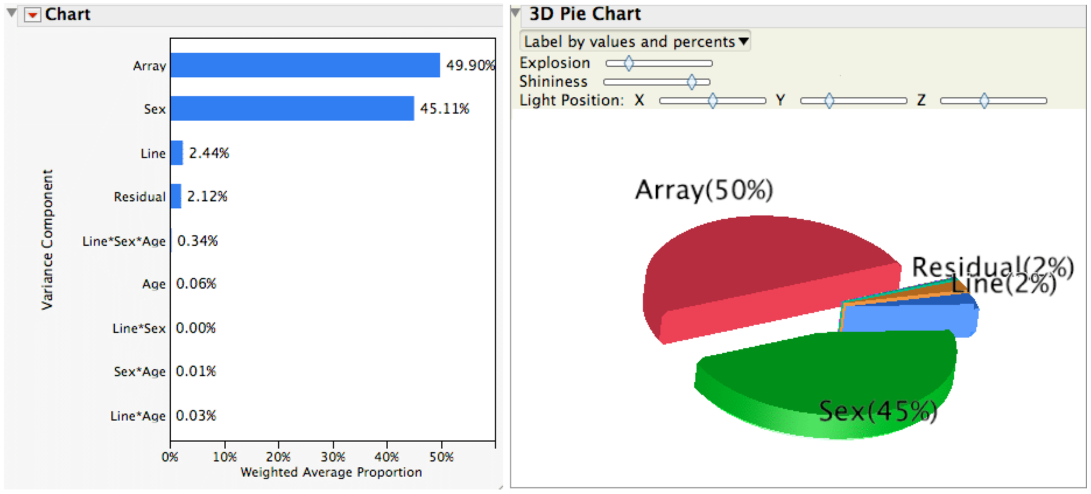
1. Numerical
2. Spatial
3. Kinematic
4. Physical
5. Biotical
6. Sensitive-psychical
7. Logical
8. Cultural-historical
9. Social
10. Economical
11. Aesthetical
12. Juridicial
13. Ethical
14. Fiducial

They make a convincing case that these ordered aspects are irreducible in the sense that you cannot eliminate any of them without getting into irrecoverable binds and self-refuting contradictions. Furthermore, nearly all philosophical conflicts throughout history have arisen from different attempts to make one of these aspects the divine/ultimate one upon which all others depend. Such reductions have often turned Ockham's Razor into Sweeney Todd's.

With reference to the bar and pie charts in *Figure 1*, the bar chart relies primarily on the numerical, spatial and logical aspects, whereas the interactive pie chart adds aesthetics and kinematics. These latter two aspects make a big difference and enable the pie chart to connect with the viewer on more levels.

We're naturally drawn to things that are beautiful and exhibit pleasing colors, symmetry, and interactivity. We travel the world to engage with captivating wonders and works of art, both natural and man-made. We often reward business professionals and

Figure 1 Comparison of Bar Chart and 3-D Pie Chart



politicians who build their careers not on the substance of their message, but by the elegance and flair with which they convey it.

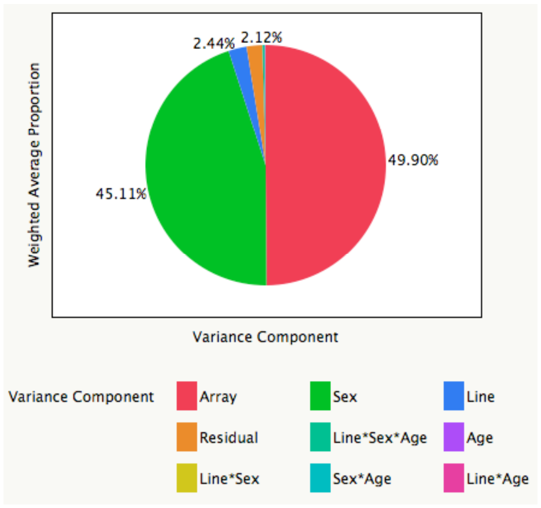
The pie chart also offers a biotic connection to various round delectables—mom's apple pie, pizza, cheesecake, and even quiche. It appears that we are environmentally conditioned to be sorely tempted by the 3-D pie chart, however evil it might be.

You can read this article as a blog on our web site, along with a plethora of replies, and reply yourself if you want to.

The blog commenters were merciless and the dialog is interesting. Here is an excerpt:
"I have to disagree strenuously with this article. 3-D graphics are misleading. …you gloss over the 'volumetric distortion' without ever noting that this makes 3-D pie charts misleading. I note that you failed to compare the bar chart to the 2-D pie chart, which also does a better job than the bar chart; nor did you compare your 3-D pie chart to a 2-D pie chart. In either case, the 2-D pie chart wins for effective communications (see *Figure 2*). 3-D charts are a triumph of form over substance."

In the words of another commenter:
"…A simple bar graph that is sorted and colored can produce useful insight. You can easily see 4 groupings: the 2 at about 50%, the 2 at about 2%, the 2 at about 2%, the 2 at about .2% and the 3 at 0%. You can also visually see the 5% difference between the top two (See *Figure 3*). Can you see these things looking at your 3-D pie chart?"

Figure 2 2-D Pie Chart



One other comment deserves an explanation:

"I use JMP software almost every day and I love it. It is the Swiss army knife of analytics software. But in one particularly important area, it falls flat, and I think that shortfall may have led you to wander down the dark path to the pie chart. In JMP it is not possible to easily sort a bar chart by the values of the bars."

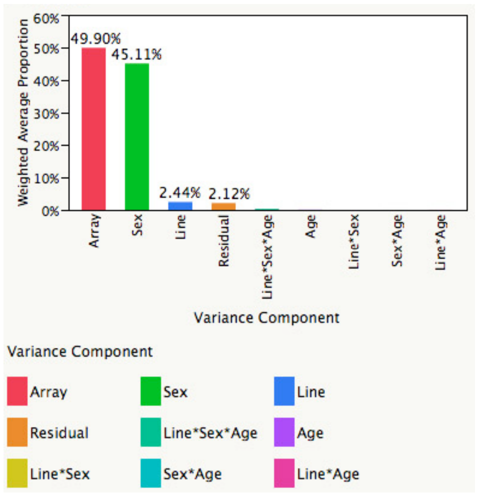
Yes it is possible and not difficult! *Figure 3* shows a bar chart done using **Graph** > **Chart**. It is sorted and colored by Variance Component. For character columns (such as the Variance Component column), this is done by assigning column properties.

- Use **Cols** > **Column Info** for Variance Component.
- select **Value Ordering** from the Properties menu on the Column Info dialog, and order the values any way you want them to appear in plots and charts.
- Use the **Value Ordering** property to color the bars. There is a default set of colors or you can choose your own colors.

If there are too many rows to arrange quickly in a Value Ordering list, first use **Tables** > **Sort** to sort by Weighted Average and then apply the **Row Order** column property to preserve the table order of the rows in the bar chart.

The 2-D bar and pie charts are indeed effective presentations of the data. However, in this particular example the bar chart doesn't really need color to effectively convey its information. The mono-color bar chart in *Figure 1* is sufficiently informative and esthetically neater.

Figure 3 Colored Sorted Bar Chart



Expression Handling Functions: Part I
Unraveling the Expr(), NameExpr(), Eval(), ... Conundrum

Joseph Morgan, JMP Division of SAS Institute

Many beginning and intermediate JMP Scripting Language (JSL) programmers are unaware of the power of abstraction available from JSL expressions. Such meta-programming constructs are not always available in widely used programming languages such as C++ but are commonly found in functional programming languages such as Lisp. As it turns out, such constructs are particularly useful when the application being developed is complex. They facilitate process abstraction. Robert Sebesta (1999) describes abstraction:

> "The ability to define and then use complicated structures or operations in ways that allow many of the details to be ignored. The degree of abstraction allowed and the naturalness of its expression is important."

This article attempts to unravel the mystery surrounding JSL expression handling functions and show how such functions can be used to solve nontrivial JSL programming challenges.

JSL Expressions

What exactly is a JSL expression? Chapter 3 of the JMP Scripting Guide (JSG) defines JSL expressions thus:

> "A JSL expression is any combination of variables, constants, and functions linked by operators that can be evaluated."

The key phrase here is "... *that can be evaluated.*" This means that each of the following is a JSL expression.

```
100.1                 //numeric literal
"string literal"      //string literal
x                     //variable (or name)
x & (y | z)           //logical expression
z*2 + z^2 -10 + pi() //arithmetic expression
```

However, more complex examples like the following are also JSL expressions.

```
x = [];
for(i=1, i<=5, i++,
   x ||= random uniform(); show(x)
)
```

Although the term *script* is often used to refer to an example like this, it is really just an expression. Remember that the semicolon ";" is the *glue* operator that returns the value of its right-most argument. A script is nothing more than a single `glue()` function call with expressions as its arguments. To see this, notice that the previous example is equivalent to the following `glue()` function call.

```
glue(assign(x, []),
   for(assign(i, 1),
      less or equal(i, 5),
      post increment(i),
      glue(concat to( x,
                  random uniform()
                  ),
         show(x)
      )
   )
)
```

Hence, a JSL expression may be as simple as a literal or variable, but could be as complex as a script.

What is an Expression Handling Function?

> A useful way to think of expression handling functions is as the set of JSL functions that enables you to regard expressions as data.

Functions such as `Expr()`, `NameExpr()`, `Eval()`, `Function()`, and `Recurse()` allow you to assign expressions to variables for later retrieval and possible evaluation. There are also functions that allow expressions to be assembled, disassembled, and probed. `Insert()` and `Remove()` are two of several functions that may be used to assemble and disassemble expressions whereas `Arg()` and `Head()` are intended for probing. JMP offers a full complement of these functions thus ensuring that JSL programmers can easily realize the abstraction by Sebesta (1999).

These functions (see *Table 1*) fall into two categories: those that evaluate their arguments when invoked and those that do not. The best way to understand this difference is to experiment with these functions. To follow along, launch JMP and run the code fragments presented in the following sections.

Table 1 JSL Expression-Handling Functions

Evaluate Arguments	Do Not Evaluate Arguments
`Parse()`	`Expr()`
`Eval()`	`NameExpr()`
`EvalList()`	`EvalExpr()`
`Function()`	`Arg()`
`Recurse()`	`NArg()`
`Substitute()/SubstituteInto()`	`Head()`
`Remove()/RemoveFrom()`	`HeadName()`
`Insert()/InsertInto()`	

Expression Handling by Example

The following questions were real problems presented by JSL programmers who had a task they were trying to complete. These challenges are not intended to represent the range of questions a typical JSL programmer is likely to face, but they comprise a series of typical and commonly encountered questions.

1. The Substitute() vs. SubstituteInto() Question

Suppose you want to write a script that invokes the distribution platform but the column to be analyzed is stored in a variable. In cases like this, the `Substitute()` or `SubstituteInto()` function may be used but it is sometimes not clear which one should be used.

For example, the following script uses `Substitute()` to replace `colx`, with `weight`, but fails.

```
//script 1
stmt = Expr(distribution(column(colx)));
x = "weight";
Result = Substitute(stmt, Expr(colx), x);
show(stmt); show(Result);
```

If you execute this script, the log shows:

```
Not Found in access or evaluation of 'distribution' ,
Bad Argument( {colx} ), distribution( Column( colx ) )
```

Because `Substitute()` evaluates its arguments, it attempts to evaluate `stmt`, but fails because `colx` does not exist. One solution is to properly *quote* the first argument of `Substitute()`. That is, use `NameExpr()` to retrieve the value of `stmt`.

```
//script 1 - revised
stmt = Expr(distribution(column(colx)));
x = "weight";
Result = Substitute(NameExpr(stmt),Expr(colx),x);
show(stmt); show(Result);
```

Now, execute this revised script to see the value of `stmt` and `Result` displayed in the log.

```
stmt:distribution(Column(colx))
Result:distribution(Column("weight"))
```

Alternatively, `SubstituteInto()` may be used. The difference is that, unlike `Substitute()`, `SubstituteInto()` does not evaluate its first argument but simply updates it in place.

```
//script 2
stmt = Expr(distribution(column(colx)));
x = "weight";
SubstituteInto(stmt,Expr(colx), x);
show(stmt);
```

When you execute this script, the result in the log is

```
stmt:distribution(Column("weight"))
```

Summary Points

Point 1:

A common JSL mistake is to assume that executing `Expr(x)` is equivalent to executing `NameExpr(x)`. Indeed, in the following example, these two statements return the same thing.

```
Expr(4 + 35)
NameExpr(4 + 35);
```

If you execute them one at a time, the log shows,

```
Expr(4 + 35);
4 + 35
NameExpr(4 + 35);
4 + 35
```

The result is the same for both statements. `Expr(x)` returns *its argument unevaluated* and `NameExpr(x)` returns *the value of its argument unevaluated*. The argument to `NameExpr(x)` should be a variable, but when it is an expression it simply returns its argument.

Consider the next statement.

```
x = Expr(2 + 50);
```

When you execute this statement the expression **2 + 50** will be stored in x.

Now consider the following statements.

```
Expr(x);
NameExpr(x);
```

Execute each statement and look at the log.

```
Expr(x);
x
NameExpr(x);
2 + 50
```

Since `Expr()` returns its *argument unevaluated*, the name `x` is returned, whereas `NameExpr(x)` returns the *value of its argument unevaluated* — **2 + 50**.

***Point*: Executing `Expr(x)` is not equivalent to executing `NameExpr(x)`.**

2. Obtaining Distinct Items From a List

Suppose you have a sorted list and want to retrieve only distinct items. There is no JSL function to accomplish this, but it is easy to script a solution.

Consider the following two lists.

```
Things = {"apple", "apple", "apple", "cat", "cat", "cat",
   "golden", "grape", "mango", "mango", "silver",
   "silver"};
Numbers = {1,200,200,200,400,400};
```

One approach is to iterate over items in each list and pick out the distinct items as the iteration progresses. However, here is an alternative and compact solution that illustrates the `EvalList()` function.

```
indx = {};
indx[1 :: NItems(Things )] =
   Expr( 0 == i++ | Things [i - 1] != Things [i] );
   i = 0;
distinctlst = Things [Loc( EvalList( indx ), 1 )];
```

Note that the second statement creates a list of logical expressions and that this list contains the same number of items as the sorted list. Each expression is intended to compare the corresponding entry in the sorted list to the item at its left. When evaluated (by `EvalList()` in the fourth statement), each expression in the list evaluates to either true or false. The `Loc()` function in the fourth statement converts this list of `0`s and `1`s into a vector of indices that retrieves the distinct items.

The following function is a more robust solution.

```
distinct list = Function( {lst},
   Local( {indx = {}, i = 0},
      If( Is List( lst ),
         If( N Items( lst ) < 2,
               lst,
               indx[1 :: NItems( lst )] =
               Expr( 0 == i++ | lst[i - 1] != lst[i] );
               lst[Loc( EvalList( indx ), 1 )];
            )
         )
      )
   );
```

Calling the function with the list as its argument gives the following unique items.

```
Distinct List(Things);
{"apple", "cat", "golden", "grape", "mango", "silver"}
Distinct List(Numbers);
{1, 200, 400}
```

Summary Points

Point 2:

When using the `Eval()` function, a common mistake is to assume that executing `Eval( x )` is equivalent to executing `x`. This mistake can be easily made if you examine examples like the one below, where the second and third statements produce the same results.

```
x = Expr(4 + 25);
x;
Eval( x );
```

The first statement stores the expression `4 + 25` in `x`. If you execute the second and third statements in turn, you see the following in the log.

```
x;
29
Eval( x );
29
```

However, what if the first statement was a nested `Expr()` function as in the example below.

```
x = Expr(Expr(4 + 25));
x;
Eval( x );
```

Note that, for this example, the first statement stores the expression `Expr(4 + 25)` in x. If you execute the second and third statements in turn, you see the following in the log.

```
x;
4 + 25
Eval ( x );
29
```

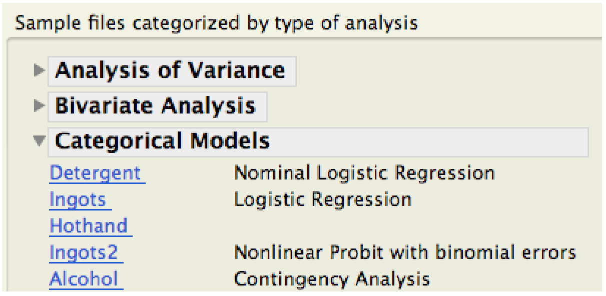
The results are now different.

Point**: Executing `Eval( x )` is not equivalent to executing `x`.**

3. The Literal Argument Challenge

Suppose you are interested in creating a dialog that contains several outline nodes, each of which contains hyperlinks to different data tables (see Figure 1). The *Sample Data Index* found in the JMP **Help** menu is an example of such a dialog.

Figure 1 Outline Nodes in Sample Data Directory



Sample files categorized by type of analysis

▶ Analysis of Variance
▶ Bivariate Analysis
▼ Categorical Models

Detergent	Nominal Logistic Regression
Ingots	Logistic Regression
Hothand	
Ingots2	Nonlinear Probit with binomial errors
Alcohol	Contingency Analysis

The following script illustrates how one of these outline nodes could be built, using the JMP sample data index as the example.

```
//Brute Force Method
New Window( "Sample Directory",
Outline Box( "Categorical Models",
Lineup Box( N Col(2), Spacing(0),
Button Box("Detergent",underline style(1),
Open( "$SAMPLE_DATA/Detergent.jmp" )),
Text Box( "Nominal Logistic Regression"),
Button Box( "Ingots2", underline style(1),
Open( "$SAMPLE_DATA/Ingots2.jmp" )),
Text Box( "Logistic Regression" ),
)));
```

This approach rapidly becomes unwieldy when adding statements to construct more and more outline nodes, each with multiple buttons. Instead, imagine a different approach where the script iterates over a list of outline node titles, data table names, and descriptions. As it iterates over the list, it constructs the corresponding dialog.

The following list of lists is for a two-node dialog.

```
// create a list of lists
sample = {
{"Anova",
    {"Blood Pressure", "Multiple Repeated
    Measures"},
    {"Typing Data", "1-way Anova"}
},
{"Categorical Models",
    {"Detergent", "Logistic Regression"},
    {"Ingots2", "Nonlinear Probit Analysis"}}
};
```

Notice that each inner list consists of an initial entry, which is the outline node title. It is followed by several lists of pairs, where the first item is the data table name, and the second item is a table description.

The following function builds the sample file dialog. The `addnode()` function takes two arguments: the first is a reference to a dialog box, and the second is a list. The `addnode()` function is a nested loop that iterates through the list and creates an outline node from the first entry in each inner list. For each inner list pair, it creates a button box with an associated `open()` script, along with the text box that provides the button description. .

```
//Function to build sample file dialog
addnode = Function( {ref, lst},
    For( x = 1, x <= N Items( lst ), x++,
    ref << append( Outline Box( lst[x][1],
    lbx = Lineup Box( N Col( 2 ),
    spacing( 0 )) ) );
    For( y = 2, y <= N Items( lst[x] ), y++,
      table = "$SAMPLE_DATA/" ||
         lst[x][y][1] || ".jmp";
         cmd = Expr( lbx << append( bbx =
         Button Box( lst[x][y][1],
            Open( Expr( table ) ) ) )
         );
            Eval( EvalExpr( cmd ) );
         bbx << underlinestyle;
         lbx << append( Text Box( lst[x][y][2] )
    );
    );
    )
);
```

To start, you need to first create a skeleton dialog to contain the outline nodes and then `addnode()` is invoked.

```
//Create a panel box to contain nodes
New Window( "Sample Files",
    pbx = Panel Box( "Files categorized by
    analysis" ));
//Invoke Sample file function
addnode( pbx, sample );
```

Note that the `append(Button Box(...))` message has been cast as an expression, and that this expression contains a sub-expression, `Expr(table)`. When `Eval Expr( cmd )` is evaluated, `Expr(table)` is replaced with its value and, as a result, the value of `table` at the time of button creation is preserved.

The ‘literal argument challenge’ in this script occurs in the way the `append(Button Box(...))` message is written. A common mistake is to write the statement thus:

```
lbx << append( bbx = Button Box( lst[x][y][1],
    Open( table ) ) );
```

instead of the correct expression in the script,

```
cmd = Expr( lbx << append( bbx = Button Box(
    lst[x][y][1],
    Open( Expr( table )))));
```

Although the first statement appears to work, each button actually opens the same data table. In fact, that button always open `Ingots2.jmp`, which happens to be the last data table in the example list. The problem is the `table` variable providing the name for each button. Although `table` contains the correct data table name when each button is created, its value after the dialog is created, and therefore when any button is clicked, will be the last value that was assigned to it.

Here is another correct option.

```
Eval( Substitute(
    Expr( lbx << append( bbx = Button Box( lst[x][y][1],
    Open( xxx ) ) ) ),
    Expr( xxx ), NameExpr( table )));
```

For this solution, the `append(Button Box(...))` message has also been cast as an expression, but it is used here as the first argument of `Substitute()`. Recall that `Substitute()` evaluates its arguments and `NameExpr()` returns the value of its argument unevaluated. So, each time this statement is executed, `Substitute()` returns the value of its first argument but with the value of `table` in place of the pattern *`xxx`*. Therefore, the effect is the same as the correct solution shown previously.

Concluding Comments

The primary purpose of these examples is to illustrate the use of several expression-handling functions. A secondary purpose is to point out common errors and misunderstandings that JSL programmers sometimes experience when attempting to use these functions. Hopefully, we have partly achieved that objective.

Reference

SAS Institute, Inc. (2008), JMP Scripting Guide, Cary, NC: SAS Institute, Inc.

Sebesta, Robert M. (1999), Concepts of Programming Languages, Addison Wesley, Reading, MA.

Summary Points

Point 3:

Remember that `EvalExpr()` does not evaluate its argument. It clones its argument and replaces any `Expr()` sub-expressions with their evaluated values. Consider this example.

```
 y = Expr (
    Distribution(
       Column(Expr("X" || Char(i)))
    )
 );
 i=3;
 x = NameExpr(y);
 EvalExpr(x);
```

As expected, statement 4 returns

```
Distribution(Column( "X3" )).
```

So, why not combine statement 3 and statement 4? That is, replace the two separate statements with:

```
    EvalExpr(NameExpr(y));
```

When this combined expression executes, `NameExpr(y)` is returned. Note that `EvalExpr()` does not evaluate `NameExpr(y)`; it simply clones it and, since `NameExpr(y)` does not itself contain `Expr()` sub-expressions, `NameExpr(y)` is returned as is.

Point: **`EvalExpr()` does not evaluate its argument.**

Point 4:

If you choose to nest `Eval()` functions, think carefully about how the combined statement will be evaluated. Since `Eval()` evaluates its argument and then evaluates the result, nesting `n Eval()` statements is not equivalent to n instances of `Eval()`. Consider the following example.

```
x = Expr(Expr(Expr(Expr(1 + 2
      ))));
    Eval(Eval(x));
    y = Eval(x);
    Eval(y);
```

Try these statements yourself, executing them one by one, and note the results in the log.

Point: **n nested `Eval()` statements is not equivalent to n `Eval()` statements.**

About *JMPer Cable*
A Technical Publication for JMP® Users
Issue 26 Winter 2010

Editor:
Ann Lehman

Winter 2010 Issue Contributors:
Xan Gregg, Mark Bailey, Tonya Mauldin, Joseph Morgan, Chuck Pirrello, José G. Ramírez, John Sall, Russ Wolfinger

Subscribe Today

If you enjoy receiving *JMPer Cable* in the mail, subscribe today! In the future, only subscribers will receive print copies of *JMPer Cable*. To avoid missing even one valuable issue, subscribe online at

www.jmp.com/jmpercable

You can also find *JMPer Cable* current and back issues, along with their data tables and scripts, online. It's all at

www.jmp.com/jmpercable

For additional user resources, go to

www.jmp.com/community

To order additional licenses of JMP, to learn about the most recent release, or to inquire about content of this JMPer Cable issue, call 1-877-594-6567