



Inside This Newsletter

Articles in this special issue of the JMP Per Cable focus on JMP Scripting Language (JSL) and its application in a variety of fields and circumstances.

Dynamic Illustrations in the Classroom p. 5

ANOM in JMP p. 7

Creating a List of Open Data Tables p. 10

Meet the Trainer: An Interview p. 11

Breaking News

Join us for the second annual JMP User Conference on June 7-8, 2005 at the SAS World Headquarters in Cary, North Carolina.

The conference brings together users of all levels and includes sessions for JMP novices. It will be followed by two days of training on June 9-10.

For details, see <http://support.sas.com/training/jmp/index.html>

Using SAS IOM Commands in JSL

Matt Flynn, SAS Institute

The SAS IOM commands available in JMP Scripting Language (JSL) make it possible to connect directly from within JMP to a SAS server to retrieve data for further analysis within JMP.

The script presented in this article focuses on library submission, highlighting the SAS IOM commands for connecting and executing SAS jobs, and returning the results back to JMP for further processing. For a complete, downloadable version of the script, with comments, see `Connect_to_SAS.jsl` at <http://www.jmp.com/news/jmpcable>.

Background

I had positive experiences connecting other PC desktop tools to remote SAS servers via SAS Integration Technologies (SAS/IT) and saw that with the addition of the IOM commands, it would be possible to achieve this using JSL. Automating the often laborious multi-step process of retrieving data would allow ease of use in performing statistical analysis in JMP.

About the Script

This script allows a Windows JMP environment to talk to a remote SAS server. It connects to external servers and retrieves data in a point-and-click

environment. The script enables an automated method of launching (possibly remote) SAS batch jobs, retrieving SAS datasets (which may be the results of those batch jobs or simply the latest update of required transactional data) and imports the data seamlessly into JMP for statistical analysis and reporting. It eliminates the tedious, manual process of connecting to servers, navigating to the data, retrieving the data (perhaps via FTP), bringing that data into JMP, and then performing an analysis. The script allows the analysis to receive full attention and concentration. Even better, the script can be shared with JMP users who are not JSL programmers, enabling them to easily retrieve the remote data they need.

The script presented in this article is specific to a machine that has access to a local copy of PC SAS or a remote UNIX server running SAS. The following example sets up the script to connect to the two SAS platforms, execute SAS batch jobs, retrieve data (possibly the results of those batch jobs), and pulls that data into JMP for either interactive analysis or to repeat pre-programmed analysis platforms.



Connecting to a SAS Server

The first step is to write the part of the script that asks the user to select between a local and a UNIX server:

```
result = Dialog( title( "Run a SAS batch job" ) ,
  vlist(
    lineup( 2 ,
      "Select Server" , server = ComboBox( "local" , "UNIX" ) ,
      hlist( Button( "OK" ) , Button( "Cancel" ) ) )
  )
);
```

This creates the window in Figure 1.

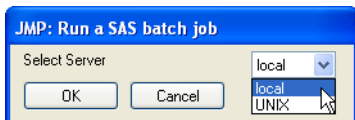


Figure 1: Ask the user to choose from available servers.

Now we want JMP to parse the results of the dialog box into variables. The following JSL does this:

```
show( result ) ;
server = result[ "server" ] ; show( server ) ;
button = result[ "Button" ] ; show( button ) ;
```

In this script, server 1 is the local server and server 2 is the remote server.

Furthermore, `button == 1` means that the user pressed OK and `button == -1` means that the user pressed Cancel. The JSL IOM command `sas connect("servername" , portID)` calls SAS on either the local machine or a remote UNIX host. Insert the appropriate name and port ID of your target SAS server here. The resulting dialog prompts for the appropriate userid and password:

```
if ( button == 1 ,
  // display user selections in the log
  if ( server == 1 ,
    show( "local" , server ) ,
    show( "UNIX" , server )
  );
  // connect to local or remote SAS instance
  if ( server == 1 ,
    sas connect( "localhost" , 8591 ) ,
    sas connect( "unix.sas.com" , 5310 )
  );
);
```

This creates the window in Figure 2.

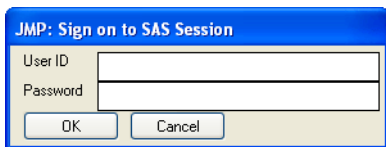


Figure 2: Checking User ID and password.

For verbal feedback, you could display a verbal and visual message to the user while

the SAS server starts up:

```
caption( { 400 , 400 } , spoken(
  1 ) , Delayed( 0 ) ,
  "Connecting to SAS
  server..." );
wait( 3 );
caption( remove );
```

Retrieving SAS Files

Next, the script retrieves a list of files in a certain directory. In this script the directory is hard-coded to look at specific directories, both local and on the remote server. You should assign the desired directory name for your needs. For example, the script looks at the local directory `C:\temp` and the remote UNIX home directory `~/temp`. You could generalize the script to branch based upon User ID, for example:

```
if ( server == 1 ,
  path = "C:\temp" ,
  path = "~/temp/"
);
show( "path" , path );

if ( server == 1 ,
  filelist = sas get file
  names in path( path ) ,
  filelist = sas get
  file names in
  path( "~/temp/" );
);
show( filelist );
```

Next, the script queries the directory and retrieves a list of files located in that directory. At this point, the list is limited to only the executable SAS program files available in this directory, so that the user can choose one to execute.

```
sasFiles = { };
for ( i = 1 , i <= n items(
  fileList ) , i++ ,
  len = length( fileList[ i ] )
);
if ( len > 4 ,
  if ( lowercase( substr(
    fileList[ i ] , len -
    3 ,
    len ) ) == ".sas" ,
```

```

        insertinto( sasFiles,
                    fileList[ i ] ) ) ;
    ) ;
) ;
show( sasFiles ) ;

```

Executing a SAS Job

Now display a dialog prompting the user to select a SAS program to run:

```

dlg = Dialog( "Select SAS program
             to run",
             vlist(
                 selection = listBox(
                     sasFiles ) ,
                 hlist( Button("OK"),
                       Button( "Cancel"
                               ) ) )
             ) ;
sel = dlg[ "selection" ] ; show(
sel ) ;

```

This creates the window in Figure 3.

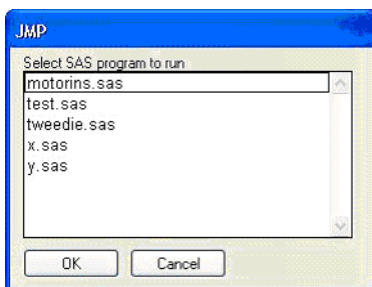


Figure 3: Select a SAS program to run.

Optionally Asking for Parameters for the SAS Job

For selected SAS programs, prompt for a user-specified parameter. If the user selects a specified program (in this case, the dialog is hard-coded to appear for the data file x.sas), the user is prompted to input parameters to add flexibility to the SAS batch job call. This one simply asks the user how many observations to process during the job.

```

if ( sel == {"x.sas"} ,
    result = Dialog( title(
"Please specify how many Obs to
run" ),
                    vlist(
                        lineup( 2 ,
                              "Set number of obs",
                              obs = EditNumber( 10 ) ,

```

```

                    hlist( Button( "OK" ) , Button( "Cancel" ) )
                    )
                    )
                    )
);

```

This creates the window in Figure 4.

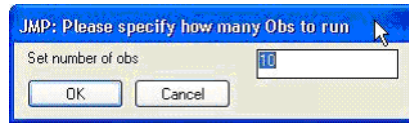


Figure 4: Optionally ask for parameters for the SAS job.

Next, the results of the dialog box are parsed into JSL variables:

```

show ( sel ) ;
if ( sel == {"x.sas"} ,
    obs = result[ "obs" ] ; show( obs )
);

```

If you're using JMP 5.1.1 and later, use the standard way to submit an existing SAS program file:

```
SAS Submit File( "C:\test\x.sas" ) ;
```

If you're using JMP 5.1 and earlier, use this workaround:

```
SAS Submit( "%include C:\test\x.sas" ) ;
```

It is a little tricky to pass the value of above selected program name and a parameter into the SAS submit string below. In this example, a SAS Macro command is built using JMP's concat function then submitted to SAS.

```

if ( sel == {"x.sas"} ,
    pgm = concat( "%let obs=", char(obs) , " ; %put *** obs = " ,
char(obs) , " ; " , "%include 'c:\temp\", sel[1] , " ' ; run ; " ) ,
    pgm = concat( "%include 'c:\temp\", sel[1] , " ' ; run ; " )
);
show(pgm) ;

```

Now submit the job with this command:

```
SAS Submit(pgm) ;
```

Retrieving the SAS job

Now that you have run the job, check for the success of the remote SAS job using the IOM command SAS Get Log():

```
log = SAS Get Log(); show(log);
```

The output can be retrieved into the JSL log using:

```
SAS Get Output();
```

You can specify details about opening the SAS data files. It's a good idea to first list all files in directory (some created in x.sas or y.sas), and then to trim the list to just SAS data files (*.sas7dcat).

```

filelist = sas get file names in path( path ) ; show( filelist ) ;
sasFiles = {} ;
for( i = 1 , i <= n items( fileList ) , i++ ,
    len = length( fileList[ i ] ) ;
    if ( len > 7,

```

```

if ( lowercase( substr( fileList[ i ] , len -7 , len ) ) ==
      "sas7bdat" ,
      insertinto( sasFiles , fileList[i] )
    ) ;
) ;
) ;

sasFiles = sort list( sasFiles ) ;

show( sasFiles ) ;

dlg2 = Dialog( "Select dataset to open" ,
  vlist(
    lineup( 3 ,
      selection = listBox( sasFiles ) ,
      vlist( Button("OK"), Button("Cancel") )
    )
  ) ;

sel2 = dlg2[ "selection" ] ; show( sel2 ) ;

show( path ) ;
show( server ) ;
if ( server == 1 ,
  dsn2 = concat( "C:\temp\" , sel2[1] ) ; show( dsn2 ) ; open(
    dsn2 ) ,
  dsn1 = concat( "~/temp/" , sel2[1] ) ; show( dsn1 ) ;
  dsn2 = concat( "C:\temp\" , sel2[1] ) ; show( dsn2 ) ;
  sas get file ( dsn1 , dsn2 ) ;
  open( dsn2 ) ;

) ;

show( path ) ; show( dsn2 ) ;

```

This creates the window in Figure 5.

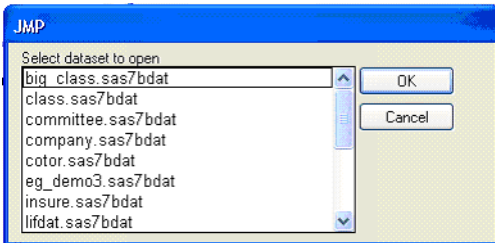


Figure 5: Display a list of SAS datasets (some could be output files of the SAS batch job).

Running a JMP Analysis

Now, run a JMP analysis on the SAS dataset after prompting the user to select analysis columns. This example shows running a one-way analysis.

```

Oneway( Each Pair( 1 ) , All Pairs( 1 ) , Quantiles( 1 ) , Means( 1 ) ,
Means and Std Dev(1), Unequal Variances(1), Box Plots( 1 ) ,
Mean Diamonds( 1 ) , Mean Error Bars( 1 ) , Std Dev Lines( 1 ) , Comparison
Circles( 1 ) , title( "Sample Oneway Analysis" ) ) ;

```

Note: Assignment of Y and X is omitted: Y(:Height), X (:Age)

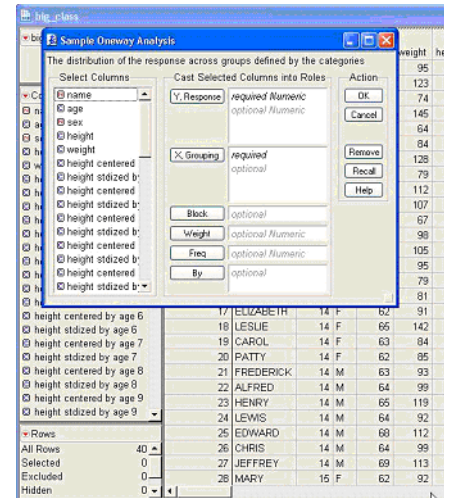


Figure 6: Resulting oneway dialog incorporating data retrieved from the remote SAS server.

Cleanup

Close the SAS session on the server when done.

```
SAS Disconnect( ) ;
```

Conclusion

JSL and the SAS IOM commands make it easy to “reach out” and retrieve data from across an organization, either as a nightly scheduled job, a query against a multi-million-row SAS dataset, or even large files in a DBMS, via SAS/ACCESS. One can easily import them into the friendly confines of JMP for either scripted or interactive analysis.

The author would like to thank JMP technical support for help creating a tricky part of the script that displays the available SAS program files so the user can choose one to execute.

Dynamic Illustrations in the Classroom

By Wayne J. Levin, JMP Instructor, Predictum Management Sciences

For students, understanding one statistical concept generally requires understanding several statistical concepts, since many of them are inextricably linked to others. Understanding power, for example, requires an understanding of standard errors, confidence intervals, analysis of variance, hypothesis tests, and F -tests.

Instructors may now dynamically demonstrate such statistical concepts using JMP Scripting Language (JSL) so students can visualize how statistical concepts behave and how they are related. Such visualization makes comprehension faster, easier, and more readily reinforced. This article

describes the use of one JSL script and how it shows linked statistical concepts.

The Script

The script used in this article, demoOneway.jsl, can be downloaded at <http://www.jmp.com/news/jmpercable>. Run the script to see the window in Figure 7, which illustrates the purpose of power and how power is linked to other statistical concepts. Download a detailed description of the window below (PowerANOVACheatSheet.pdf) from <http://www.jmp.com/news/jmpercable>.

Click and drag the sliders to make changes to the parameters. This causes

changes in the diamonds, the p -value, the power bars, and other areas of the illustration.

During class, instructors use this window to explain what other statistics “look like” when the sample size is 10. They see that power is low (0.45818) because the confidence intervals, as depicted by the height of the diamonds, are big. In other words, the variability overwhelms the difference in averages between the two groups. The conclusion is “Fail to Reject H_0 ” with a p -value of above 0.05, as shown on the p -value chart.

When the instructor slides the sample size slider from 10 to 30 (as shown in Figure 8), the window changes to

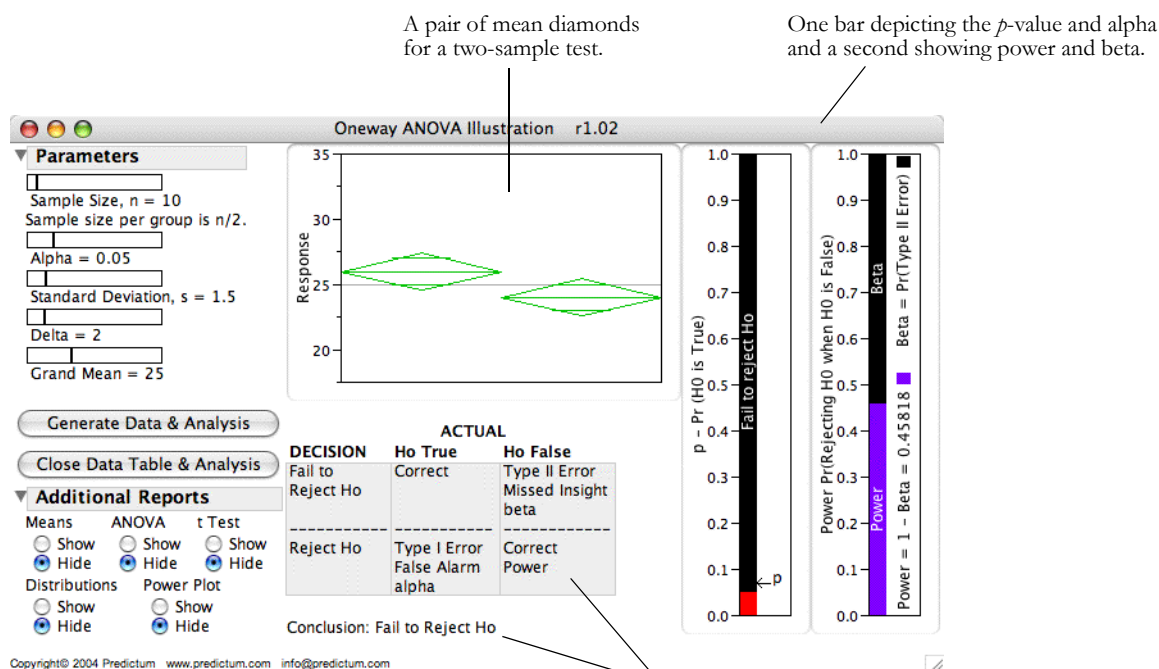


Figure 7: Window produced by demoOneway.jsl.

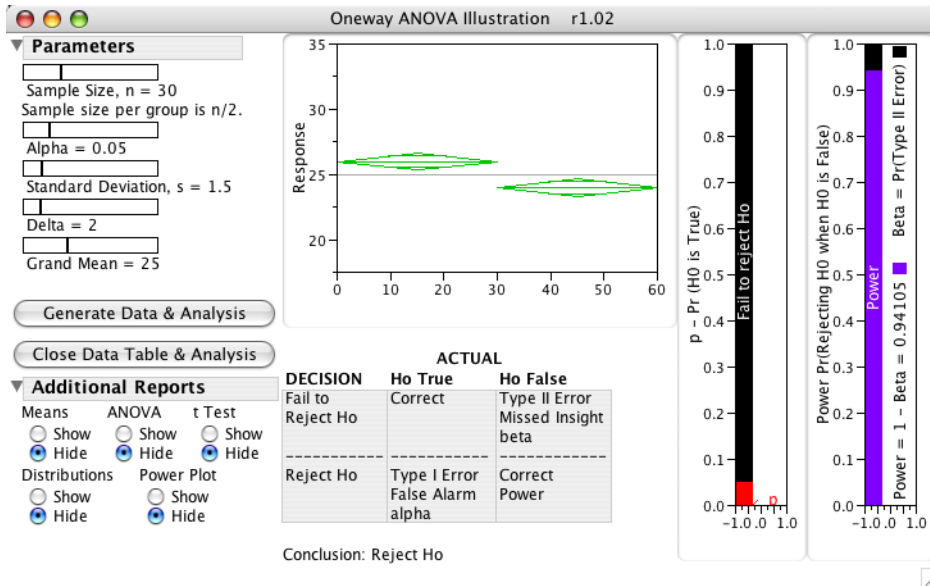


Figure 8: The window changes as the instructor moves the sample size slider from 10 to 30.

that when the standard deviation is high, there is little marginal gain in power from an additional observation. Reducing variability is its own reward, as seen on the right, because when the standard deviation is low, power rises considerably with each additional observation collected.

Conclusion

This script has been well received by students in my classroom. With it, students visualize concepts faster and easier—I can almost see the light bulbs going off in their minds. They are also eager to have copies of the script after the training session so they can reinforce their understanding. The script helps them apply these concepts to their analyses.

show the effects. Students see the diamonds shrink (the mean diamonds became tighter) as the standard error drops, the p -value descends, and power becomes stronger, occupying almost the entire bar on the right (0.94105). As the p -value drops below alpha, the conclusion line changes to “Reject H_0 .”

Similarly, the script also provides a power plot that adjusts in real time as the sliders are manipulated (Figure 10). Using the power plot, it is easy to see

Advanced Options

The script provides a set of additional reports to further clarify power. When the Show option is selected for ANOVA (Analysis of Variance), the script reveals an ANOVA report. Figure 9 shows ANOVA reports that contain a depiction of the F distribution under both the null and alternate hypotheses with a pointer to the current F -ratio.

Instructors can use the tables to compare sample sizes with all other parameters fixed. They can move the parameter sliders to update the table.

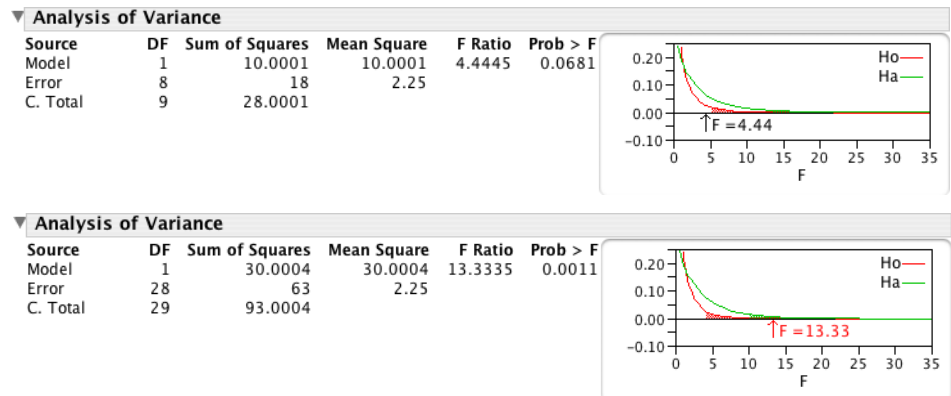


Figure 9: ANOVA reports with N=10 (top) and N=30 (bottom).

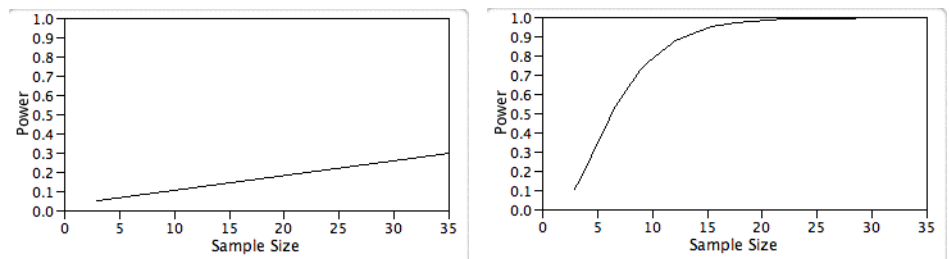


Figure 10: Power plots with higher (left) and lower (right) standard deviation.

ANOM in JMP

Karen Copeland, Boulder Statistics

The Analysis of Means (ANOM) is a graphical multiple comparison procedure for comparing a group of means, rates, or proportions to the overall mean rate or proportion. The goal is to identify any group that appears different from the overall mean. The ANOM is specifically designed to compare a group of means to the overall mean. This differentiates it from ANOVA (which is more general) and the Tukey-Kramer procedure (which is specific to pairwise differences between means). One key benefit to using the ANOM over the ANOVA is the graphical component to the ANOM procedure. The ANOM decision chart, explained below, is an excellent tool for understanding the data analysis.

The ANOM decision chart is similar to a control chart in that it has a center line and decision limits. The group means are plotted against the decision limits; those that fall beyond the limits are statistically different from the overall mean. Laplace first used an ANOM-type procedure in 1827. The graphical form of the procedure and the phrase “analysis of means” were proposed by Ott (1967) and at that point in time approximate critical values were used. Exact critical values have been available since the early 80’s when they were obtained by Peter Nelson (1982). These exact critical values are based on evaluating a negatively equi-correlated k -dimensional singular t -distribution

and, thankfully, are tabulated for the rest of us to use. Tables of critical values can be found in, for example, Nelson (1982), Nelson (1983), Mason *et al.*, (1989), Vardeman (1994), and Nelson, *et al.* (2002).

ANOM assumes that the data are at least approximately normal and that the different treatments all have the same variance. For equal sample sizes, the decision limits are computed using

$$\bar{y} \pm h(\alpha, I, N-I) \sqrt{MSE} \sqrt{\frac{I-1}{N}}$$

where

- \bar{y} = overall mean
- $h(\alpha, I, N-I)$ = tabulated critical value
- I = number of means being compared
- N = total number of data points

$$MSE = \frac{\sum_{i=1}^I s_i^2}{I}$$

For unequal sample sizes, the decision limits (see Nelson (1989) for details) are computed using:

$$\bar{y} \pm m(\alpha, I, N-I) \sqrt{MSE} \sqrt{\frac{N-n_i}{Nn_i}}$$

where

- \bar{y} = overall mean
- $m(\alpha, I, N-I)$ = tabulated critical value
- I = number of means being compared
- N = total number of data points
- n_i = sample size of the i^{th} treatment

$$MSE = \frac{\sum_{i=1}^I (n_i - 1) s_i^2}{N - I}$$

The critical values $m(\alpha, I, N-I)$ are Studentized Maximum Modulus quantiles that are found by numerical integration and, like the h values of the equal sample size case, are tabled in references such as Nelson (1989), Nelson *et al.* (2002), and Nelson *et al.* (2005).

The studentized maximum modulus distribution is the distribution for

$$M(k, \nu) = \max |T_i|$$

where T_i has a univariate t distribution with ν degrees of freedom.

Notice that in the unequal case, the ANOM decision chart has different decision limits for each different treatment sample size.

The ANOM procedure can also be extended to comparing rates and proportions (see Nelson *et al.* (2002) or Nelson *et al.* (2005)).

JMP Scripts for ANOM

ANOM is not a standard procedure in JMP. However, scripts generate ANOM decision charts for comparing means with either equal or unequal sample sizes.

You can download these scripts at <http://www.jmp.com/news/jimpercable>. The script for equal sample sizes is named ANOMB.jsl and the script for unequal sample sizes is named ANOMUB.jsl.

The scripts use look-up files which contain the critical values for two to eight means and significance levels

$\alpha=0.1, 0.05, 0.01, 0.001$. The look-up files can also be downloaded from <http://www.jmp.com/news/jmpcable>. They are Exact Factors for Balanced ANOM.jsl and Exact Factors for ANOMUB.jsl. These files must reside in the same folder as the corresponding scripts, or the statement calling the look-up file in the script needs to include the path to the look-up file.

Balanced Example: Tube Weight Data

A company that produces a medical gel filling product packaged in 4oz. (113.4 g) tubes recently purchased a new filling machine. One regulatory requirement is that they must validate the filling process. That is, they must demonstrate that the machine fills the tubes to the proper amount when using specified machine settings.

As a first step in the validation process, four machine settings were studied to understand their impact on the tube fill weights. A partial view of the data from seven tubes at each of the four settings is given in Figure 11. This data table (fweights.jmp) can be downloaded from <http://www.jmp.com/news/jmpcable>.

	Treatment	Weight
1	1	119.5
2	1	119.9
3	1	120.4
4	1	121.2
5	1	118.7
6	1	119.3
7	1	119.6
8	2	122.1
9	2	123.9
10	2	123.6
11	2	122.4
12	2	122.7
13	3	122.4
14	3	122.1
15	3	122.8
16	3	122.5
17	3	122.3
18	3	122.6
19	4	121.5
20	4	121.8
21	4	121.2
22	4	121.9
23	4	121.6
24	4	121.4

Figure 11: Partial view of the four treatments' data.

In this example, there are four treatments (different machine

settings), and the first question of interest is: Are there any differences in the machine settings with regard to the resulting fill weights?

In this particular example, the data are continuous and likely close enough to being normally distributed. Hence, they can be modeled using a normal distribution. The variances of each treatment are also similar, so we can proceed with the ANOM.

The first step is to run the ANOMB.jsl

script. Figure 12 shows the dialog box that appears when you run the script.

Now simply enter the columns of interest and select the desired level of significance.

The output, shown in Figure 13, includes the decision limits, a chart, a summary of the group means and standard deviations, as well as the ability to change the level of significance.

From this chart, you can conclude (at

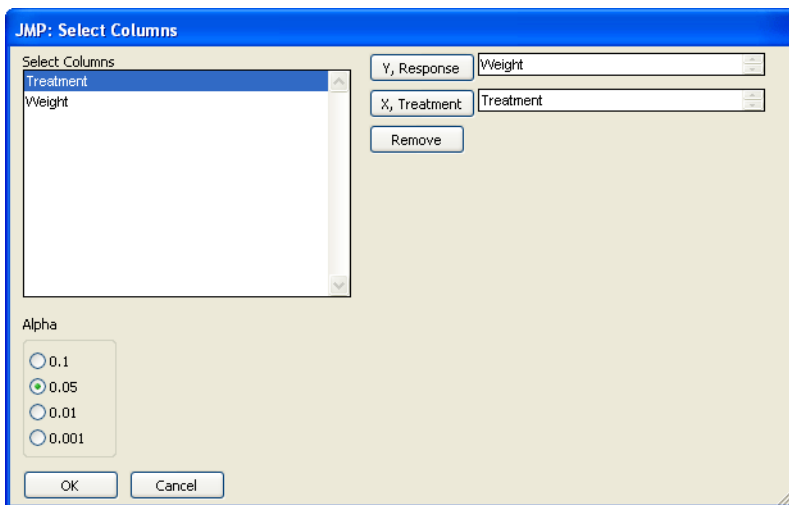


Figure 12: The ANOM script dialog.

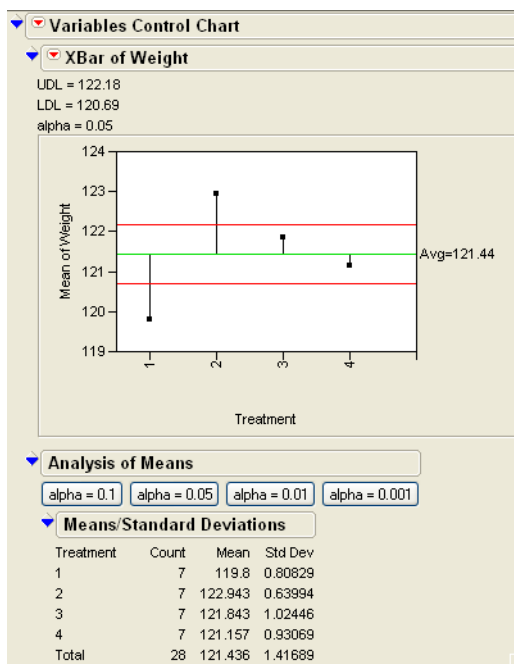


Figure 13: ANOM script output for fweights.jmp.

