04C005I

Project Number: JDW-0301 -41

A Visual Basic Program for Resampling Based Statistical Analysis

An Interactive Qualifying Project Report

Submitted to the Faculty

of the

WORCESTER POLYTECHNIC INSTITUTE

In partial fulfillment of the requirements for the

Degree of Bachelor of Science

By

Joshua Smolic

Minesh G Patel

Huy Dinh Nguyen

Date: Dec 18, 2003

Professor Jayson D. Wilbur, Major Advisor

Abstract

New methods for statistical analysis are usually not adopted by practitioners until the computational components of the method are made accessible via standard software packages. This project involves developing a Visual Basic application to make new statistical methodology accessible to practitioners familiar with Microsoft Excel. The program performs resampling-based statistical analysis and allows for various multiple testing corrections.

Table of Contents

1. Introduction

1.1 Background and Motivation

Statistics is changing. Modern computers and software make it possible to look at data graphically and numerically in ways that were previously computationally infeasible. They enable more realistic, accurate, and informative analyses than could be done before with previous means.

The bootstrap, permutation tests, and other resampling methods are part of this revolution. Resampling methods allow us to quantify uncertainty by calculating standard errors and confidence intervals and performing significance tests. They require fewer assumptions than traditional methods and generally give more accurate answers. Moreover, resampling lets us tackle new inference settings easily (Moore, 2003).

Modern computers are able to do massive amounts of calculations. Resampling tests are not widely available though. These tests are rather new and the programs that do have them tend to be very complicated and expensive. This project proposes a Visual Basic program that works to make an easily used interface for scientists to use that don't have a background in statistics and computer science. For this reason it runs in Microsoft Excel and minimal amounts of user interaction are needed for the tests.

1.2 Literature review

There are several commercial software products are available today to perform statistical analysis of data. Most do not allow for resampling-based inference. However, two commercial programs that provide this feature are described below.

1.2.1 StatXact

StatXact (Cytel Software Corporation, Cambridge, MA) is a software package for exact nonparametric statistical inference on categorical or continuous data. This software provides lot of flexibility in terms of being compatible with different data formats. StatXact contains many procedures covering the following types of statistical tests: one-sample, two independent samples, two dependent samples, more than two independent or dependent samples, stratified samples, measures of agreement and measures of association. The procedures include exact permutation test, asymptotic, Monte Carlo, Bootstrap and many others. Sometimes StatXact will not be able to obtain an exact p-value due to insufficient computer resources and dataset being too large for the exact procedures. The permutation test is exact and able to preserve the Type-I error rate while the asymptotic test is not and bootstrap is ultra-conservative. More information about StatXact can be found at the Cytel Software website: http://www.cytel.com .

1.2.2 InStat

Instant Biostatistics (GraphPad Software, San Diego, CA) is easy to use software that performs statistical analysis on a small amount of data. InStat is not designed to manage large database with many variables. InStat will perform many statistical tests including, the unpaired t test, nonparametric Mann-Whitney test, paired t test, Wilcoxon test, Friedman test, one sample t test, F test, chi-square test, and Fisher exact test. For each variable, InStat computes its best-fit computation with standard error, 95% confidence interval and p-value testing whether the variable contributes significantly to the model. Standard error of the mean and differences between standard error of the means can also

be calculated in this program. For more information about InStat can be found at the GraphPad Software website: http://www.graphpad.com .

1.3 Hypothesis Testing

1.3.1 Introduction

Hypothesis testing is a method of statistical inference in making a decision about a scientific hypothesis, on the basis of observed data. A statistical hypothesis testing problem consists of five main components:

Scientific Hypothesis theorized outcome of the experiment or study.

Statistical Model a model used to describe the observed data. It depends on how the data are obtained. The model most often used for comparing population means is

$$X_{ij} = \mu_i + \varepsilon_{ij} \qquad j = 1,\ldots, n_i;\ i= 1,\ldots, k$$

X_{ij} is the j^{th} observation from the i^{th} population

μ_i is the mean of the i^{th} population

ε_{ij} is the random error associated with the j^{th} observation from the i^{th} population.

Most often it is assumed that ε_{ij} are independent $N(0, \sigma^2)$ random variables.

Statistical Hypothesis is a statement about one or more parameters in the model we're considering. There are typically two hypotheses: a null hypothesis (H_o) and an alternative hypothesis (H_a). The null hypothesis represents a theory that has been put forward, either because it is believed to be true or because it is to be used as a basis for argument, but has not been proved. The alternative hypothesis is a statement of what a statistical hypothesis test is set up to establish. The null hypothesis contradicts the

scientific hypothesis and the alternative hypothesis corresponds to the scientific hypothesis of interest.

Test Statistic a quantity calculated from the sample of data. Its value is used to decide whether or not the null hypothesis should be rejected in the chosen hypothesis test.

Sampling distribution the frequency distribution of a statistic obtained from an extremely large number of random samples drawn from a specified population. For example, the distribution of the test statistic under assumption H_0 is true.

P-value the probability of getting a value of the test statistic as extreme as or more extreme than that observed by chance alone, if the null hypothesis H_0 is true. Typically, the null hypothesis is rejected when the p-value is small (e.g., $p<0.05$).

1.3.2 Statistical Errors

When performing a hypothesis testing it is possible to draw wrong conclusion. Type I and Type II errors are two errors that can be made in hypothesis testing. A Type I error occurs when the null hypothesis is falsely rejected. A Type II error occurs when the null hypothesis is not rejected even though it is false. The probability of making a Type I error is represented by the Greek letter alpha (α) and the probability of making a Type II error is denoted by the Greek letter beta (ß). The table below illustrates this.

		Truth	
		H_0 True	H_0 False
Decision	Reject H_0	Type I error	Correct
	Don't Reject H_0	Correct	Type II error

1.3.3 Multiple Testing

When testing multiple hypotheses one may observe several small p-values, but when performing multiple tests, the probability of making at least one Type I error would be large. A way to solve this problem is by using multiple testing corrections. Suppose there are m null hypotheses $H_{o1},...,H_{om}$ that are to be tested simultaneously against the corresponding alternatives $H_{a1}, \ldots, H_{am}$.

Two error rates associated with Type I errors in multiple testing are the false discovery rate (FDR) and the familywise error rate (FWER). The false discovery rate is the expected ratio of the number of erroneously rejected null hypotheses to the total number of rejected null hypotheses. The familywise error rate is the probability of falsely rejecting at least one true null hypothesis. FWER control reduces the probability of any false positive occurring in an entire study.

Bonferroni correction is the simplest multiple testing correction that controls the FWER. It is very effective when small numbers of tests are made. If there are m hypotheses needing to be tested instead of a single hypothesis, the alpha level is divided by m. For example, suppose we were looking at the effect of different drugs on blood pressure. Instead of testing each drug at the 0.05 alpha level, we would test at alpha= 0.05/10= 0.005 level. This would ensure that the overall chance of making a Type I error is still less than 0.05. Bonferroni correction can also be applied by adjusting the p-value. A Bonferroni adjusted p-value is the number of hypotheses being tested multiply the normal

p-value. If the adjusted p-value ended up greater than 1.0, it would be rounded down to 1.0.

The false discovery rate is defined in the following way. Let R denote the number of null hypotheses that are rejected and V denote the number of null hypotheses that are falsely rejected in a multiple testing procedure. Define Q = V/R if R > 0, and Q = 0 if R = 0; Q is the proportion of Type I errors among the rejected null hypotheses. Then, E(Q) is the FDR of that procedure.

$$FDR = E(Q) = \begin{cases} V/R & \text{if } R > 0 \\ 0 & \text{if } R = 0 \end{cases} \qquad (1)$$

To control the false discovery rate (FDR) the following procedure is employed. The p-values are sorted from smallest to largest: $p_1 < p_2 < \ldots < p_m$ and the hypotheses corresponding to p-values less than or equal to p_h are rejected where

$$h = \left\{ \max 1 \le i \le m : p_i \le \alpha \frac{i}{m} \right\} \qquad (2)$$

2. Methodology

2.1 Introduction

The resampling based statistical methods are used to make statistical inference about observed data without making explicit distributional assumptions. The exact permutation test, the randomization and the bootstrap test are resampling methods to work to find the

same conclusion going about it different ways. While the exact permutation test only considers the set of permutations of the original data, the bootstrap considers all samples from the observed data with replacement. The randomization test is an approximation of the exact permutation test obtained by taking a random sample of all possible permutations.

2.1.1 Test statistic

The test statistic for testing the null hypothesis (H_0) of equal population means is called the F statistic and is computed as the ratio of MSM to MSE: F = MSM/MSE. When the null hypothesis is true and the data follow a normal distribution, the sampling distribution for the test statistic F is an F distribution with first parameter equal to the degrees of freedom associated with SSM, and second parameter equal to the degrees of freedom associated with SSE. In other words, we can express the F ratio as

$$F = [SSM/(k-1)]/[SSE/(n-k)] \qquad (3).$$

Let's step back one-step and explain what these terms and variables mean. SSM is sum of squares model. This is the between-groups mean variation. The degree of freedom for SSE as mention above is $(k - 1)$ and this k is the total number of groups. MSM is the mean square model. SSE is the sum of squares error is the sum of the squares of the residuals. This is the within-group variation. Its degree of freedom is $(n - k)$ where n is the total number of observations and k, again, is the total number of groups. MSE is the mean square error.

The formula in (3) could be shortened and written as

$$F = \frac{\sum_{i=1}^{k} n_i \left(\overline{X}_{i.} - \overline{X}_{..}\right)^2 \Big/ (k-1)}{\sum_{i=1}^{k} \sum_{j=1}^{n_i} \left(\overline{X}_{ij} - \overline{X}_{i.}\right)^2 \Big/ (n-k)} \qquad (4)$$

For this equation, K is the number of groups, n_i is the number of elements in each group, $\overline{X}_{i.}$ is the mean of each group, $\overline{X}_{..}$ is the total mean of all of the observations, $\overline{X}_{ij}$ is each element.

To reduce the number of calculations when calculating the F statistic, one could drop the degrees of freedom part of the numerator and denominator of formula (4). This result could also correct.

Compute original F statistic by

k = number of groups

n_i = number of samples in each group

$\overline{X}_{ij}$ = j[th] sample from i[th] group

$\overline{X}_{i.}$ = mean of i[th] group

$\overline{X}_{..}$ = overall mean

$$F = \frac{\sum_{i=1}^{k} n_i \left(\overline{X}_{i.} - \overline{X}_{..}\right)^2}{\sum_{i=1}^{k} \sum_{j=1}^{n_i} \left(\overline{X}_{ij} - \overline{X}_{i.}\right)^2} \qquad (5)$$

The p-value is a measure of how consistent the test statistic is with the null hypothesis. The p-value is the probability that an observation from the $F_{k-1, n-k}$ distribution exceeds the observed F ratio. A small p-value indicates strong evidence against the null hypothesis and in favor of the alternative hypothesis.

2.2 Exact Permutation Test

2.2.1 What Is an Exact Permutation Test?

An exact permutation test consists of computing a p-value for a given groups of data samples. The p-value of the test is computed by comparing the observed value of the test statistic Ho. Usually under Ho, all permutations are assumed to be equally likely. This distribution consists of all values of the test statistic obtained by suitable permutations of the data values.

A permutation is a reordering of the numbers *1, 2... n*. For example, permutations of the numbers 1 through 6 are shown below.

(1, 2, 3, 4, 5, 6)
(1, 3, 2, 4, 5, 6)
(4, 5, 2, 6, 1, 3)
(3, 2, 1, 6, 4, 5)
.....
.....
etc.....

Note that this includes the standard order in first line. There are n! (n factorial) permutations of n objects. In this case, 6! = 720, so they aren't all written out here.

The term permutation test refers to rearrangements of data. The null hypothesis of this test specifies that the permutations are all equally likely. The exact permutation test used for this project forms all of the permutations, calculating the test statistic for all and considering these values all equally likely. The permutation of the data in each group by moving the data from one group to the other will give the user a test statistic relating the two groups.

Permutation tests were originally proposed by R.A. Fisher (Fisher, 1935) as a first attempt to perform non-parametric statistical tests on observed responses derived from experiments. P.I. Good (1993) describes the five step process used to perform permutation comparison of two samples. These general steps are the same for all permutation tests:

1. Analyze the problem—identify the hypothesis and the alternative(s) of interest.
2. Choose a statistic that will distinguish the hypothesis from the alternative.
3. Compute the test statistic for the original observations.
4. Rearrange (relabel) the observations.
   (a) Compute the test statistic for the new arrangement.
   (b) Compare the new value of the test statistic with the value obtained for the original observations. Repeat steps (a) and (b) until the user is ready to make a decision. For exact test, this will include all possible permutations.
5. Draw a conclusion. Reject the null hypothesis and accept the alternative if the value of the test statistic for the observations as they were labeled originally is an extreme value in the permutation distribution of the statistic.

2.2.2 Advantages of Exact Permutation tests

- Generality:
  - They are appropriate when sampling or experimental units are randomly sampled whether or not experimental units are not selected at random, as long as treatments are randomly assigned to experimental units.
  - They do not depend on the distribution of the data, so they are distribution free in the truest sense.

- It is fairly easy to take into account specifics of the situation of interest and use non-standard test statistics.

- The P-Values are based only on the data given:

  In some tests, the p-values are based on distributional assumptions, such as normality, or approximation, such as the normal approximation to the binomial. With a permutation test, neither of these situations occurs

2.2.3 Disadvantages of Exact Permutation tests

- Applicability

  Permutation tests are not applicable to all problems using any test statistic. As an example, there is no permutation test of the mean of a single distribution using the *t* statistic as the test statistic, since the *t* statistic doesn't change under permutations of the data.

- Ease of Use :

  Despite the fact that the computing power exists to make permutation tests practical, most standard statistical software packages have not yet implemented permutation tests. However, specialized commercial programs are available, and it is just a matter of time until suitable software becomes widely available. For complex models, there are pitfalls in the implementation of permutation tests that require deep thinking about statistical issues.

- Lack of statistical power due to generality.

2.3.4 Algorithm used for exact permutation test

1. Compute F statistics for the observation by equation (5).
2. All numbers from the groups are appended together in order. All permutations of the numbers in order are generated. By selecting the groups in the same order each time, all permutations of the groups are also generated.
3. Count number of permutations whose F-statistic is larger than or equal to the F-statistic is observed data
4. The p-value is the number of permutations larger divided by the total number of possible permutations.

2.2.5 Assumptions

Exact permutation test assumes that under Ho that X_1, X_2 ...X_n are exchangeable. Exchangeable means that the same distributions will result regardless of the ordering of the data.

2.3 Randomization Test

2.3.1 What is Randomization Test?

In practice, even with current fast and powerful computers, exact tests can only be performed when the number of observations (n) is small. For example, if there are 10 observations (n= 10), assuming that there are two groups and there are 5 observations in each group, there are 10! /(5! 5!) = 252 permutations to calculate and if there are 20 observations, there would be 20! /(10! 10!) = 184756 permutations to calculate. As the number of observations increases and the number of permutations increases tremendously. It will take a long time to calculate the F statistics for all these permutations. This is where the randomization test is useful.

A randomization test is a procedure that generates an approximate p-value for a permutation test by computing values of the test statistic from a random sample of all permutations, rather than from all permutations. When the number of observations is so large that exact tests are inappropriate to use, an approximation is used. Generating a subset of all possible permutations because only a sub sample of all possible permutations is calculated does this.

2.3.2 What Are The Advantages and Disadvantages?

The advantages and disadvantages of randomization test are the same as those for the exact permutation test discussed in Section 2.2. Since randomization test is the approximate permutation test, one of the disadvantages is the p-values are approximate. A distinct advantage is that it is still possible to perform a Randomization when the number of observations is large enough to prohibit an Exact test.

2.3.3 Algorithm

There are five steps in calculating the randomization test procedure. These steps are

1. Analyze the problem-identify the null hypothesis and the alternative(s) of interest.
2. Choose a test statistic.
3. Compute a test statistic for the observed or original data.
4. Randomly regenerate the original data and calculated the test statistic. Repeat this step m time where m is the number of permutations (subset of all possible permutations).

5. Calculate the p-value by dividing the number of permutations by the total number of calculated test statistics, done in part 4, that have equal or greater value than the observed test statistic. If the total number of calculated test statistics is 10 and the number of permutations is 10000 then the p-value is 10/10000 or equal to 0.0001. Draw conclusion about the null hypothesis based on this p-value.

2.3.4 Assumptions

When performing a randomization test, we must assume that the random variables that we permute are exchangeable. This basically comes down to the assumption that if the null hypothesis were true, the labels assigning subjects to groups are interchangeable. Even after the data have been collected, the mean of a typical group would have the same expectation after we shuffled subjects among groups. This is why we will create a sampling distribution by taking the data at hand, shuffling them, as reassigning them to groups. When we do this repeatedly, the results we obtain will be the sampling distribution for the test statistic under the null hypothesis

2.4 Bootstrap Test

2.4.1 What is the Bootstrap Test?

The bootstrap was originally proposed in 1979 by Efron (1979). Bootstrap is a general resampling procedure that is computer-based. This statistical method is used to make inferences about a population based on sample statistics.

The bootstrap is empirical approach to understanding the distributional properties of a test statistic, but is also useful as a means of estimating statistics and their standard errors

and hypothesis testing. The observed distribution of sample values is used as an estimate of the underlying probability distribution of the population. Then, the distribution of a statistic for fixed sample sizes is obtained by repeatedly resampling from the distribution n, so that instead of individual partitions of the data having the potential to occur more than once, the individual values themselves may appear repeatedly in a single sample. Under this resampling algorithm the number of possible sample arrangements is much greater compare to the exact and randomization tests. For example with a total sample size $n = 12$, with component samples of size 7 and 5, $12^7 \times 12^5 = 8.9161004 \times 10^{12}$ arrangements are possible. Therefore the test statistic under this algorithm will have a larger standard error since sub-samples of n can deviate lot more.

Consider a set of data drawn from some population. The elements of that population are 121, 118, 110, 34, 22 and 12, with group 1 having values 121,118 and 110 and group 2 with values 34, 22 and 12. Generate a random sample with same number of observations in each group where each value is equally likely to take any value form the whole sample. Then compute the test statistic for that random sample and compare it with the test statistic from the original sample. Perform this for large number of times in order to obtain any desired statistical inferences, such as hypothesis testing and confidence interval estimation. Randomly generated samples using this technique would look like the following for this example:

Bootstrap sample	Treatment	Control	Test statistic
1	22 ,121,121	121,34,110	0.11
2	110,121,110	110,110,12	1320.11
3	34,121,34	110,12,110	205.44

2.4.2 Advantages of Bootstrap?

- Bootstrapping can be used to estimate the sampling distribution of any well-defined statistics of sample data.
- Bootstrapping makes no assumption about the population (no normality and equal variance assumptions, not even the central limit theorem).
- Bootstrapping is especially useful in situations for which no analytic formula for the sampling distribution is available and/or it is intractable (i.e., no look-up table for critical values is possible).
- Another situation in which bootstrapping is useful is when required assumptions of a test are clearly violated. For example, the t-test with two independent sample means requires the equal variance assumption). If this assumption is violated and sample sizes are not large, the test cannot be used and no other alternative exists.

2.4.3 Disadvantages of Bootstrap

- Technology

  Though computation is becoming more powerful and less expensive, a computer is needed to do bootstrapping. The technology is within the range of today's computers.

- Coverage

  There is some evidence of a modest under coverage for bootstrap confidence intervals. For example, since 95% bootstrap confidence intervals such as those we present here might contain the true parameter being estimated less than 95% of the time.

- The P-value differs in general from the ideal P-value.

2.4.4 Algorithm used for Bootstrap

1. Compute F statistics for the observed date by equation (5).
2. Generate random number for each element in the set of original observations to get a resample. If there is a seed available then use that as a seed to generate random numbers otherwise use system clock as seed.
3. Calculate the F-statistic for the resampled data
4. Count number of permutations whose F statistic is larger than the observed F statistic.
5. Repeat steps 2 and 3 for the number of permutations specified.
6. The p-value is the number of permutations larger divided by the total number of possible permutations.

2.4.5 Assumptions

Bootstrap assumes that under H_o, X_{ij} is equally likely to take on any value from the sample population.

3. Procedure

Once the Add-in is installed (See appendix A1 for installation instructions.), to run the Add-in, the user must select *Permutation Addin* from the *Tools* menu. This will bring up a window that will allow the user to select several features. The particular analysis is easily selected by radio buttons. Arrow buttons are used to determine the number of groups to be tested. Reference editor fields are shown corresponding to the number groups. These are used the same way any other reference used in Excel. The stipulation applied to this input is that each row selected is separate variable and that every group

was the same number of variables. If either the randomization or bootstrap test was selected, the number of permutations to calculate and the random seed are configurable. The random seed will either take the seed from the system time or a user inputted number. A button labeled *Calculate total possible* will tell the user how many permutations are possible for the users selected groups.

Error control is also available by selecting either *None*, *Bonferonni*, or *B&H FDR*. The alpha is configurable by the corresponding input box. One additional option is available to allow the user to direct the results to a worksheet. By default, the output will be displayed in a window. Once all of the options have been set, the only thing left to do is press the button *Permute*. This will run the corresponding test on the data and output it in the selected way.

This program also adds the feature of multiple testing. When a group is selected, each row will correspond to a separate sample of that group. Each sample is considered to be a separate variable that is to be tested. By selecting multiple variables from the same sample groups, calculations can be combined. Once a permutation order is calculated, the F-statistic is calculated for all tests from each group. This drastically reduces the number of calculations needed.

4. Results

4.1 Problems

Some of the complications encountered in this project arose from problems inherent in Visual Basic macros. The functions use variables that are all very exact, but something

causes there to be an extreme loss of accuracy. Only 15 decimal places of accuracy are used in the calculations because of one of functions used in the program. Which function was causing the problem was never determined. Another problem resulted from the Msgbox function call (See Appendix A2.). This resulted in a limit being put on the output that can be printed to a message box. Only up to 16 variables can have results outputted to a window.

The exact permutation test has to calculate every possible permutation, so very quickly it become computationally impossible. The timed results of this test are available in *Chart Exact* of Appendix A3. Many of the results are not available because they would take too long to calculate. The randomization times are in *Chart Randomization* of Appendix A3 and the bootstrap times are in *Chart Bootstrap* of Appendix A3.

4.2 Analysis of results

For the exact permutation test, the time per permutation is actually a polynomial equation: Y = 1E-10 * x2 + (0.0001 + 0.0001*z) * x with x = number of permutations, y = number of seconds and z = number of variables. This is a very quickly growing time due to the number of permutations also growing exponentially. Even three groups with five elements will take over 3 minutes on a Pentium 3 500MHz. This just reiterates that the exact test will not work for large samples. The Add-in will also estimate computation time for the randomization and bootstrap tests. However, the estimates can be highly inaccurate.

5. Conclusions and Recommendations

This project resulted in an easy to use add-in that runs in Microsoft Excel. While this is easy to use, it is not as fast as this could be and is sensitive to rounding errors. This program will work with sufficient accuracy for most jobs. However due to the inefficiencies of a Visual Basic, some computations are time consuming.

There are many ways in which this program could be improved including: Much of this program could be cleaned up to run faster. This program could also be sped up by changing programming languages. Many of these calculations can be done as vectors opposed to arrays. Other such optimizations are built in to other programming languages.

However this program does meet the requirements of the project description. It is easy to use and will work with very little knowledge of statistics. All of the code is available and would be easily adaptable by anyone who wanted to customize it.

Bibliography

Chernick , Michael R. (1999) *Bootstrap Methods, A practioner's Guide,* John Wiley & Sons, Inc.

Efron, Bradley. *An introduction to the bootstrap*. New York: Chapman & Hall, 1993

Efron, B. *Bootstrap methods: another look at the jackknife.* Annals of Statistics, 1979

Easton, Valerie J. and John H. McColl Statistics Glossary (2001) http://www.cas.lancs.ac.uk/glossary_v1.1/main.html

Fisher, R.A. *The Design of Experiments,* Edinburgh: Oliver and Boyd, 1935

Good, Phillip (1994) *Permutation Tests, A Practical Guide to Resampling Methods for Testing Hypothesis,* Springer-Verlag.

Heydebreck, Anja von. Multiple testing with gene expression array data (2002) http://www.dkfz-heidelberg.de/biostatistics/training/day1_multiple_testing.pdf

Lane, David M. HyperStat Online Textbook (2003) http://davidmlane.com/hyperstat/index.html

Moore, David S. Practice for business statistics (2003) http://bcs.whfreeman.com/pbs/cat_160/PBS18.pdf

Petruccelli, Nandram, and Chen (1999) *Applied Statistics for Engineers and Scientists*, Prentice Hall.

Wilbur, Jayson D., Permutation-based analysis of factorial experiments (2003) Lecture Notes, WPI

Appendices

A1: Readme.doc

Readme for Permutation Add-in
Version 1.0
Date: Dec 17, 2003

Minimum System Requirements:

System must have installed:
Microsoft Office Excel XP
.NET Framework 1.1
System recommendations:
Windows 2000 or XP
Pentium III 500 MHz or better
512 Mb or RAM or better

Installation:

Included with the program is a file called test.xla. This is the permutation add-in for Excel.

Temporary:
To temporarily load this add-in, all that is required is to open the test.xla file. It will automatically load in Microsoft Excel.

Permanent:

1. Move the file test.xla to somewhere that it won't be deleted or renamed. (ex. C:/program files/Microsoft office/).
2. Open Excel
3. Click the *Tools* menu
    a. Select the menu item *Add-ins*..
4. Press the *Browse* button
5. Navigate the browse window to where you moved the test.xla file
    a. Select the test.xla file
    b. Press the *OK* button
6. If the *Test Addin* option is not selected, do so now by clicking the box just to the left of the name
7. Press the *OK* button

Running the Addin:

Once Excel is running, to run the Add-in, the user must select "permutation addin" from the *Tools* menu. This will bring up a window that will allow the user to select several features. The particular analysis is easily selected by radio buttons. Arrow buttons are used to determine the number of groups to be tested. Reference editor fields are shown corresponding to the number groups. These are used the same way any other reference used in Excel. The stipulation applied to this input is that each row selected is separate variable and that every group was the same number of variables. If either the randomization or bootstrap test was selected, the number of permutations to calculate and the random seed are configurable. The random seed will either take the seed from the system time or a user inputted number. A button labeled *Calculate total possible* will tell the user how many permutations are possible for the users selected groups.

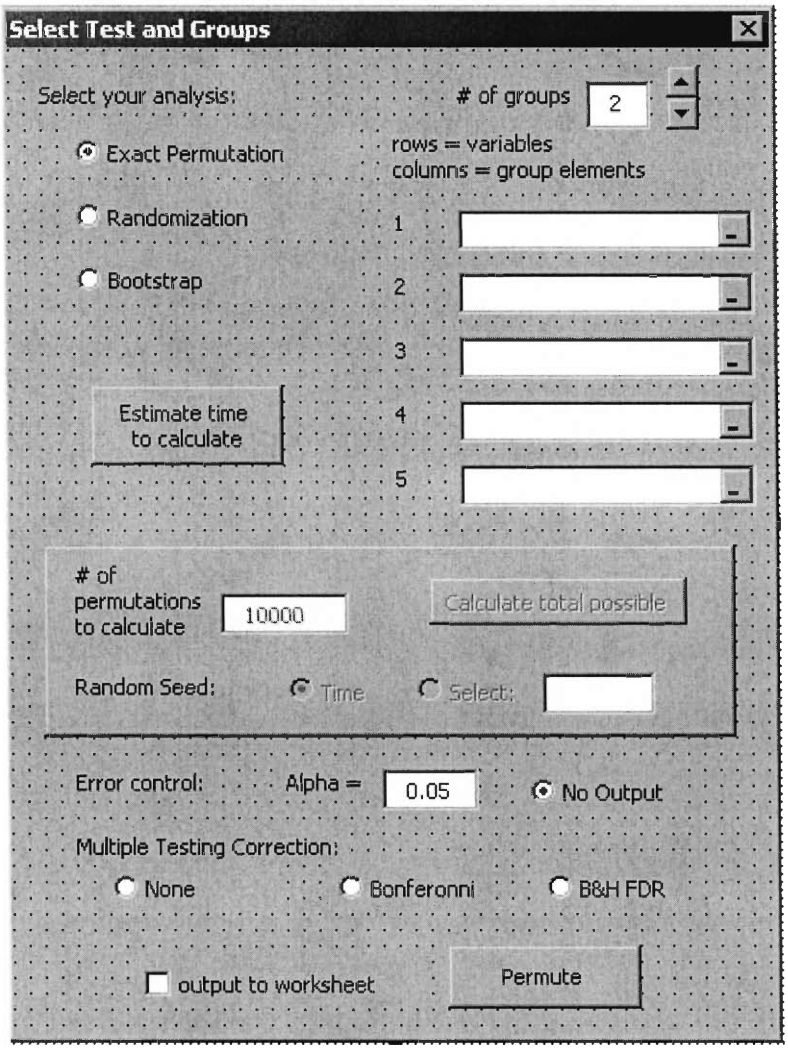
Error control is also available by selecting either *None*, *Bonferonni*, or *B&H FDR*. The alpha is configurable by the corresponding input box. One additional option is available to allow the user to direct the results to a worksheet. By default, the output will be displayed in a window. Once all of the options have been set, the only thing left to do is press the button *Permute*. This will run the corresponding test on the data and output it in the selected way.

This program also adds the feature of multiple testing. When a group is selected, each row will correspond to a separate sample of that group. Each sample is considered to be a separate variable that is to be tested. By selecting multiple variables from the same sample groups, calculations can be combined. Once a permutation order is calculated, the

F statistic is calculated for all tests from each group. This drastically reduces the number of calculations needed.

A2: Source_code.doc

Main select form (group_select):



Code from main select form (group_select):

```
Private col(0, 0) As Double
Private groups() As Integer
Private boxes(5) As Object
Private test As Integer
```

```
Private Sub CommandButton1_Click()
Dim col() As Double
Dim length, offset() As Integer
Dim num_var As Integer
Dim eachGroup() As Double

num_var = 0
length = 0
If (boxes(1).Value <> "") Then
    num_var = Range(boxes(1).Value).Rows.Count
    length = length + Range(boxes(1).Value).Columns.Count
    Dim inc, temp As Integer
    inc = 1
    ReDim eachGroup(1 To inc)
    eachGroup(inc) = length
    inc = inc + 1
    For a = 2 To TextBox1.Value
        ReDim Preserve eachGroup(1 To inc)
        If (boxes(a).Value = "") Then
            MsgBox ("error: blank input")
            Exit Sub
        End If
        If (Range(boxes(a).Value).Rows.Count <> num_var) Then
            MsgBox ("error: #variables doesn't match")
            Exit Sub
        End If

        temp = Range(boxes(a).Value).Columns.Count
        length = length + temp
        eachGroup(inc) = temp
        inc = inc + 1
    Next a
Else
    MsgBox ("error: blank input")
    Exit Sub
End If
ReDim col(1 To num_var, 1 To length)
ReDim groups(1 To TextBox1.Value)
ReDim offset(1 To num_var)
For a = 1 To num_var
    offset(a) = 1
Next a
For a = 1 To TextBox1.Value
    Set r = Range(boxes(a).Value)
    groups(a) = r.Columns.Count
    For b = 1 To r.Rows.Count
        For c = 1 To r.Columns.Count
            If IsNumeric(r.Cells(b, c)) Then
                col(b, offset(b)) = r.Cells(b, c)
                offset(b) = offset(b) + 1
            Else
                MsgBox ("error: a value is not a number")
                Exit Sub
            End If
        Next c
```

```
    Next b
Next a
If (OptionButton4.Value = False) Then
    If (TextBox3.Value < 0 Or TextBox3.Value > 1) Then
        Exit Sub
    End If
End If
Dim seed As Variant
If Me.OptionButton9.Value = True Then
    If Me.TextBox4.Value = "" Then
    seed = Empty
    Else
        seed = Me.TextBox4.Value
    End If
Else
    seed = Empty
End If

If (OptionButton1.Value) Then
    Call perm_a_groups2(col, groups)
Else
    If (OptionButton2.Value) Then
'        MsgBox ("randomization test")
        If IsNumeric(group_select.TextBox2.Value) = False Or _
                group_select.TextBox1.Value = "" Then
            MsgBox ("error: invalid number of permutations")
            Exit Sub
        End If

        Dim col0(), groups0(), vars0() As Double
        Dim counter, incr, numCol As Integer

        numCol = UBound(groups) * UBound(col, 1)

        ReDim col0(1 To numCol)
        ReDim vars0(1 To (UBound(col, 2) * UBound(col, 1)))

        Dim i, j As Integer
        Static k As Integer
        Dim numVars As Integer



        numVars = UBound(col, 1)
        'store numbers in vars0()
        k = 1
        For i = 1 To UBound(col, 1)
            For j = 1 To UBound(col, 2)
                vars0(k) = col(i, j)
                k = k + 1
            Next j
        Next i

        j = 1
        'store number in col0
        For i = 1 To numCol
            col0(i) = eachGroup(j)
```

```
            j = j + 1
            If j > UBound(eachGroup) Then
                j = j Mod UBound(eachGroup)
            End If
        Next i


        Dim pv() As Variant
        ReDim pv(1 To UBound(col, 1))
        If seed = Empty Then
            Call Module3.multiVariables(vars0, col0, numVars, pv,
Me.TextBox2.Value)
        Else
            Call Module3.multiVariables(vars0, col0, numVars, _
                  pv, Me.TextBox2.Value, seed)
        End If

        Else
            If (OptionButton3.Value) Then
                'MsgBox ("bootstrap test")
                If IsNumeric(group_select.TextBox2.Value) = False Or _
                        group_select.TextBox2.Value = "" Then
                    MsgBox ("error: invalid number of permutations")
                    Exit Sub
                End If
                Call Module4.bootstrap(col, groups,
group_select.TextBox2.Value, seed)
            End If
    End If
End If

End Sub

Private Sub CommandButton2_Click()
    Dim d As Double
    Dim top, num_groups As Integer
    d = 1
    num_groups = TextBox1.Value
    For a = 1 To num_groups
        If boxes(a).Value = "" Then
            Exit Sub
        End If
    Next a
        'loop to find true combination
    For i = 1 To num_groups - 1
        top = 0
        For j = i To num_groups
            top = top + Range(boxes(j).Value).Columns.Count
        Next j
        d = d * Module2.combination(top,
Excel.Range(boxes(i).Value).Columns.Count)
    Next i
    MsgBox ("total number of possible permutations = " & d)

End Sub

Private Sub OptionButton1_Click()
```

```
    Me.TextBox2.Enabled = False
    Me.TextBox4.Enabled = False
    Me.OptionButton9.Enabled = False
    Me.OptionButton8.Enabled = False
    Me.CommandButton2.Enabled = False
End Sub

Private Sub OptionButton2_Click()
    Me.TextBox2.Enabled = True
    Me.TextBox1.Enabled = True
    Me.OptionButton9.Enabled = True
    Me.OptionButton8.Enabled = True
    Me.CommandButton2.Enabled = True
End Sub

Private Sub OptionButton3_Click()
    Me.TextBox2.Enabled = True
    Me.TextBox1.Enabled = True
    Me.OptionButton9.Enabled = True
    Me.OptionButton8.Enabled = True
    Me.CommandButton2.Enabled = True
End Sub

Private Sub OptionButton8_Click()
    Me.TextBox4.Enabled = False
End Sub

Private Sub OptionButton9_click()
    Me.TextBox4.Enabled = True
End Sub

Private Sub SpinButton1_Change()
    Me.TextBox1.Value = SpinButton1.Value
    For a = 2 To TextBox1.Value
       boxes(a).Visible = True
    Next a
    For a = TextBox1.Value + 1 To 5
       boxes(a).Visible = False
    Next a
End Sub

Private Sub UserForm_activate()
    Set boxes(1) = Me.RefEdit1
    Set boxes(2) = Me.RefEdit2
    Set boxes(3) = Me.RefEdit3
    Set boxes(4) = Me.RefEdit4
    Set boxes(5) = Me.RefEdit5
    test = 0
End Sub

Private Sub CommandButton3_Click()
Dim d As Double
        Dim top, num_groups, counter As Integer
        d = 1
        counter = 0
        num_groups = TextBox1.Value
        For a = 1 To num_groups
```

```
            If boxes(a).Value = "" Then
                Exit Sub
            End If
        Next a
            'loop to find true combination
        For i = 1 To num_groups - 1
            top = 0
            For j = i To num_groups
                top = top + Range(boxes(j).Value).Columns.Count
            Next j
            d = d * Module2.combination(top,
Excel.Range(boxes(i).Value).Columns.Count)
            counter = counter + Range(boxes(i).Value).Columns.Count
        Next i
        counter = counter +
Range(boxes(num_groups).Value).Columns.Count
    If (Me.OptionButton1.Value = True) Then
        MsgBox ("seconds: " & CStr((0.000000000103156 * (d) * (d)) + _
            (0.0001159 * d * Range(boxes(1).Value).Rows.Count) +
(0.0005167 * d)))
    End If
    If (Me.OptionButton2.Value = True) Then
        MsgBox ("seconds: " & CStr((0.05965 * counter) + _
            (0.2976 * Range(boxes(1).Value).Rows.Count) + (0.00004905 *
(Me.TextBox2.Value))) ^ 2)
    End If
    If (Me.OptionButton3.Value = True) Then
        MsgBox ("seconds: " & CStr((0.02671 * counter) + _
            (0.24874 * Range(boxes(1).Value).Rows.Count) + (0.00002153
* (Me.TextBox2.Value))) ^ 2)
    End If
End Sub
```

Module 1 code:

```
Sub Start_Perm()
    'show input form
    'UserForm1.Show
    group_select.Show
End Sub

Sub Start_perm2()
    group_select2.Show
End Sub

Private Function Max(ByRef a, ByRef b)
    If a > b Then
        Max = a
    Else
        Max = b
    End If
End Function

Public Sub screen_output(ByRef pvalue, ByRef start_time, ByRef
end_time)
    Dim sorted() As Double
```

```
    Dim sorted_temp(1 To 2) As Double
    ReDim sorted(1 To 2, 1 To UBound(pvalue))

    'output to screen in msgbox or worksheet
    If group_select.out_worksheet_checkbox.Value = False Then
        Dim out As String
        If (group_select.OptionButton4.Value) Then
            For a = 1 To UBound(pvalue)
                out = out + CStr("Variable " & a & " P-value = " &
pvalue(a) & Chr(10))
            Next a
        End If
        If (group_select.OptionButton5.Value) Then
            For a = 1 To UBound(pvalue)
                out = out + CStr("Variable " & a & " P-value = " &
pvalue(a))
                If (pvalue(a) <= CDbl(group_select.TextBox3.Value))
Then
                    out = out + CStr(" Significant: yes" & Chr(10))
                Else
                    out = out + CStr(" Significant: no" & Chr(10))
                End If
            Next a
        End If
        If (group_select.OptionButton6.Value) Then
            For a = 1 To UBound(pvalue)
                out = out + CStr("Variable " & a & " P-value = " &
pvalue(a))
                If (pvalue(a) <= (CDbl(group_select.TextBox3.Value) /
UBound(pvalue))) Then
                    out = out + CStr(" Significant: yes" & Chr(10))
                Else
                    out = out + CStr(" Significant: no" & Chr(10))
                End If
            Next a
        End If
        If (group_select.OptionButton7.Value) Then
            For a = 1 To UBound(sorted, 2)
                sorted(1, a) = pvalue(a)
                sorted(2, a) = a
            Next a
            top_index = UBound(sorted, 2) - 1
            Do While top_index >= 1
                For a = 1 To top_index
                    If sorted(1, a) > sorted(1, a + 1) Then
                        sorted_temp(1) = sorted(1, a)
                        sorted_temp(2) = sorted(2, a)
                        sorted(1, a) = sorted(1, a + 1)
                        sorted(2, a) = sorted(2, a + 1)
                        sorted(1, a + 1) = sorted_temp(1)
                        sorted(2, a + 1) = sorted_temp(2)
                    End If
                Next a
                top_index = top_index - 1
            Loop
            rbh = 0
            For a = 1 To UBound(sorted, 2)
```

```
                If (sorted(1, a) <= ((group_select.TextBox3.Value) * _
                        (a / UBound(sorted, 2)))) Then
                    rbh = Max(rbh, sorted(1, a))
                End If
            Next a
            For a = 1 To UBound(pvalue)
                out = out + CStr("Variable " & a & " P-value = " &
pvalue(a))
                If (pvalue(a) <= rbh) Then
                    out = out + CStr(" Significant: yes" & Chr(10))
                Else
                    out = out + CStr(" Significant: no" & Chr(10))
                End If
            Next a
        End If
        If (Len(out) > 1024) Then
        out = CStr("Sorry, there is too much output to fit in a
window." & Chr(10))
        End If
        out = out + CStr("seconds to calculate: " & (end_time -
start_time))
        Call MsgBox(out, vbInformation, "Results")
    Else
        Dim newsheet As Worksheet
        Set newsheet = Application.Worksheets.Add
        newsheet.Range("a1").Value = "Variable"
        newsheet.Range("b1").Value = "P-Value"
        For a = 1 To UBound(pvalue)
            newsheet.Range(CStr("a" & a + 1)).Value = CStr(a)
            newsheet.Range(CStr("b" & a + 1)).NumberFormat =
"########0.000000"
            newsheet.Range(CStr("b" & a + 1)).Value = CStr(pvalue(a))
        Next a
        If (group_select.OptionButton5.Value) Then
            newsheet.Range("c1").Value = "Significant"
            For a = 1 To UBound(pvalue)
                If (pvalue(a) <= CDbl(group_select.TextBox3.Value))
Then
                    newsheet.Range(CStr("c" & a + 1)).Value = "yes"
                Else
                    newsheet.Range(CStr("c" & a + 1)).Value = "no"
                End If
            Next a
        End If
        If (group_select.OptionButton6.Value) Then
            newsheet.Range("c1").Value = "Significant"
            For a = 1 To UBound(pvalue)
                If (pvalue(a) <= (CDbl(group_select.TextBox3.Value) /
UBound(pvalue))) Then
                    newsheet.Range(CStr("c" & a + 1)).Value = "yes"
                Else
                    newsheet.Range(CStr("c" & a + 1)).Value = "no"
                End If
            Next a
        End If
        If (group_select.OptionButton7.Value) Then
            For a = 1 To UBound(sorted, 2)
```

```
                sorted(1, a) = pvalue(a)
                sorted(2, a) = a
            Next a
            top_index = UBound(sorted, 2) - 1
            Do While top_index >= 1
                For a = 1 To top_index
                    If sorted(1, a) > sorted(1, a + 1) Then
                        sorted_temp(1) = sorted(1, a)
                        sorted_temp(2) = sorted(2, a)
                        sorted(1, a) = sorted(1, a + 1)
                        sorted(2, a) = sorted(2, a + 1)
                        sorted(1, a + 1) = sorted_temp(1)
                        sorted(2, a + 1) = sorted_temp(2)
                    End If
                Next a
                top_index = top_index - 1
            Loop
            rbh = 0
            For a = 1 To UBound(sorted, 2)
                If (sorted(1, a) <= CDbl(group_select.TextBox3.Value) *
_
                        (a / UBound(sorted, 2))) Then
                    rbh = Max(rbh, sorted(1, a))
                End If
            Next a
            newsheet.Range("c1").Value = "Significant"
            For a = 1 To UBound(pvalue)
                If (pvalue(a) <= rbh) Then
                    newsheet.Range(CStr("c" & a + 1)).Value = "yes"
                Else
                    newsheet.Range(CStr("c" & a + 1)).Value = "no"
                End If
            Next a
        End If
        newsheet.Range("d1").Value = "Seconds to calculate"
        newsheet.Range("d2").Value = end_time - start_time
    End If

End Sub
```

Module 2 code:

```
Private result() As Variant
Private initial_result() As Variant
Private pvalue() As Variant
Private num_set() As Variant
Private temp1, temp2 As Variant
Private remain_nums_temp() As Integer
Private overall_mean() As Variant
Private group_mean As Variant
Private mean_offset As Integer
Private groups() As Integer

Sub Auto_Open()
Dim a As Boolean
```

```
a = False
For b = 1 To CommandBars("tools").Controls.Count
    If CommandBars("tools").Controls.Item(b).Caption = "Permutation
Addin" Then
        a = True
    End If
Next b
If a Then
Else
    Set newItem =
CommandBars("Tools").Controls.Add(Type:=msoControlButton,
Temporary:=True)
    With newItem
        .BeginGroup = True
        .Caption = "Permutation Addin"
        .FaceId = 0
        .OnAction = "Start_Perm"
    End With
End If
End Sub

Private Function Max(ByRef a, ByRef b)
    If a > b Then
        Max = a
    Else
        Max = b
    End If
End Function

Public Function combination(ByVal top, ByVal bottom) As Variant
'combinations
    Dim a, b As Variant
    a = 1
    b = 1
    For c = top - bottom + 1 To top
        a = a * c
    Next c
    For d = 1 To bottom
        b = b * d
    Next d
    combination = a / b
    'MsgBox (combination)    'anything over 32000 is very long
End Function

Function perm_a_groups2(ByRef col, ByRef pub_groups)     'setup to
permutate
    Dim num_combinations As Variant
    Dim top As Integer
    Dim start_time
    Dim end_time
    Dim top_index As Integer
    Dim rbh As Variant


    start_time = Timer

    groups = pub_groups
```

```
    num_combinations = 1
    'loop to find true combination
    For i = 1 To UBound(groups) - 1
        top = 0
        For j = i To UBound(groups)
            top = top + groups(j)
        Next j
        num_combinations = num_combinations * combination(top,
groups(i))
    Next i
    ReDim result(1 To UBound(col, 1))
    ReDim initial_result(1 To UBound(col, 1))
    Dim listbefore(), curgroup, curln, remainnums() As Integer
    ReDim remain_nums_temp(1 To UBound(col, 2))
    ReDim remainnums(1 To UBound(col, 2))
    ReDim listbefore(UBound(col, 2))
    ReDim num_set(1 To UBound(col, 1), 1 To UBound(col, 2))
    ReDim pvalue(1 To UBound(col, 1))
    ReDim overall_mean(1 To UBound(col, 1))

    'setup num_set(,)
    For i = 1 To UBound(col, 2)
        remainnums(i) = i
        For a = 1 To UBound(col, 1)
            num_set(a, i) = col(a, i)
        Next a
    Next i
    curgroup = 1
    curln = 1
    For a = 1 To UBound(pvalue)
        pvalue(a) = 0
    Next a

    'find overall mean
    For a = 1 To UBound(pvalue)
        overall_mean(a) = 0
        For b = 1 To UBound(num_set, 2)
            overall_mean(a) = overall_mean(a) + (num_set(a, b))
        Next b
        overall_mean(a) = overall_mean(a) / UBound(num_set, 2)
    Next a

    'find initial result
    For a = 1 To UBound(listbefore)
        listbefore(a) = a
    Next a
    Dim temp1, temp2, group_mean As Variant
    For a = 1 To UBound(num_set, 1)
        temp1 = 0
        temp2 = 0
        mean_offset = 0
        group_mean = 0
        For b = 1 To UBound(groups)
            'set group mean
            group_mean = 0
            For c = 1 To groups(b)
```

```
                group_mean = group_mean + (num_set(a, listbefore(c +
mean_offset)))
            Next c
            group_mean = group_mean / (groups(b))

            For c = 1 To groups(b)
                'set bottom fraction
                temp2 = temp2 + ((num_set(a, listbefore(c +
mean_offset))) - group_mean)_
                    * ((num_set(a, listbefore(c + mean_offset))) -
group_mean)
            Next c

            'add top of fraction
            temp1 = temp1 + (groups(b) * (group_mean - overall_mean(a))
* _
                (group_mean - overall_mean(a)))

            mean_offset = mean_offset + (groups(b))
        Next b
        If (temp2 = 0) Then
            initial_result(a) = 0
        Else
            initial_result(a) = temp1 / temp2
        End If
    Next a


    'permutate
    Call permgroups2(listbefore, curgroup, curln, remainnums, _
        UBound(remainnums), 0)
    For a = 1 To UBound(pvalue)
        pvalue(a) = pvalue(a) / num_combinations
    Next a

    end_time = Timer

    'output to screen in msgbox or worksheet
    Call Module1.screen_output(pvalue, start_time, end_time)
    'MsgBox ("Seconds to calculate: " & end_time - start_time)


End Function

Private Sub permgroups2(ByRef listbefore, ByVal curgroup, _
        ByVal curln, ByVal remainnums, _
        ByVal remainnums_length, ByVal listbefore_length)
    If curgroup < UBound(groups) Then

        If curln <= groups(curgroup) Then

            For j = 1 To remainnums_length

            If (remainnums(j) > listbefore(listbefore_length)) Or curln
= 1 Then
                offset = 0
                For k = 1 To remainnums_length
```

```
                    If k = j Then
                        offset = 1
                    Else
                        remain_nums_temp(k - offset) = remainnums(k)
                    End If
                Next k

                listbefore(listbefore_length + 1) = remainnums(j)

                Call permgroups2(listbefore, curgroup, curln + 1, _
                    remain_nums_temp, remainnums_length - 1, _
                    listbefore_length + 1)
            End If
            Next j
        Else
            Call permgroups2(listbefore, curgroup + 1, 1, remainnums, _
                remainnums_length, listbefore_length)
        End If


    Else
        Call perm_out2(listbefore, curgroup, curln, remainnums)
    End If
End Sub

Private Sub perm_out2(ByRef listbefore, ByRef curgroup, _
        ByRef curln, ByRef remainnums)
'ReDim Preserve listbefore(UBound(num_set))
    Dim temp1, temp2, group_mean As Variant

    offset = 1
    For j = UBound(num_set, 2) - groups(UBound(groups)) + 1 To
UBound(num_set, 2)
        listbefore(j) = remainnums(offset)
        offset = offset + 1
    Next j

    'find output for result

    For a = 1 To UBound(num_set, 1)
        temp1 = 0
        temp2 = 0
        mean_offset = 0
        group_mean = 0
        For b = 1 To UBound(groups)
            'set group mean
            group_mean = 0
            For c = 1 To groups(b)
                group_mean = group_mean + (num_set(a, listbefore(c +
mean_offset)))
            Next c
            group_mean = group_mean / (groups(b))

            For c = 1 To groups(b)
                'set bottom fraction
                temp2 = temp2 + ((num_set(a, listbefore(c +
mean_offset))) - group_mean)_
```

```
                    * ((num_set(a, listbefore(c + mean_offset))) -
group_mean)
            Next c

            'add top of fraction
            temp1 = temp1 + (groups(b) * (group_mean - overall_mean(a))
* _
                (group_mean - overall_mean(a)))

            mean_offset = mean_offset + (groups(b))
        Next b

        If (temp2 = 0) Then
            If (temp1 = 0) Then
                result(a) = 0
            Else
                result(a) = 1.79769313486231E+308
            End If
        Else
            result(a) = temp1 / temp2
        End If
    Next a
    For a = 1 To UBound(num_set, 1)
        If (result(a) >= initial_result(a) * (1 - 0.000000000000001))
Then                ' add to remove divide errors
        'If (result(a) >= initial_result(a)) Then
            pvalue(a) = pvalue(a) + 1
        End If
    Next a
End Sub
```

Module 3 code:

```
Public Function Rand(Upper As Integer, Lower As Integer) As Integer
    'generate a number from Lower to Upper Bound inclusively
    Rand = Int((Upper - Lower + 1) * Rnd + Lower)
End Function
Public Function RandomNumbers(ByRef arrNums As Variant, ByVal
totalNumber, ByVal HowMany As Integer, Unique As Boolean) As Variant

    'RandomNumbers() function - randomly generate a group of unique
numbers and
    ' store these numbers in arrNums array
    Dim x           As Integer
    Dim n           As Integer
    Dim colNumbers  As New Collection
    Static i
    i = 1

    Do Until i > UBound(arrNums)
     With colNumbers
        'First populate the collection
        For x = 1 To totalNumber
            .Add x
        Next x
```

```
        For x = 1 To HowMany
            n = Rand(0, colNumbers.Count + 1)
            arrNums(i) = colNumbers(n)
            If Unique Then
                colNumbers.Remove n
            End If
            i = i + 1
        Next x
    End With
    'Redim colNumbers As New Collection
    Do Until colNumbers.Count <= 0
        arrNums(i) = colNumbers(1)
        colNumbers.Remove 1
        i = i + 1
    Loop
    Loop
End Function

Public Function SortRandom(ByVal array1 As Variant, numOfRand As
Integer, ByRef new_arr1 As Variant) As Variant
    'call randomNumbers to randomly sort an array of numbers and store
in ranTemp()
    Call RandomNumbers(array1, new_arr1, numOfRand, True)
End Function
 Private Function calculate_F_obs(ByVal array1 As Variant, ByVal array2
As Variant, _
                                    ByVal totalElements As Integer,
ByVal total_mean As Variant, ByRef F_value As Variant) As Variant
    Dim group_sum(), group_mean(), top(), bottom() As Variant
    ReDim group_sum(1 To (UBound(array1) / totalElements) *
UBound(array2))
    ReDim group_mean(1 To (UBound(array1) / totalElements) *
UBound(array2))
    ReDim top(1 To UBound(array1) / totalElements)
    ReDim bottom(1 To UBound(array1) / totalElements)

    Dim i, j, k, temp

    k = 1
    temp = 1
    'finding the group_sum and group_mean
    For i = 1 To UBound(group_sum)
        group_sum(i) = 0
        For j = 1 To array2(temp)
            group_sum(i) = group_sum(i) + array1(k)
            k = k + 1
        Next j
        group_mean(i) = group_sum(i) / array2(temp)
        temp = temp + 1
        If temp > UBound(array2) Then
            temp = 1
        End If
    Next i

    'finding top fraction
    k = 1
    temp = 1
```

```
    For i = 0 To UBound(top)
        For j = temp To UBound(group_mean) / UBound(total_mean) * k
            If j Mod UBound(array2) = 1 Then
                i = i + 1
                top(i) = 0
            End If
            top(i) = top(i) + (group_mean(j) - total_mean(k)) ^ 2
            temp = temp + 1
        Next j
        k = k + 1
        If i < UBound(top) Then
            i = i - 1
        End If
    Next i

    Dim temp2, array2Count
    temp = 1
    array2Count = array2(temp)
    temp2 = 1
    k = 1

    'finding bottom fraction
   For i = 1 To UBound(bottom)
        bottom(i) = 0
        For j = 1 To totalElements
            If j > array2Count Then
                temp2 = temp2 + 1
                temp = temp + 1
                array2Count = array2Count + array2(temp)
            End If
            bottom(i) = bottom(i) + (array1(k) - group_mean(temp2)) ^ 2
            k = k + 1
        Next j
        temp2 = temp2 + 1
        temp = 1
        array2Count = array2(temp)
   Next i

    'now calculate f_stats
    For i = 1 To UBound(F_value)
        If bottom(i) = 0 Then
           bottom(i) = 1
        End If
        F_value(i) = top(i) / bottom(i)
    Next i

End Function
Private Function RandTest(ByVal array1 As Variant, ByVal array2 As
Variant, ByVal numVar As Integer, ByVal numPerm As Variant, _
                          ByVal elementIn1Var As Integer, ByVal
totalSum As Variant, ByVal totalmean As Variant, ByRef p_val As
Variant) As Variant
    Dim counter(), new_arr1(), sort_index(), total_sum() As Variant
    Dim F_orig(), F_new() As Variant
    ReDim total_sum(1 To numVar)
    ReDim F_orig(1 To numVar)
    ReDim new_arr1(1 To UBound(array1) * numPerm)
```

```
    ReDim F_new(1 To numPerm * numVar)
    ReDim counter(1 To numVar)

    Dim i, j, k, st, en, numOfRan
    For i = 1 To UBound(counter)
        counter(i) = 0
    Next i

    Dim newArr1Count

    numOfRan = elementIn1Var - array2(UBound(array2))
    ReDim sort_index(1 To elementIn1Var * numPerm)


    'find f_obs original value
    Call calculate_F_obs(array1, array2, elementIn1Var, totalmean,
F_orig)

    Call RandomNumbers(sort_index, elementIn1Var, numOfRan, True)

   newArr1Count = 0
    'update array1
    k = 1

    For j = 1 To UBound(F_orig)
        For i = 1 To UBound(sort_index)
            new_arr1(k) = array1(sort_index(i) + newArr1Count)
            k = k + 1
        Next i
        newArr1Count = newArr1Count + elementIn1Var
    Next j

    'find new F_obs value
    st = Timer
    Call calculate_F_obs(new_arr1, array2, elementIn1Var, totalmean,
F_new)
    en = Timer
    Value = en - st
    j = 0
    For i = 1 To UBound(F_new)
        If i Mod (UBound(F_new) / numVar) = 1 Then
            j = j + 1
        End If
        If F_new(i) >= F_orig(j) Then
         counter(j) = counter(j) + 1
        End If
    Next i
    'find p_value
    For i = 1 To UBound(p_val)
        p_val(i) = counter(i) / numPerm
    Next i
    'return p_value
    RandTest = p_value
```

```
End Function

 Public Function multiVariables(ByVal array1 As Variant, ByVal array2
As Variant, ByVal numVars As Variant, ByRef p_values As Variant, ByVal
numPerm As Variant, Optional ByVal seed As Variant) As Variant

    Dim k, numOfG, sumG As Integer
    Dim new_arr2(), total_sum(), total_mean() As Variant
    Dim start_time, end_time, total_time As Variant
    Dim total_element, i, j

    'number of Groups
    numOfG = UBound(array2) / numVars

    ReDim new_arr2(1 To numOfG)
    ReDim total_sum(1 To numVars)
    ReDim total_mean(1 To numVars)

    If IsMissing(seed) Then
        Rnd
        Math.Randomize
    Else
       Rnd [-1]
        Math.Randomize [seed]
    End If

    start_time = Timer

    For k = 1 To numOfG
        new_arr2(k) = array2(k)
    Next k

    total_element = 0
    For k = 1 To UBound(new_arr2)
        total_element = total_element + new_arr2(k)
    Next k

    i = 1
    'find the total sum and total mean of each group
    For k = 1 To UBound(array1)
        total_sum(i) = 0
        For j = 1 To total_element
            total_sum(i) = total_sum(i) + array1(k)
            k = k + 1
        Next j
        total_mean(i) = total_sum(i) / total_element
        k = k - 1
        i = i + 1
    Next k
      '  total_time = end_time - start_time

        'call randTest()
       Call RandTest(array1, new_arr2, numVars, numPerm, total_element,
total_sum, total_mean, p_values)
        end_time = Timer
```

```
    Call Module1.screen_output(p_values, start_time, end_time)

End Function
```

Module 4 code:

```
Function bootstrap(ByRef datalength() As Double, ByRef grouplength() As
Integer, ByRef numofperm As Variant, seed As Variant) As Double
Dim rndval As Integer
Dim grouptotal, diff1, squa1, totalmean, ssto, totsum, groupmean,
squa2, diff2, sse As Double
Dim fstat As Double
Dim store() As Double
Dim temp() As Double
ReDim store(1 To UBound(datalength, 1))
Dim pvalue() As Double
ReDim pvalue(UBound(datalength, 1))
Dim start_time
Dim end_time
Dim orignal() As Double
Dim z As Integer
Dim other() As Double

start_time = Timer


If seed <> Empty Then
        Rnd (-1)
        Math.Randomize (seed)
    Else
        Rnd
        Math.Randomize
    End If

orignal() = ankit(datalength, grouplength)

For s = 1 To UBound(pvalue)
pvalue(s) = 0
Next s



For i = 1 To numofperm

    For m = 1 To UBound(datalength, 1)
    z = 1

    For x = 1 To UBound(datalength, 2)
    totsum = datalength(m, x) + totsum
    Next x
    totalmean = totsum / UBound(datalength, 2)
    totsum = 0
```

```
If (m = 1) Then
ReDim temp(UBound(datalength, 2))
For v = 1 To UBound(temp)
rndvalue = Int((UBound(datalength, 2) * Rnd()) + 1)
temp(v) = rndvalue
Next v
End If


For j = 1 To UBound(grouplength)

 ReDim other(grouplength(j))

    For t = 1 To UBound(other)
    other(t) = 0
    Next t


    For k = 1 To grouplength(j)
    other(k) = datalength(m, temp(z))
    grouptotal = other(k) + grouptotal
    z = z + 1
    Next k

    groupmean = grouptotal / grouplength(j)

    For p = 1 To UBound(other)
    diff1 = other(p) - groupmean
    squa1 = diff1 * diff1
    ssto = squa1 + ssto
    Next p

 diff2 = groupmean - totalmean
 squa2 = diff2 * diff2
 sse = squa2 + sse
 grouptotal = 0

 Next j

If (ssto = 0) Then
  If (sse = 0) Then
  fstat = 0
  Else
  fstat = 1.79769313486232E+307
  End If
 Else
     fstat = sse / ssto
End If


store(m) = fstat

 ssto = 0
 sse = 0
 totalmean = 0
```

```
 Next m

For s = 1 To UBound(datalength, 1)
If store(s) > orignal(s) Then
pvalue(s) = pvalue(s) + 1
End If
Next s

Next i


 For g = 1 To UBound(pvalue)
    pvalue(g) = pvalue(g) / numofperm
Next g

end_time = Timer

Call screen_output(pvalue(), start_time, end_time)


End Function



Function ankit(ByRef datalength() As Double, ByRef grouplength() As
Integer) As Double()
Dim z As Integer
z = UBound(datalength)
Dim fstat As Double
Dim grouptotal, diff1, squa1, totalmean, ssto, totsum, groupmean,
squa2, diff2, sse As Double
Dim orignalf() As Double
ReDim orignalf(1 To UBound(datalength, 1))

Dim p As Double
Dim q As Integer
Dim n As Integer
Dim m As Integer
Dim x As Integer
x = 1
q = 0


For y = 1 To UBound(datalength, 1)
For n = 1 To UBound(datalength, 2)
totsum = datalength(y, n) + totsum
Next n
totalmean = totsum / (UBound(datalength, 2))
totsum = 0
For i = 1 To UBound(grouplength)

p = 0

For j = 1 To grouplength(i)
p = datalength(y, x) + p
diff1 = datalength(y, x) - totalmean
```

```
squa1 = diff1 * diff1
ssto = squa1 + ssto
x = x + 1
Next j
groupmean = p / grouplength(i)
x = j
diff2 = groupmean - totalmean
squa2 = diff2 * diff2
sse = squa2 + sse
grouptotal = 0
Next i

If (ssto = 0) Then
     If (sse = 0) Then
     fstat = 0
     Else
        fstat = 1.79769313486232E+307
     End If
    Else
        fstat = sse / ssto
   End If


orignalf(y) = fstat

sse = 0
ssto = 0
totalmean = 0
Next y
ankit = orignalf()
End Function
```

A3: Contents of attached CD:

1) Readme.doc: This contains the instructions on how to install and use the add-in.

2) Source_code.doc: This is all of the source code from the test.xla file. This includes all of the modules, forms, and code in the forms.

3) Stats.xls: This contains the test times recorded using the installed add-in. It shows where the formulas came from to produce the *Estimate time to calculate* button.

4) test.xla: This is the add-in that was produced by this project.