INL/EXT-12-26182

Uncertainty Analysis of RELAP5-3D©

Alexandra E. Gertman
George L. Mesina

July 2012

INL/EXT-12-26182

Uncertainty Analysis of RELAP5-3D©

Alexandra E. Gertman
George L. Mesina

July 2012

Idaho National Laboratory
Idaho Falls, Idaho 83415

http://www.inl.gov

Prepared for the
International RELAP5 User Group
and for the
U.S. Department of Energy
Under DOE Idaho Operations Office
Contract DE-AC07-05ID14517

1-1-2012

Uncertainty Analysis of RELAP5-3D©

Alexandra E. Gertman
Boise State University

George L. Mesina
Idaho National Laboratory

Uncertainty Analysis of RELAP5-3D©

Alexandra E. Gertman[*] Dr. George L. Mesina[†]

July 2012

[*]Mathematics Department, Boise State University

[†]Idaho National Laboratory

Abstract

As world-wide energy consumption continues to increase, so does the demand for the use of alternative energy sources, such as Nuclear Energy. Nuclear Power Plants currently supply over 370 gigawatts of electricity, and more than 60 new nuclear reactors have been commissioned by 15 different countries. The primary concern for Nuclear Power Plant operation and lisencing has been safety. The safety of the operation of Nuclear Power Plants is no simple matter- it involves the training of operators, design of the reactor, as well as equipment and design upgrades throughout the lifetime of the reactor, etc. To safely design, operate, and understand nuclear power plants, industry and government alike have relied upon the use of best-estimate simulation codes, which allow for an accurate model of any given plant to be created with well-defined margins of safety. The most widely used of these best-estimate simulation codes in the Nuclear Power industry is RELAP5-3D. Our project focused on improving the modeling capabilities of RELAP5-3D by developing uncertainty estimates for its calculations. This work involved analyzing high, medium, and low ranked phenomena from an INL PIRT on a small break Loss-Of-Coolant Accident as well as an analysis of a large break Loss-Of-Coolant Accident. Statistical analyses were performed using correlation coefficients. To perform the studies, computer programs were written that modify a template RELAP5-3D input deck to produce one deck for each combination of key input parameters. Python scripting enabled the running of the generated input files with RELAP5-3D on INL's massively parallel cluster system. Data from the studies was collected and analyzed with SAS. Summaries of the results of our studies are presented.

Table of Contents

1 Nomenclature[1]

Accronym	Definition
ADS	Automatic Depressurization System
ALWR	Advanced Light-Water Reactor
AP600	Advanced Passive 600 MWe Reactor
CMT	Core Makeup Tank
ECCS	Emergency Core Coolant System
INL	Idaho National Laboratory
IRWST	In-containment Refueling Water Storage Tank
LBLOCA	Large Break Loss-of-Coolant Accident
LOCA	Loss-of-Coolant Accident
MSLB	Main Steam Line Break
NPP	Nuclear Power Plant
NRC	United States Nuclear Regulatory Commission [1]
PIRT	Phenomena Identification and Ranking Table
PPMCC	Pearson Product Moment Correlation Coefficient
PRHR	Passive Residual Heat Removal
PWR	Pressurized Water Reactor
SBLOCA	Small Break Loss-of-Coolant Accident
SG	Steam Generator
$\hat{\rho}$	The estimated value of PPMCC
$\hat{\mathfrak{r}}_s$	The estimated value of Spearman's rho
$\hat{\tau}$	The estimated value of Kendall's tau

[1] Also referred to as USNRC

2 Motiviation

As scientists, we wish to confidently provide the answer to any question quantitatively. However, every measured value has uncertainty attached to it and thus any answer we provide must include uncertainty. Uncertainty in measurement is not the only uncertainty we must concern ourselves with. For any system we construct, outside factors will play some role within our calculations and findings, and therefore additional uncertainties must be accounted for in any solutions we provide.

In practice it is not reasonable to assume that we could quantify all uncertainties for a particular solution. Even if it were possible to quantify all possible solutions, combining all the uncertainties together could cause our data to lose significance, i.e. to become "washed out" in terms of significance. Therefore we must utilize practices that limit the number of uncertainties we consider for any particular problem. This is where the studies of Statistical Uncertainty Analysis and Sensitivity Analysis have resulted from.[8]

Uncertainty (and Sensitivity) Analysis continue to be vitally important to Nuclear Power Plants (NPPs), as well as many other fields. For NPPs in particular, the fields of uncertainty and sensitivity analysis are vitally important, as margins of safety for plant operation are a critical aspect of licensing and plant operation.

In 1988, the USNRC (United States Nuclear Regulatory Commission) issued a revision to the emergency core coolant system (ECCS) which allows for the use of best estimate plus uncertainty methods in safety analysis of LWRs (Light Water Reactors). In support of this licensing revision, the code scaling, applicability and uncertainty (CSAU) methodology was developed. As a part of that methodology, the Phenomena Identification and Ranking Table (PIRT) was developed.[14]

The PIRT process is a structured and facilitated elicitation process in which experts are asked to rank various phenomena pertaining to a particular scenario. The phenomena are typically classified as "high", "medium", or "low".[6] The PIRT process of today also typically includes the utilization of best-estimate codes to assist in the ranking process of phenomena.

We wish to find a way to mathematically evaluate the accuracy of PIRTs, and to provide insight to validate and/or to make suggested changes to a given PIRT.

To do this, we utilized a RELAP5-3D input deck and a corresponding PIRT to determine variables of interest. Using those, we were able to create a variable specification file (spec file), as well as a template input file. We designed a program which was able to take the template input file and a specification file for the study and generate more specification files which would allow the study to be broken into sets. Then each set was run individually on the INL's supercomputer cluster (using a Python script), where up to 1,737 input files were created using our program, and run with a RELAP5-3D executable. The data was collected using the python script, and then a statistical analysis was conducted. A depiction of this process can be seen in Figure 1.

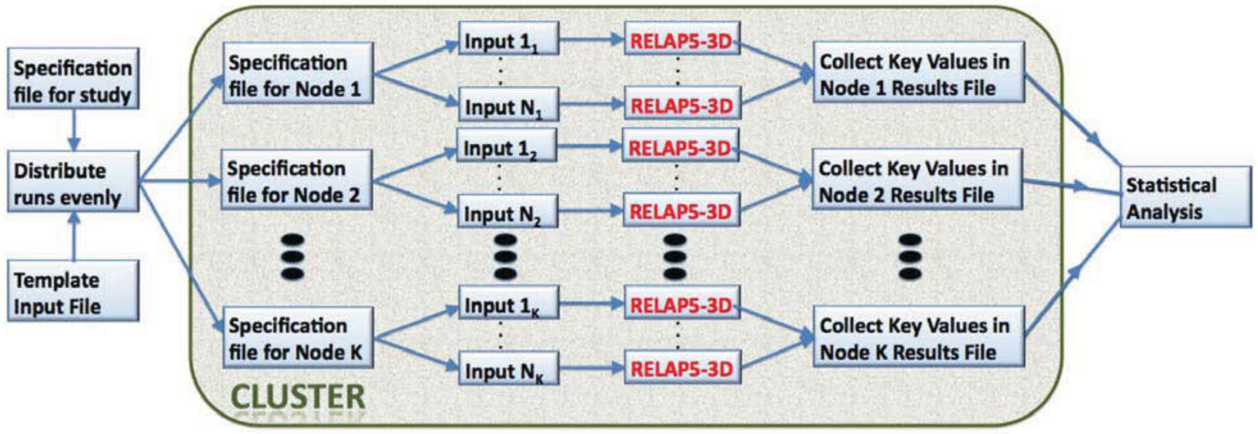

Specification file for study
Distribute runs evenly
Template Input File
Specification file for Node 1
Specification file for Node 2
Specification file for Node K
Input 1_1
Input N_1
Input 1_2
Input N_2
Input 1_K
Input N_K
RELAP5-3D
Collect Key Values in Node 1 Results File
Collect Key Values in Node 2 Results File
Collect Key Values in Node K Results File
Statistical Analysis
CLUSTER

Figure 1: Algorithmic Design

We will begin by listing general statistical definitions in Section 3.1 on Page 8, defining project specific definitions in Section 3.2 on Page 10, and then discussing Coefficients of Correlation in Section 4 on Page 11. We will discuss general properties of correlation coefficients in Section 4.1 on Page 11, and discuss three types of correlation coefficients in Sections 4.2, 4.3, and 4.4 on Pages 14, 15, and 16. The relationship between the three is discussed in Section 4.5 on Page 19. We then discuss our experiments in Section 5 on Page 20. We will conclude by discussing our results in Section 6 on Page 35, summarizing our findings and discussing future work following from this project in Section 7 on Page 37.

3 Standard Formulas

3.1 General Definitions

For a general closed non-degenerate interval, $[c, d]$, with n points equally spaced within the interval (i.e. uniformly distributed), the increment size of the interval, ξ, is $\xi = \dfrac{d-c}{n-1}$.

The probability that a sample point x will occur is the proportion of ocurrences of the sample point in a long series of experiments, and is denoted by $P(x)$. $P(x) \in [0, 1]$.

The probability density function (PDF) of a continuous random variable X, denoted by $f(x)$, is defined such that $\int_{-\infty}^{\infty} f(x)dx = 1$. The probability that an observation lies between x_1 and x_2 is defined as: $\mathrm{P}(x_1 < X < x_2) = \int_{x_1}^{x_2} f(x)dx$. The mean ($\mu_x$) or expected value $E[X]$ of the continuous random variable X, is $E[X] = \int_{-\infty}^{\infty} xf(x)dx$.

Other expectations are mathematically useful and important. We may define expecation more generally for $g(X)$, a function of the continuous random variable X, by:

$$E[g(X)] = \int_{-\infty}^{\infty} g(x)f(x)dx.$$

The variance of the continuous random variable X, where $\mu_x = E[X]$ is constant, is defined as:

$$\text{variance}(X) = \sigma_x = E[(X-\mu_x)^2] = \textstyle\int_{-\infty}^{\infty}(x-\mu_x)^2 f(x)dx = E[X^2] - \{\mu_x\}^2.$$

The covariance of two random variables, X and Y with PDF $f(x,y)$, is defined as:

$$\text{Cov}(X,Y) = E[(X-\mu_x)(Y-\mu_y)] = \textstyle\int_{-\infty}^{\infty}\int_{-\infty}^{\infty}(X-\mu_x)(Y-\mu_y)f(x,y)dydx =$$
$$\textstyle\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} xyf(x,y)dxdy - \mu_x E[X] - \mu_y E[Y] + \mu_x\mu_y = E[XY] - \mu_X\mu_Y$$

Let X be a random variable with n data points. Let each point be denoted by x_i for $i = 1, 2, ..., n$. (i.e. the data points are $x_1, x_2, ..., x_n$.)

The sample mean of X, $\hat{\mu_x}$, is defined as[2]:

$$\hat{\mu_x} = \frac{1}{n}\sum_{i=1}^{n} x_i.\text{[2, p 39,81-82,87,106-113]}$$

The sample variance of X, $\hat{\sigma}^2$, where $\tilde{s}^2$ is an unbiased estimate of $\hat{\sigma}^2$, is defined as:

$$\hat{\sigma_x^2} = E[\tilde{s}^2] = \frac{\sum_{i=1}^{n}\left(x_i - \mu_x\right)^2}{n-1} = \frac{\sum_{i=1}^{n}\left(x_i\right)^2 - \frac{1}{n}\left(\sum_{i=1}^{n} x_i\right)^2}{n-1}.\text{[11, p. 11]}$$

The sample covariance of two random variables, X, and Y, each with n data points, where each point of X is denoted by x_i and each point of Y is denoted by y_i for $i = 1, 2, ..., n$, is

$$\widehat{\text{Cov}}(X,Y) = \frac{1}{n-1}\sum_{i=1} n\left[(x_i - \hat{\mu_x})(y_i - \hat{\mu_y})\right].$$

[2] $\hat{\mu_x}$ is also denoted $\bar{x}$

3.2 Project Specific Definitions

In our input modification program, we are given a base file name, the number of variables, the number of sets the study will be divided into, the number of groups of variables,[3] as well as a list of variables and some basic variable information. The variable information includes the minimum and maximum values of the interval over which the variable ranges, the number of points within the interval that the variable will have (i.e. sample size), the standard deviation of the variable, the type of probability distribution the points will have within the interval (currently only the uniform is available in our program), and the group number of the variable.

For a specific variable, v, with n points, a minimum of a, and a maximum of b (where $a < b$), for which the points are distributed uniformly throughout the interval, the following formulas follow from the general definitions.

$$\xi_v = \frac{b-a}{n-1}$$
$$\hat{\mu_v} = \frac{b-a}{2}$$

$$\begin{aligned}\hat{\sigma_v} &= \frac{1}{n}\sqrt{\frac{1}{n-1}\sum_{i=1}^{n}\left[a+\xi_{\mathrm{v}}(i-1)-\hat{\mu_v}\right]^2} = \frac{1}{n}\sqrt{\frac{1}{n-1}\sum_{i=1}^{n}\left[a+\frac{b-a}{n-1}(i-1)-\frac{b-a}{2}\right]^2} \\ &= \frac{1}{2n(n-1)}\sqrt{\frac{1}{n-1}\sum_{i=1}^{n}\left[3an-a-bn-b-2ai+2bi\right]^2}\end{aligned} \tag{1}$$

[3]In some cases, groups of variables may need to be changed simultaneously in input file generation, and in those cases, groups are needed. In the case where a set of variables needs to be simultaneously changed all of those variables will be in one group, and in the case where only one variable needs to be changed it will be placed in its own group. For instance, in the LOFT study discussed later in the paper, variables 1-24 represent peaking factor and are in group one, variable 25 represents fuel clad gap width and is in group 2, variables 26-43 represent fuel thermal conductivityand belong to group 3, etc.

4 Coefficients of Correlation

Let X and Y be two random variables with a bivariate probability distribution.

The covariance of X and Y, denoted by $\mathrm{Cov}(X, Y)$, is a measure of the amount of association (or, equivalently, correspondence) and the direction of association between X and Y.

The most important measure of the degree of correlation between two variables is the correlation coefficient. It standardizes the covariance in such a way as to eliminate the dependency that covariance has on the scale of measurement for the data set. [2, 187]

4.1 General Properties of Correlation Coefficients

A correlation coefficient is said to measure the strength of the relationship between variables. It is not an inferential statistical test. Rather, it is a descriptive statistical measure that depicts the strength of the relationship between two or more variables.

Let ζ denote a correlation coefficient of X and Y.

ζ should satisfy four key criteria.[3]

1. $\zeta \in [-1, 1]$
2. If the larger (smaller) values of X tend to be paired with the larger (smaller) values of $Y \Rightarrow \zeta > 0$, $\zeta \to +1$ if the correlation is strong. $\Rightarrow \exists$ a positive correlation between X and Y.
3. If the larger (smaller) values of X tend to be paired with the smaller (larger) values of Y, $\Rightarrow \zeta < 0$ and $\zeta \to -1$ if the correlation is strong. $\Rightarrow \exists$ a negative correlation between X and Y.

4. If the values of X tend to be randomly paired with the values of $Y \Rightarrow \zeta \to 0$. [4] When $\zeta \approx 0$, $\Rightarrow$ X and Y are uncorrelated (or, equivalently have no correlation, or zero correlation.)

Correlational information does not provide any conclusions regarding cause and effect, rather it indicates the degree of statistical relationship between two variables. [11, p 72]

The absolute value of ζ, $|\zeta|$, indicates the strength of the relationship between X and Y. As $|\zeta| \to 1$, the stronger the relationship between X and Y, and the more accurately a researcher can predict the value of y_i given x_i. As $|\zeta| \to 0$, the weaker the relationship between X and Y, and the less accurately a researcher can predict a particular y_i given x_i. When $\zeta = 0$, the relationship between X and Y cannot be predicted as ζ is no more accurate than a prediction based purely upon chance.[11, p 945-946]

The sign of ζ indicates the direction of the relationship between X and Y. $\zeta > 0$ indicates a **direct** relationship and $\zeta < 0$ indicates an **inverse** relationship.

Some general guidelines for the interpretation of ζ are:

- $\zeta \in [0.7, 1] \Rightarrow$ strong direct relationship
- $\zeta \in [0.3, 0.69] \Rightarrow$ moderate direct relationship
- $\zeta \in [0.01, 0.29] \Rightarrow$ weak direct relationship
- $\zeta \approx 0 \Rightarrow$ no consistent pattern which allows for prediction of one variable's values based upon knowledge of the other variable's values
- $\zeta \in [-0.29, -0.01] \Rightarrow$ weak indirect relationship
- $\zeta \in [-0.69, -0.3] \Rightarrow$ moderate indirect relationship

[4] $\zeta \approx 0$ when X and Y are independent. $\zeta = 0 \not\Rightarrow X$ and Y are independent

- $\zeta \in [-1, -0.7] \Rightarrow$ strong indirect relationship. [11, p 72]

This allows us to make a fairly intuitive comparison of the strength of a correlation coefficient to a PIRT ranking: $|\zeta| \in [0.7, 1] \Rightarrow$ 'high', $|\zeta| \in [0.3, 0.7) \Rightarrow$ "medium', and $|\zeta| \in [0, 0.3)$ 'low'. We use this in our AP600 and LOFT studies, the results of which are discussed in Section 6 on Page 35

As with any statistical computation, it is important to determine the significance of the calculation, or in other words, it is crucial that we determine whether the correlation coefficient that we've computed is statistically significant. To determine whether the correlation coefficient is significant, it is common practice to perform inferential statistical tests to evaluate one or more hypothesis concerning the correlation coefficient. [11, p 946] In our studies, we utilized the p-value, sometimes referred to as a prob value or the associated probability or the significance probability[4, p. 18-19], as the inferential statistical test to evaluate the statistical significance of the correlation coefficient. We considered a p-value ≤ 0.05 to indicate statistical significance of the correlation coefficient. The null hypothesis, H_o, is: $\mathrm{H}_o : \zeta = 0$, p-value $\leq 0.05 \Rightarrow$ significant evidence that there is correlation between our two variables. On the other hand, for the same H_o if the p-value $> 0.05 \Rightarrow$ we fail to reject the null hypothesis, i.e. we do not have sufficient evidence that the two variables are correlated. In our studies, if the p-value was too large, we were unable to make any conclusions about that phenomena based on its correlation coefficient.

There are many different methods of calculating correlation coefficients. The three we explore (Pearson, Spearman, and Kendall) are the most frequently used.

4.2 Pearson's Product Moment Correlation Coefficient

The Pearson Product Moment Correlation, denoted by ρ, is a measure of the **linear** relationship between X and Y. It is defined as: $\rho(X,Y) = \frac{\text{Cov}(X,Y)}{\sigma_X \sigma_Y}$. It is a correlation coefficient (and thus meets all four criteria specified in the section above.) In the bivariate normal case, $\rho(X,Y) = 0 \Rightarrow$ independence of X and Y.

The Pearson Product Moment Correlation Coefficient is the most commonly used measure of correlation.[11, p 71]

ρ is invariant under positive linear transformations of the random variables but it is not invariant under all order-preserving transformations, and requires that the two variables have a bivariate normal distribution.[11, p. 947] If X and Y are not approximately normally distributed, then another correlation coefficient should be used.

The most commonly evaluated hypothesis for the PPMCC is: in the population represented by the sample, the two variables have no correlation.[11, p 945]

The statistic computed for the PPMCC will be denoted by $\hat{\rho}$.

The coefficient of determination is $\hat{\rho}^2$, and it represents the proportion of variance on one variable which can be accounted for by variance on the other variable.[11, p 953]

For X and Y of sample size n, using the notation from section 3.1:

$$\hat{\rho} = \frac{\sum_{i=1}^{n} x_i y_i - \frac{1}{n} \sum_{i=1}^{n} x_i \sum_{i=1}^{n} y_i}{\sqrt{\left\{ \left[\sum_{i=1}^{n} x_i^2 - \frac{1}{n} \left(\sum_{i=1}^{n} x_i \right)^2 \right] \left[\sum_{i=1}^{n} y_i - \frac{1}{n} \left(\sum_{i=1}^{n} y_i \right)^2 \right] \right\}}}. \text{ [11, p 950].}$$

We compute the p-values for the Pearson Product Moment Correlation Coefficient by treating $t = \sqrt{\frac{\hat{\rho}^2 (n-2)}{1 - \hat{\rho}^2}}$ as coming from a t-distribution with $n-2$ degrees of freedom.[10, p

18-19]

4.3 Spearman's Rho

Spearman was developed in 1904 and is a bivariate measure of correlation which is used with rank-order data.[11, p 1061] Spearman's is the application of PPMCC to ranked data.

Spearman's determines the degree to which a monotonic relationship exists between two variables.[11, p 1062]

The underlying null hypothesis for Spearman's Rank-order Correlation Coefficient is: in the population represented by the sample, the correlation between the ranks of the two is 0. [11, p 1063]

The sample statistic computed to estimate the value of Spearman's rho (or Spearman's r) will be dentoed by $\hat{\mathfrak{r}_s}$.

For X and Y of sample size n, using the notation from section 3.1 and letting R_{xi} denote the rank of variable x_i, and R_{yi} denote the rank of variable y_i:

$$\hat{\mathfrak{r}_s} = 1 - \frac{6\sum_{i=1}^{n}\left(R_{xi} - R_{yi}\right)^2}{n\left(n^2 - 1\right)}.$$

SAS computes $\hat{\mathfrak{r}_s}$ by ranking the data and using the ranks in the Pearson formula. When ties occur, the averaged ranks are used. The p-value for Spearman is computed by treating $t = \sqrt{\frac{\hat{\mathfrak{r}_s}^2(n-2)}{1-\hat{\mathfrak{r}_s}^2}}$ as coming from a t-distribution with $n-2$ degrees of freedom.[10, p 19] This is the same way the p-value was computed for Pearson, except that $\hat{\mathfrak{r}_s}$ is used in place of $\hat{\rho}$.

4.3.1 Generic Example of Computing Spearman's Rho and the Corresponding P-value

Suppose we have $n = 4$, and the following points, (x_i, y_i), within our data:

$(0.3, 1.5), (0.5, 2.5), (0.6, 2.3),$ and $(0.9, 4.6)$.

Ranking these points, (R_{xi}, R_{yi}): $(1,1), (2,3), (3,2),$ and $(4,4)$.

$$\Rightarrow \hat{\mathfrak{r}}_s = 1 - \frac{6\left((1-1)^2 + (2-3)^2 + (3-2)^2 + (0-0)^2\right)}{4\left(4^2-1\right)} = 0.8, \text{ and } t = \sqrt{\frac{0.8^2\left(4-2\right)}{1-0.8^2}} \approx$$

1.88562, and using a table of selected values $\Rightarrow$ p-value ≈ 0.9.

4.4 Kendall's Tau

Kendall's tau was developed in 1938 and is a bivariate measure of correlation used with rank-order data. The population parameter estimate is denoted by τ. The sample statistic computed to estimate the value of τ will be represented by $\hat{\tau}$.

Kendall's tau measures the degree of agreement between two sets of ranks with respect to the relative ordering of all possible pairs of subjects/objects.[11, p 1079]

For X and Y of sample size n, using the notation from section 3.1 and letting R_{xi} denote the rank of variable x_i and R_{yi} denote the rank of variable y_i.

A pair, (R_{xi}, R_{yi}) and (R_{xj}, R_{yj}) is said to be **concordant** if $(R_{xi} - R_{xj})(R_{yi} - R_{yj}) > 0$, or **discordant** if $(R_{xi} - R_{xj})(R_{yi} - R_{yj}) < 0$. Let n_c denote the number of concordant pairs of ranks and let n_d denote the number of discordant pairs in the ranks. Then $\hat{\tau}$ is defined as: $\hat{\tau} = \dfrac{n_c - n_d}{\left\{\dfrac{n(n-1)}{2}\right\}}$.

Let t_k denote the number of tied x values in the kth group of tied x values, u_l denote the number of tied y values in the lth group of tied y values, n denote the sample size, and define $\text{sgn}(z)$ as:

$$\text{sgn}(z) = \begin{cases} 1, & \text{if } z > 0 \\ 0, & \text{if } z = 0 \\ -1, & \text{if } z < 0 \end{cases}.$$

Let $T_0 = \frac{1}{2}n(n-1)$, $T_1 = \sum_k \frac{1}{2}t_k(t_k - 1)$, and $T_2 = \sum_l \frac{1}{2}u_l(u_l - 1)$.

Then we can define $\hat{\tau}$ as: $\hat{\tau} = \dfrac{\sum_{i<j} \big(\text{sgn}(x_i - x_j)\text{sgn}(y_i - y_j)\big)}{\sqrt{(T_0 - T_1)(T_0 - T_2)}}$.

Let $s = \sum_{i<j} \big(\text{sgn}(x_i - x_j)\text{sgn}(y_i - y_j)\big)$.

Define the variance of s, $V(s)$, as: $V(s) = \dfrac{v_0 - v_t - v_u}{18} + \dfrac{v_1}{2n(n-1)} + \dfrac{v_2}{9n(n-1)(n-2)}$,
where $v_0 = n(n-1)(2n+5)$, $v_t = \sum_k t_k(t_k - 1)(2t_k + 5)$, $v_u = \sum_l u_l(u_l - 1)(2u_l + 5)$,
$v_1 = \left(\sum_k t_k(t_k - 1)\right)\left(\sum_l u_l(u_l - 1)\right)$, and $v_2 = \left(\sum_k t_k(t_k - 1)(t_k - 2)\right)\left(\sum_l u_l(u_l - 1)(u_l - 2)\right)$.

We compute the p-values by treating $\dfrac{s}{\sqrt{V(s)}}$ as coming from a standard normal distribution.[10, p. 20-21]

4.4.1 Generic Example of Computing Kendall's Tau and the Corresponding P-value

To illustrate the computation of Kendall's tau, as well as the corresponding p-value, we will utilize the same example as was used in the generic example of computing Spearman's rho and its corresponding p-value in Section 4.3.1 on Page 16.

Suppose we have $n = 4$, and the following points, (x_i, y_i), within our data:

$(0.3, 1.5), (0.5, 2.5), (0.6, 2.3),$ and $(0.9, 4.6)$.

Ranking these points, (R_{xi}, R_{yi}): $(1,1), (2,3), (3,2),$ and $(4,4)$.

The concordant pairs are: $(1,1),(2,2)$ & $(1,1),(3,2)$ & $(1,1),(4,4)$ & $(2,3),(4,4)$ and $(3,2),(4,4)$.

The discordant pair is: $(2,3),(3,2)$.

$\Rightarrow n_c = 5$ and $n_d = 1$.

$$\Rightarrow \hat{\tau} = \frac{5-1}{\left\{\frac{4(4-1)}{2}\right\}} = \frac{2}{3}$$

If we use the second formula for $\hat{\tau}$, we get the same result.

$s = 5 - 1 = 4$, $t_k = 0$, and $u_l = 0$.

$T_0 = \frac{1}{2} \times 4 \times (4-1) = 6$, $T_1 = 0$, and $T_2 = 0$.

$$\Rightarrow \hat{\tau} = \frac{5-1}{\sqrt{(6-0)(6-0)}} = \frac{2}{3}.$$

We can now compute $V(s)$.

$v_o = 4 \times (4-1) \times (2 \times 4 + 5) = 156$, $v_1 = 0$, and $v_2 = 0$.

$$\Rightarrow V(s) = \frac{156 - 0 - 0}{18} = \frac{26}{3}$$

$$\Rightarrow \frac{s}{\sqrt{V(s)}} = \frac{4}{\left\{\sqrt{\frac{26}{3}}\right\}} \approx 1.35873$$

Using a table of values for the standard normal distribution, $\Rightarrow$ p-value $\approx 1 - 0.3708 = 0.6292$.

4.5 Discussion of Pearson, Spearman, and Kendall

It is recommended that for interval or ratio data in the bivariate case that the Pearson product-moment correlation coefficient be used, for ordinal or rank order data of bivariate case that Spearman's rank-order correlation coefficient or Kendall's tau be used.[11, p 117]

Pearson is a stronger correlation coefficient[5] than Spearman or Kendall, but has additional restrictions, such as the requirement of parametric data.[6] Spearman's rho is more commonly used than Kendall's tau for two primary reasons. The computations for tau tend to be more tedious than those completed when calculating rho.[7][11, p. 1080] The second reason is that when the sample is derived from a bivariate normal distribution, Spearman's rho tends to provide a reasonably good approximation of Pearson, while Kendall's tau will not. An advantage of Kendall's tau is that it has a sampling distribution that approaches normality very quickly. Spearman's rho, on the other hand, needs a fairly large sample size to employ the normal distribution to approximate the sampling distribution of rho.[11, p. 1080] As a general rule, $|\tilde{\tau}_c| < |\tilde{\mathfrak{r}_s}|$ for a set of data, and $\frac{\hat{\tau}}{\hat{\mathfrak{r}_s}} \to (0.67)^2$.[11, p. 1080][8]

In Sections 4.3.1 and 4.4.1 we had $\hat{\tau} = \frac{2}{3}$ and $\hat{\mathfrak{r}_s} = 0.8$, $\Rightarrow |\hat{\tau}| < |\hat{\mathfrak{r}_s}|$, which follows the general rule. However, $\frac{\hat{\tau}}{\hat{\mathfrak{r}_s}} \approx 0.83333 \neq (0.67)^2$, yet, it does not disprove the general rule as we just used a generic example which had a very small sample space.

[5]'Stronger' in this case is referring to a higher degree of statistical significance.

[6]Recall, it specifically requires a bivariate distribution which is approximately normal.

[7]With the advances in computing, and the introduction of statistical software packages, this has become less of an issue.

[8]This is not the case when the value of Spearman's rho is 1 or -1. In that case, Kendall's tau will also be 1 or -1 respectively. This is the same for the case when Kendall's tau is known to be 1 or -1.

5 Experimental Information

5.1 RELAP5-3D

5.1.1 Terminology and Basic Deck Requirements[7, p. A1-1]

Term/Accronym	Definition/Description
BE	Best-Estimate
Card	an 80-character record in the input deck
Card #	the first field on the data card. It is an unsigned integer, and is used to specify the component number the card supplies information about.
Comment Card	identified by a '∗' or a '$' as the first nonblank character. Blank cards are treated as comment cards. With the exception of printing out their contents, there is no processing of comment cards.
Continuation Card	identified by a '+' as the first nonblank character on the card, may follow a data card or another continuation card. Fields on each card must be completed on that particular card (i.e. a field may not continue from one card onto the next.)
Data Card	contains varying numbers of fields that may be integer real, or alphanumeric. If a matching card number is found, the data card that appears last in the deck is the card that is used.
Input Deck	term used to describe the input files
Terminator Card	identified by a '/' or a '.' as the first nonblank chearcter. Comments may follow '/' or '.' on a particular card
Title Card	identified by an '=' sign as the first nonblank characters and contains no more than 80 characters including the '='.
Word	fields on the data card following the card number, a word is terminated by comma(s) or space(s). Numeric words must begin with a digit (0-9) or a sign (+ or -), or a decimal point.

A RELAP5-3D input deck typically consists of at least one title card, optional comment cards, data cards and a terminator card.

5.1.2 Safety Analysis of NPPs & BE Codes

To ensure safety in the reactor design, operator training, and upgrades to the Nuclear Power Plant (NPP), Best Estimate (BE) codes are frequently used to model NPPs and to analyze their safety.

A *Best Estimate* code is a system code that is capable of predicting/ modeling physical phenomenon, free of any conservatism regarding selected acceptance criteria, and able to provide a sufficiently detailed model to describe the relevant processes.[9, p. 193] RELAP5-3D is the BE code used in this study.

5.1.3 About

RELAP5-3D is a fully integrated, multi-dimensional thermal-hydraulic and kinetic modeling program. It is a BE code developed at the INL, which serves as a modeling and simulation tool to support engineering design and safety analysis of nuclear reactors. It is also used for non-nuclear fields such as fossil power plants, oil and gas pipelines, municipal steam supply systems.

5.1.4 Running RELAP5-3D

Upon performing the calculations of the input model, RELAP5-3D produces three output files: a '.p' file, a '.plt' file, and a '.r' file. The '.p' file is a printed output file, the '.plt' file is a plot file, and the '.r' file is the restart file. In our studies, we were primarily concerned with examining the printed output file. We wrote a Python script, which is described in Section 5.5 on Page 33. This script locates our desired ouput parameter value from the '.p' file, and then places that value into a parameter value file which is created with the input modification program described in Section 5.4 on Page 29.

5.2 AP600 Deck

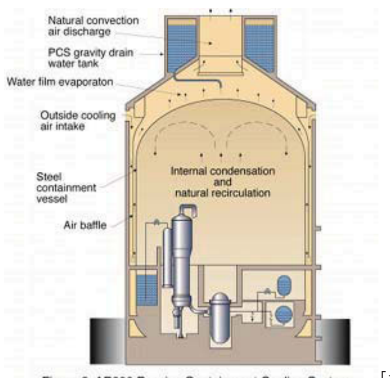
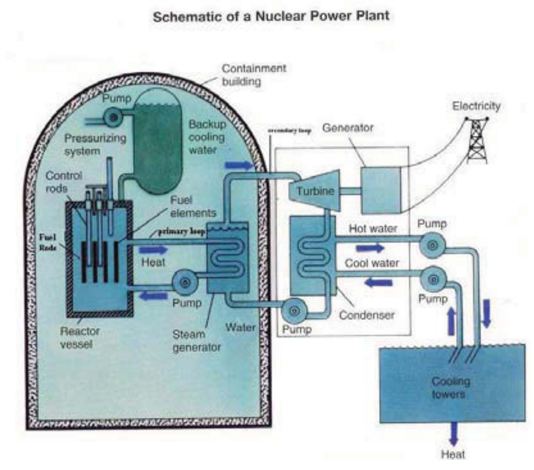

Natural convection air discharge
PCS gravity drain water tank
Water film evaporaton
Outside cooling air intake
Steel containment vessel
Air baffle
Internal condensation and natural recirculation

Figure 3. AP600 Passive Containment Cooling System [13]



[5]

5.2.1 Background on the AP600 Nuclear Power Plant

The AP600 Nuclear Power Plant (NPP) was designed by Westinghouse as a part of the cooperative U.S. Department of Energy (DOE) and the Electric Power Research Institute (EPRI) Advance Light Water Reactor Program (ALWR). [12]

For safety, the AP600 relies on operation of passive systems. The Phenomena Identification and Ranking Table (PIRT) compiled by Burtt, et al (used in this paper, see the references section, number [1]) addresses AP600 behavior expected during small break loss-of-coolant, main steam line break, and steam generator tube rupture accidents. [1]

5.2.2 Variable Names, Representation, & Corresponding PIRT[1] Phenomena

Variable	Representing	Corresponding to (in PIRT)	PIRT Ranking
VAR1	Fission-Product Yield (i.e. decay-heat)	Core Power	High
VAR2	Roughness	Flow Resistance in the IRWST	High
VAR3	Form Loss	Flow in Accumulator (pressurized volume)	High
VAR4	Form Loss	Flow in Accumulator (pressurized volume)	High
VAR5	Area (ADS-4)	ADS Energy Release	High
VAR6	Area (Break Valve)	Mass Flow in Break[9]	High
VAR7	Area (Orifices)	Injection Line - CMT	High
VAR8	Initial Level of Pressure	Level in CMT	High
VAR9	Roughness	ADS Flow Resistance	Medium
VAR10	Roughness	ADS Flow Resistance	Medium
VAR11	Fouling Factor	PRHR (Heat Transfer between PRHR & IRWST)	Medium
VAR12	Fouling Factor	Steam Generators (Primary to Secondary Heat Transfer)	Medium
VAR13	Roughness	PRHR-Flow Resistance # 1	Low
VAR14	Roughness	PRHR-Flow Resistance #2	Low
COREMIN	Control Variable 116	Minimum Value of the Core Collapsed Liquid Level [10]	N/A

5.2.3 Studies with the AP600 Deck

Using the AP600 deck, we conducted four studies. These studies were based upon the size of break in the cold leg. Each of the 13 variables[11] had three values which were evenly distributed throughout the interval (i.e. a minimum, a mean, and a max). Variable 6 denoted the size of the break and was used to create the four different studies. To have

[9]This was used to create the four different studies: 2 inch Break, 4 inch Break, 6 inch Break, and 8 inch Break.

[10]COREMIN is the key output parameter used in the AP600 studies

[11]i.e. VAR1-5, VAR7-14. VAR 6 was used to vary the break size.

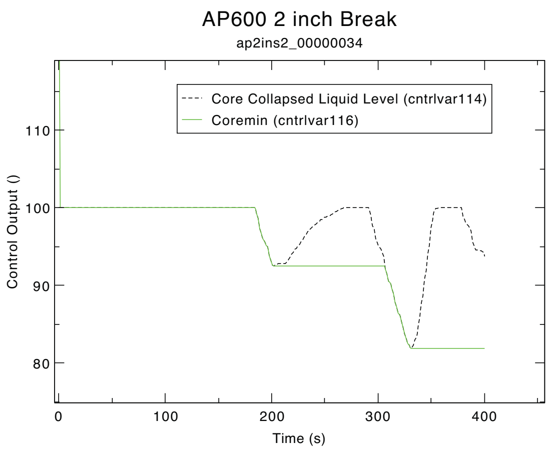
every possible combination of variable values, this would mean creating $3^{13} = 1,594,323$ input files.

Performing even a single study with 3^{13} runs of 400 seconds would require 7,381 days on a single processor. Even utilizing the INL cluster, described in Section 5.5 on Page 33, which has 32 cores per node and 12 nodes, it would require 19 days to run the study. There are many ways to reduce the number of runs in a study. We decided to investigate the use of grouping our studies into smaller sets to reduce the number of required runs for each study. We first separated each study into two or three sets. The variables that had correlation coefficients that were ranked high or medium among the sets would then be grouped together and run again. Through experimentation, we found that separating the studies into smaller sets did not significantly affect the calculations of the relative correlation coefficients and substantially reduced the amount of time needed to run our studies.

The size of the break dictates how long the transient runs. In order to further decrease the run-time of our studies, we plotted each of the four transients (2 inch, 4 inch, 6 inch, 8 inch) to determine what the duration of the transient should be for each respective study. The duration of the transient that was selected for each study encompassed the time when the collapsed core level reached its lowest height. These plots can be seen in Figures 2, 3, 4, and 5, respectively.[12] We determined that it was effective to run the 2 inch break transient for 400 seconds, the 4 inch break transient for 200 seconds, the 6 inch break transient for 200 seconds, and the 8 inch break transient for 75 seconds.

The results from the 'top study' of each of the cold-leg break studies are presented in Section 6.1.1 on Page 35.

[12]coremin represents the key output parameter, which is a control variable we created that takes the minimum of control variable 114 (collapsed core liquid level). Control Output() is the (Percentage of) Collapsed Core Liquid Level.



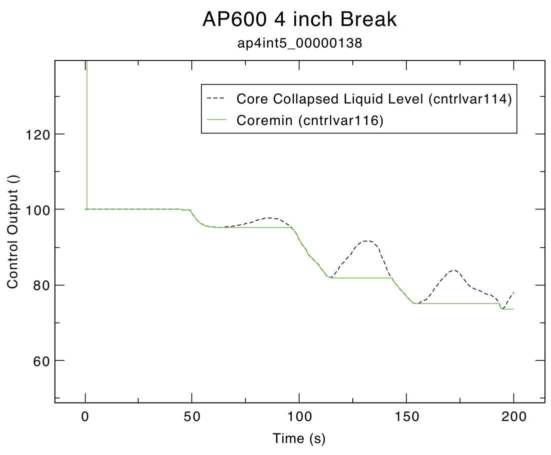
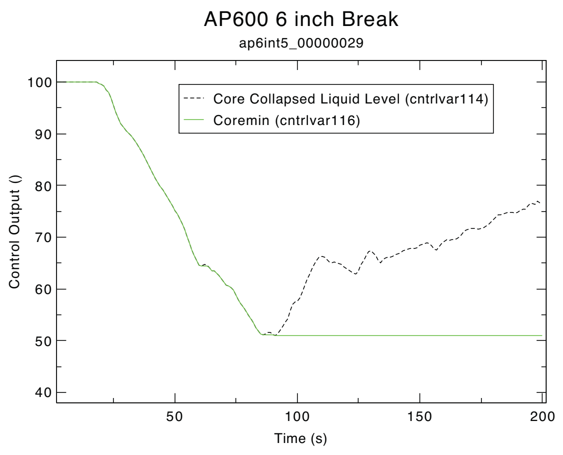


Figure 2: 2 inch Break Transient Plot

Figure 3: 4 inch Break Transient Plot

Our first study that we conducted was on a four inch break. To decrease run-time, we broke the study into two sets: the first set had only variables 1-5 and 7-8 (i.e. the PIRT-ranked 'high' variables, note that as stated previously, variable 6 is the break size and thus will not be adjusted within the 4 inch study) varying, while the other 6 variables were held at the nominal (or mean) value. The first set was $3^7 = 2,187$ input files. The second set had only variables 9-14 (i.e. the PIRT-ranked mediums and lows) varying, while the other 7 variables were held at their respective nominal values. The second set consisted of $3^6 = 729$ input files. After both sets were complete, and analyzed using SAS, we created a new study, which was based on the top 5 most correlated variables from both 4 inch studies: VAR1, VAR8, VAR11, VAR12, and VAR13. Just as before, all five variables varied between the three values while the other 8 variables were held at their respective nominal values. This created $3^5 = 243$ input files.

Our second study was on a 2 inch cold leg break. We put this study into three sections (the other variables are held at their respective nominal values): variables 1-5 varying ($2^5 = 243$ input files), variables 7-10 varying ($2^4 = 81$ input files), and variables 11-14 varying ($2^4 = 81$ input files). After all three sets were analyzed using SAS, we created a new study, based on



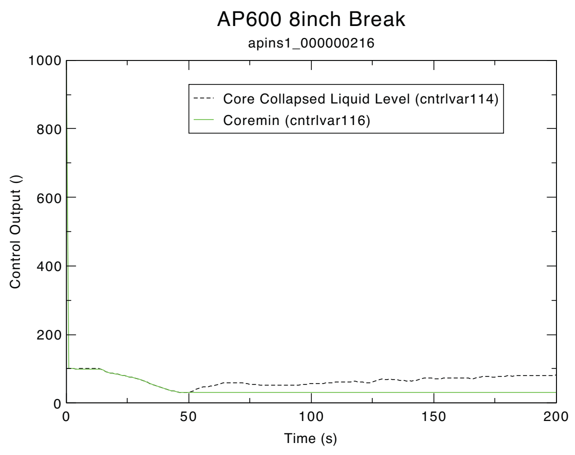


Figure 4: 6 inch Break Transient Plot

Figure 5: 8 inch Break Transient Plot

the top 4 most correlated variables from the three sets: VAR1, VAR8, VAR12, and VAR13. As with the four inch study, we varied these variables between the three variables, while the other 9 variables were held at their respective nominal values. This created $3^4 = 81$ input files.

Our third study was on a 6 inch cold leg break. We split the study into 3 sections, which were identical to the sections for the 2 inch break study. After the three sets were analyzed with SAS, a new study was created based on the top 5 most correlated variables from the three sets: VAR1, VAR7, VAR8, VAR12, and VAR13. These were varied as with the 2 inch and 4 inch break studies: the 5 variables were varied by 3 values, and the other 8 were held constant at their respective nominal values. This created $3^5 = 243$ input files.

Our final study of the AP600 deck was on an 8 inch cold leg break. As with the 2 inch and 6 inch break studies, we split the 8 inch study into the same sets. After the three sets were analyzed using SAS, a new study was created based on the top 4 most correlated variables from the three sets: VAR1, VAR8, VAR12, and VAR13, creating $3^4 = 81$ input files for this section of the study.

In all of the studies, there were variables which had their respective p-value > 0.05. This does not indicate that these variables were statistically insignificant, or that they were unimportant to the minimum core level. Rather, it indicates that we cannot make a conclusion regarding their correlation to the minimum core level, and that further investigation is needed for these variables.

5.3 LOFT Deck

5.3.1 Background Information on the LOFT Nuclear Power Plant

Following the 1988 revised emergency core cooling system rule for LWRs which allows the use of best estimate plus uncertainty methods in safety analysis, a study was conducted at the Idaho National Laboratory (INL) based on a cold leg large break loss of coolant accident test in the Loss of Fluid Test (LOFT) experimental facility. The LOFT facility was a 50 MW PWR that was designed to model a large break loss of coolant (LBLOCA) in a commercial pressurized water reactor (PWR). The facility was operational from the late 1970's to the mid 1980's.[15]

The input deck used in this study was the same input deck used by G.E. Wilson and C.B. Davis in their "Best Estimate Versus Conservative Model Calculations of Peak Clad Temperature: An Example Based on the LOFT Facility" paper (see reference [15].)

5.3.2 Variable Names and Corresponding Phenomena [15]

Group Number	Variable(s)	Representing
1	VAR1-VAR24	Peaking Factor (i.e. core power fractions)
2	VAR 25	Fuel Clad Gap Width
3	VAR26-VAR43	Fuel Thermal Conductivity
4	VAR 44	Clad to Coolant Heat Transfer (i.e. fouling factor)
5	VAR45-VAR46	Break Discharge Coefficient
6	VAR47-VAR51	Pump Degradation
n/a	Control Variable 234	Peak Clad Temperature [13]

5.3.3 LOFT Studies

We conducted two studies using the LOFT deck. Both studies values were selected based upon G.E. Wilson and C.B. Davis' paper. For both studies, the key output parameter in the study was Peak Clad Temperature (PCT).

In the first study, each of the six groups of variables had 3 points uniformly distributed (i.e., minimum, mean, and maximum). All groups used minimum, mean, and maximum for their three values. However, the variables within each group often had differing maxima, minima, and means. For example, in group one there were 24 different sets of minimum, mean, and maximum, yet all variables in the group needed to vary in the same way. For instance, each of the 24 variables in group one all needed to experience their respective minimum simultaneously. Likewise, each of the 24 variables in group one had to achieve their mean simultaneously. Similarly, the 24 variables all experienced their respective maxima simultaneously. There were $3^6 = 729$ input files in the first study.

In the second study, each of the six groups had 5 different values (i.e., minimum, low-mean, mean, high-mean, and maximum.) The groups of variables functioned the same way for the second study as the first study, just with additional values. In other words, for

[13]Peak Clad Temperature is the key output parameter used in the LOFT study.

group one, the 24 variables were varied simultaneously, with each variable within the group experiencing it's particular low-mean simultaneously. Likewise, each of the 24 variables experienced its respective high-mean simultaneously. There were $5^6 = 15,625$ input files in the second study.

The results from both studies with the LOFT deck are presented in Section 6.2 on Page 36.

5.4 Input Modification Program

In order to run the AP600 study and the LOFT study, a Fortran 90© program was created. This program uses a marked input deck and study specification file(s) to create input files with input values varying as specified in the specification file(s).

5.4.1 Marked Input File

After an input deck[14] and the variables of interest (sometimes referred to as input parameters) as well as the key output parameter have been selected, the user must determine how these selected input parameters should vary. Once that is determined, the user should create a 'spec file' (or Specification File). Specification Files are discussed in Section 5.4.2. The base input deck should then be 'marked' by the user- this consists of locating the variables of interest (i.e. card number and word) within the deck, and then placing a '\$ XXXX'[15] in place of the current value on the card in the deck.[16] This process should be undertaken simultaneously with the creation of the 'spec file' so as to avoid mistakes. [17] After the base deck is completely marked, we refer to it as the "marked input deck."

[14]To avoid confusion, we will refer to this input deck as the base input deck.

[15]XXXX denotes the variable name of that particular input parameter

[16]In our studies, we selected 'VARX' , where $X = 1, 2, 3, ...$ to denote the various variables.

[17]There are very specific parameters for the length of character name, so care should be taken. Details of this can be found within the main program- see Appendix A for further details. Additional care should be taken when replacing values as a card may only contain 80 characters, so the user may wish to consider adjusting spaces within a given card or splitting the card (i.e. making a continuation card.) See reference [7].

5.4.2 Specification Files

The specification file(s) or 'spec file(s)', contains all of the necessary information for a given study. The first line of a spec file indicates the type of spec file it is. There are two types of spec files: (1) a Generator Specification File and (2) a Node Specification File. These are discussed in the Subsections 5.4.2(a) and 5.4.2(b) respectively. Both types of spec files contain: the base name of the files that are generated (this is also the name (without the extension) of the marked input file), the number of variables in the study, the number of nodes in the study [18], which node (or set) of this spec file corresponds to, the number of groups in the study[19], as well as variable name[20], the minimum value of the variable, maximum of the variable, standard deviation of the variable, number of variations of the variable (i.e. number of points for the variable), the distribution type of the variable, and the group number the variable belongs to[21]. A template of a spec file can be seen in Appendix B on Page 51.

4.5.2(a) Generator Specification File A "Generator Specification File" or "generator spec file" is a spec file with generate on the first line of the spec file. When combined with the executable produced by the input modification program, described in Section 5.4.3, it produces a specified number of *node* spec files, each of which has a different node number[22]. The node number corresponds to the INL cluster node, where each node has multiple cores. The INL cluster is discussed more in Section 5.5 on Page 33. To see an example of a

[18]The 'number of nodes' refers to the number of studies within the study. i.e. We break each study into multiple sets (or studies) so that it is easier to run the study on the cluster. This is discussed in greater detail in Section 5.5.

[19]The 'number of groups' refers to the number of variables that vary INDEPENDENTLY within the study- i.e., in our LOFT studies, we had variables which had to vary simultaneously (dependently). For instance, variables 1-24 all varied simultaneously, variable 25 varied independently, variables 26-43 varied simultaneously, variable 44 varied independently, variables 45-46 varied simultaneously, and variables 47-51 varied simultaneously, so we had 6 groups. In the case of the AP600 studies, all 14 variables varied independently and so the number of groups was equivalent to the number of variables, 14.

[20]Note that the '$' is not written into the spec file- though it is marked that way in the marked input file. The '$' is appended within the main program, see Appendix A for more information.

[21]i.e. VAR1-24 in LOFT study are in group 1, and so a 1 is written in that variable's line.

[22]Using the notation from the template spec file in Appendix B, the specified number of spec files is #nodesInStudy, and the node number corresponds to 'currentNode'.

generator spec file, see Appendix C on Page 52.

The motivation for generating separate node spec files was cluster supercomputer limitations. Generating thousands of input files on a single processor, then moving them to the nodes where they would run ties up communication resources need for other data flow and can seriously impair performance. It is far more efficient to generate the RELAP5-3D input files on the nodes where they will run, thereby eliminating all such input file movement.

4.5.2(b) Node Specification File A "Node Specification File" or "node spec file" is a spec file with run on the first line of the spec file. When combined with the executable produced by the input modification program, described in Section 5.4.3, it produces the input files corresponding the particular node number of the node spec file. To see an example of a node spec file, see Appendix D.

5.4.3 Main Program

The main program, input_mod_gen.f90, is included as Appendix A on Page 40. Its purpose is to create an entire statistical study by generating input relating to a PIRT analysis. It has three purposes:

1. Interpret the "generator specification file" and create a number of "node specification files."

2. Produce all RELAP5-3D input files for a given node of the cluster supercomputer from the "node specification file" and a single RELAP5-3D input template file.

3. Handle the situation where a group of input parameters vary together (are 100% correlated).

The main program is run first to create a node specification file for each cluster node; a node may have up to 32 cores that can each run RELAP5-3D independently and simultaneously.

The creation of the node spec files from a generator spec file is done through the command line prompt:

```
input_mod_gen.exe -i generator_spec_file_of_my_study
```

This creates the corresponding node spec files and these node spec files are exported to each node along with a copy of the template input file. The main program is run on each node to produce the specific input files of the statistical study, and is run on the cluster via a Python script which is described in Section 5.5 on Page 33. The complete set of possible combinations of input values can be ordered as a set of n-tuples. The set of n-tubles is generated by a recursive algorithm and therefore works for any number of input parameters.

The ordinal number of the n-tuple that is used to generate an input file is assigned as the input file's sequence number and becomes part of its name. Concatenating its sequence number to the base name of the statistical study forms its name. All input files have a unique name, regardless of the node on which it runs. The input files are created by copying the template input file then substituting the variable values corresponding to its n-tuple into the input file for that variable's marker. Markers had the form \$VAR1, \$VAR2, ... \$VARK. When some variables are 100% correlated, K is larger than n (the size of the n-tuples).

The values of the correlated variables are varied together. For example, if group one has 20 variables and a uniform distribution with 3 "levels" in use, namely minimum, mean, and maximum value, the group counts as one variable in the first position of the n-tuple previously discussed. The group has three levels: minimum, mean, and maximum. However, when an input file is generated, the particular "maximum value" of each variable is substituted for its marker (\$VAR1 through \$VAR20). Input files are created on node 3 with the following command:

```
input_mod_gen.exe -i node3_spec_file_of_my_study
```

. The number of input files generated is a function of the number of groups and the number of value levels within the group. For example, if group 1 has 6 variables and 3 levels, group 2 has only one variable with 8 levels, and group 3 has 9 levels, there would be $3 * 8 * = 216$ combinations in the entire statistical study. The program determines the number of input files that are created by each node spec file based upon the #nodesInStudy (combSet in program) by taking the celing function of $\left\{\frac{\text{\# of Input Files in Study}}{\text{\#nodesInStudy}}\right\}$. The second LOFT study had 6 groups of variables, each with 5 values for a total of 15,625 runs. Splitting these runs among 9 node spec files placed 1736 runs each on nodes 1-8 and 1737 runs on node 9.

5.5 Running on the Cluster & Python Scripting

Studies were run on one of the INL's supercomputers, Quark, which has 12 nodes. Users submit jobs, such as our statistical studies, via a batch queuing system.

Due to limited resources particularly on moving large amounts of data from the head node to computational nodes, it is much more efficient to generate the input files on the node as explained in Section 5.4.3 on Page 31.

The Python scripts are included in Appendix E on Page 53. It runs the main program on the head node of the cluster, moves the node specification file onto the cluster's computation nodes, runs RELAP5-3D in parallel on the input files, collects the output from the RELAP5-3D output files, and plaes it in a "study output file" on the line corresponding the input's n-tuple. These files are ported back to the head node and are then combined into a single "study output file."

Thereafter, SAS can be used to perform the statistical analysis.

5.6 Using SAS$^{©}$

Once all of the res files have been created, as described in Section 5.5, we copy them into the local desktop, relabeling the files' extensions as '.csv' (as opposed to '.res'), and adding an additional line to the beginning of the file which typically looks like 'runNumber,VAR1,VAR2,VAR3,....,KEYOUTPUT'. We then import each file into SAS, a statistical software package. We used the SAS package, SAS EnterpriseGuide 4.3, which offers a graphical user interface of SAS 9.2. We then combined the res files from all of the nodes into one data set, through utilization of the 'append table' feature, "tasks \ data\ append table". We then calculated the correlation coefficients using the correlation coefficient function, "tasks \ Multivariate \ Correlation Coefficient". We selected the variables as analysis variables, and correlated them with the key output parameter. The options we selected were Pearson, Hoeffding, Kendall, and Spearman, and we included the Pearson correlation options of covariances, sums of squares and crossproducts. These produced SAS reports, which are found in Appendix F. We were then able to rank the variables accordingly.

6 Results

6.1 AP600 Studies

6.1.1 AP600 Correlation Coefficient Classification

Break	High ζ	Medium ζ	Low ζ
2 inch	Core Power (PIRT High)		PRHR-Flow Resistance # 1 (PIRT Low)
4 inch	Core Power (PIRT High)		
6 inch		Core Power (PIRT High) Level in CMT (PIRT High) PRHR-Flow Resistance #1 (PIRT Low)	SG-Heat Transfer (PIRT Medium)
8 inch[23]		Core Power (PIRT High) Level in CMT (PIRT High)	SG-Heat Transfer (PIRT Medium)

6.1.2 Discussion of the Rankings of the Phenomena

All variables which were not listed in Section 6.1.1 had their respective p-value > 0.05.

In the 2 inch and 4 inch studies, the findings were as we expected based upon the PIRT recommendations. In the 6 inch and 8 inch studies, there were some surprising results. In the 6 inch study, two PIRT-ranked 'highs' had a correlation coefficient classifications of 'medium', and a PIRT-ranked 'medium' had a low correlation coefficient classification, of particular interest was that one PIRT-ranked 'low' had a medium correlation coefficient classification. In the 8 inch study, we saw a similar ranking pattern to that of the 6 inch study, with the exception being that all rankings in the 8 inch study were less than those of the PIRT, specifically, none of the rankings of the correlation coefficient classification

[23]In the 8 inch break, the average of the correlation coefficients for the various phenomena were used. In the other studies, the three correlation coeffients were in agreement. The exact correlation coefficient values can be found in the SAS© Reports in Appendix F on Page 58.

were greater than that of the PIRT. The rankings we saw in the 6 inch study (and in the 8 inch study) which were less than or equal to the PIRT rankings do not cause too much concern as the PIRT was conservative, i.e. if the panel was unsure as to whether something should be ranked as ‘high’ or ‘medium’, they ranked it as ‘high’. The result that surprised us was the phenomena in the 6 inch study which was ranked higher than the PIRT ranking in terms of correlation coefficients.

6.2 LOFT

6.2.1 LOFT Correlation Classification

Classification	**Group Name(s)**
High ζ	Fuel Clad Gap Width
Medium ζ	Clad to Coolant Heat Transfer & Peaking Factor
Low ζ	Break Discharge Coefficient & Fuel Thermal Conductivity
p-value > 0.05	Pump Degradation

6.2.2 Ranking the Phenomena Following the Correlation Computation

After completing the correlation analyses for the LOFT study, we were able to numerically rank the phenomena from 1-5, with 1 representing the variable which was most strongly correlated with PCT and 5 being the variable which was least correlated with PCT. From an engineering perspective, we would rank the phenomena by absolute value of the change in PCT from the minimum value to the maximum value of the phenomena. To approximate that computation, we held the other phenomena at their respective nominal values. $\approx \Delta$t corresponds to the approximate change in temperature when we compare the PCT of the minimum value of the variable to the PCT of the maximum value of the variable, where all other variables are held constant at their respective nominal values. The following table presents the rankings:

Rank	Phenomena	$\approx \Delta$t (Rank by $\approx \Delta$t)
1	Fuel Clad Gap Width (G2)	200 (1)
2	Clad to Coolant Heat Transfer (G4)	80 (3)
3	Peaking Factor (G1)	50 (4)
4	Break Discharge Coefficient (G5)	90 (2)
5	Fuel Thermal Conductivity (G3)	45 (5)

The ranking by correlation coefficients almost matches the ranking we get by considering Δt- with the exception being that group 5 should be ranked number 2, and then the others would adjust accordingly. This seems to demonstrate that ranking using the correlation coefficients is fairly reasonable. Additionally, it may be worth noting that by the Apendix K ruling of the NRC, all of these correlation coefficients change in temperature would require that these phenomena be ranked 'high.'

7 Final Conclusions and Potential Future Work

Overall, in the AP600 studies, we found that Core Power was the dominating input parameter which most strongly affected our key output, but found that the significance of the variables after that were largely dictated by the break size. For the most part, we found that our statistical rankings were generally the same or less than the PIRT ranking, with the exception of VAR13 in one transient (the 6 inch break.) This makes sense as the PIRT was conservative, i.e. if the experts doing the ranking were unsure as to whether or not a particular phenomenon was medium or high, they tended to rank it high. With regards to the one odd case of the variable 13 ranking medium in the 6 inch break, this indicates that we should further examine this variable and phenomenon. It may be that changing this variable actually relates to another phenomena not covered in the PIRT, or that this particular variable is more complex than we originally thought. One of the limitations of our study is that we cannot be sure how accurately our variables represent the phenomena. We were limited by what quantities it made sense to adjust, as well as the fact that there were cases where we were unable to easily select variables to vary to correspond to phenomenon. In any case, the study does demonstrate great potential for further studies, in particular in applications to other PIRT analyses.

In the LOFT studies, we found that fuel clad gap width was most strongly correlated with the PCT, and interestingly enough, found that the pump degradation seemed to have no relation to Peak Clad Temperature.

Through utilization of PIRTs and statistical methods, we believe that a more accurate uncertainty analysis can be performed.

Further studies may demonstrate that the utilization of statistical methods may improve upon the accuracy of PIRTs produced by a panel of experts. Other future work may utilize the concepts of sampling reduction techniques to vary a greater number of parameters which maintaining a relatively small run-time for studies. Additional improvements to the input file generator may include the addition of the ability for the user to specify a larger number of distributions for the values of various variables.

Aknowledgements

This work is indebted to many. The authors wish to thank and acknowledge the following individuals and organizations for their contributions to this project:

Dr. Jodi Mead, of the Boise State University Department of Mathematics, for her support of this project, and her willingness to serve as the University's advisor of this project.

The International RELAP5 Users Group (IRUG), for funding this work.

The INL's RELAP5-3D team, for their continued support and advice. Many thanks are owed specifically to the RELAP5-3D team's Paul Bayless and Cliff Davis for their assistance and guidance regarding phenomena.

The Idaho National Laboratory and Boise State University's Department of Mathematics, for their support of this work.

References

[1] J.D. Burtt, C.D. Fletcher, G.E. Wilson, C.B. Davis, and T.J. Boucher. "Phenomena Identification and Ranking Tables for Westinghouse AP600 Small Break Loss-of-Coolant Accident, Main Steam Line Break, and Steam Generator Tube Rupture Scenarios." INEL-94/0061. November 1996, Revision 2. 3, 5, 22, 23

[2] Christopher Chatfield, "Statistics for Technology." 3rd Edition. Chapman and Hall: New york, 1985. 9, 11

[3] W.J. Conover, "Practical Non-Parametric Statistics." 3rd Edition. John Wiley & Sons, Inc: New York, 1999. Pages 312-313. 11

[4] J.D. Gibbons and S. Chakraborti, "Nonparametric Statistical Inference." 3rd Edition Revised and Expanded. Marcel Dekker, Inc: New York, 1992. 13

[5] http://notrickszone.com/wp-content/uploads/2011/03/nuclear-power-plant.jpg 22

[6] T. J. Olivier and S. P. Nowlen, "A Phenomena Identification and Ranking Table (PIRT) Exercise for Nuclear Power Plant Fire Modeling Applications." NUREG/CR-6978, 1998. Pages: iii, 1. 7

[7] RELAP5-3D Manual (Volume 2, Appendix A: RELAP5-3D Input Data Requirements, Ver 3.0.0 Beta) 3, 20, 29

[8] Yigal Ronen, "Uncertainty Analysis." CRC Press: Boca Raton, 1988. 6

[9] F. D' Auria, H. Glaeser, S. Lee, J. Mišák, M. Modro, and R. Schultz. "Best Estimate Safety Analysis of Nuclear Power Plants: Uncertainty Evolution." Safety Report Series No 52, 2008. 21

[10] SAS Institute Inc 2010. "Base SAS©9.2 Procedures Guide: Statistical Procedures, Third Edition." Cary, NC: SAS Institute Inc. 15, 17

[11] David J. Sheskin, "Handbook of Parametric and Nonparametric Statistical Procedures." 3rd Edition. CRC Press: Boca Raton, 2000. 9, 12, 13, 14, 15, 16, 19

[12] http://ap1000.westinghousenuclear.com/ap1000_background.html 22

[13] http://upload.wikimedia.org/wikipedia/en/e/ef/AP600PassiveContainment.jpg 22

[14] G. E. Wilson and B. E. Boyack,"The role of the PIRT process in experiments, code development and code applications associated with reactor safety analysis." Nuclear Engineering and Design 186 (1998) 23-37. 6

[15] G.E. Wilson and C.B. Davis, "Best Estimate Versus Conservative Model Calculations of Peak Clad Temperature: An Example Based on the LOFT Facility." Ninth International Topical Meeting on Nuclear Reactor Thermal Hydraulics (NURETH-9). San Francisco, California: October 3-8, 1999. 3, 27, 28

Appendices

A Input Modification Program

```
program input_mod
!
!COGNIZANT: Alexandra E. Gertman
!CREATED:    2/22/2012
!UPDATED:    5/02/2012
!--------------------------------------------------------------
!PROGRAM DESCRIPTION:
!  This program was created to generate many input files with varying
!  parameters (i.e. phenomena or variables).
!
!  The program generates an executable (a.out) upon compilation.
!
!  The executable (a.out) is combined with a marked input file as
!  well as a study specification file (spec file) to create the
!  new input files.
!
!  A spec file includes:
!            -if the spec_file should generate new spec_files OR
!             if it should generate input files
!            -the base name of each generated input file
!             (base name also serves to indicate the name of
!             the marked input file)
!            -the number of variables
!            -the  number of jobs the combinations will be split
!             into
!            -which set of combinations this particular spec_file
!             will generate
!            -the number of groups of variables
!             (there are some cases when variables MUST be changed
!              simultaneously- in those cases, those variables
!              will all be grouped together)
!            -the name of each variable
!            -variable information such as distribution which
!             allow the program to make necessary calculations
!             to calculate the values each variable in the
!             spec file should experience, as well as the group
!             number each variable belongs to.
!
!  The executable will produce a file containing the run information
!  (run_layout) which specifies the number of input files that will
!   be built in each of the 'input file generator' spec files.
!
!  If the spec file is a 'spec file generator' spec file, when
!  compiled with the executable, it will create 'input file
!  generator' spec files (the number of which is specified in the
!  'spec file generator' spec file.
!
!  If the spec file is an 'input file generator' spec file, when
!  compiled with the executable, it will create new input files
!  based upon the marked input file and the specifications in
!  the spec file itself. Once compiled using the executable, it
!  will create a comb file (which it writes to as it creates each
!  of the input files) that contains all of the 'run information'
!  for each of the input files being generated.
!
!
!PROGRAM OUTLINE:
!
!1. Initialize variables, obtain and bypass the name of the executable file,
!   and read the command line option.
!
!2. Open the files.
!   2.1 Open spec_file - a file that has info about the study.
!      2.1.1 Read spec_file and the following information:
!             2.1.1.a. Whether the spec file is an 'input file generator' OR
!                      a 'spec file generator.'
!             2.1.1.b. BASE Name for the input files
!             2.1.1.c. Number of variables, number of 'input file generator'
!                      spec files that will be used in the study (i.e. njobs), which
!                      particular 'input file generator' the spec file is (i.e.
!                      combSet), and the number of groups of variables in the study.
!             2.1.1.d. Name of each variable (as it is marked in the file- except
!                      for the '$'- that is added immediately after the data is
!                      read-in. i.e. 'VAR1' becomes '$VAR1'.)
!             2.1.1.e. Minimum and Maximum values of each variable
!             2.1.1.f. Standard Deviation of each variable
!             2.1.1.g. Number of data points within the range
!             2.1.1.h. Probability function name (uniform, nromal, lognormal, ....).
!                      NOTE: Currently only uniform is available.
!             2.1.1.i. Group number each variable belongs to.
!      2.1.2 Calculate the number of possible combinations, and create an array based on njobs
!            and nposscombs which specifies which runs will take place on each combSet.
!      2.1.3 Create the file run_layout. Write nposscombs on line 1, and then write the
!            number of combinations in each combSet onto line 2. The file is then closed.
!      2.1.4 Based on the first line of the spec_file, one of 3 things will occur:
!             (1) If the spec_file's first line says "generate" send to the subroutine
!                 spec_gen. spec_gen will generate njobs of spec_files which have "run"
!                 instead of "generate" and have different combSet numbers. i.e. only
!                 the first two lines of the newly generated spec_files will differ
!                 from the original spec_file. Once generation  is done, program will
!                 terminate.
!             (2) If first line doesn't say 'run' or 'generate', program will terminate.
!             (3) If the first line is "run", the program will execute normally. i.e.,
!                 Follow the remainder of the outline.
!   2.2 Name of marked_input_file is created.
!   2.3 Open marked_input_file
!
!3. Generate input files
!  3.1 Calculate the variable values for each interval with 'uniform' distribution.
!      3.1.1 Calculate the interval length and increment size
!      3.1.2 Use the probability function, minimum, interval length,
!            and increment size to calculate all the values the variable
!            will have and save these in an array.
!   3.2 Input file generation through loops and subroutines.
!       3.2.1 Make columns an array of ones (of size ngroups.)
!       3.2.2 Open var_val_comb (i.e. comb file)
!   3.3 Input file name generation and input file creation through subroutines.
!       3.3.1 Generate a file names based on combination number or var value.
!               For example: 'edhtrk99' vs 'edhtrk_11_3_3'.
!       3.3.2 Call the subroutine index_gen to generate all the files.
!               -As each file is generated, the run number (i.e. comb_number) and
!                the values for each variable in that particular run/input file will
!                be written to the comb file.
!4. Close all files (with the exception of file 8, fileName, which is closed in the
!   subroutine copy_file or the subroutine spec_gen).
!------------------------------------------------------------
!
!  Data Dictionary
!
!  baseFileName        = a character (of size 10) indicating the name of RELAP5-3D
!                        input file, will serve as beginning letters of all generated
!                        input files. Additionally, it will serve as the beginning of
!                        the name of the file var_val_comb.
!  charCombNum         = a character (of size 8) which is a character conversion of
!                        the combination number of the run so that each line in the
!                        combination file starts with the run number for the
!                        specific combination of variables. It is a local variable
!                        in the recursive subroutine index_gen. It is padded with 0's
!                        to ensure that each run is EXACTLY 8 digits.
!  charCombSet         = a character of size 3 which is the character conversion of
!                        combSet. It is used so that the spec_file can be appended with
!                        the number corresponding to the set of combinations the spec
!                        file needs to generate. It's a local variable used in the
!                        subroutine spec_gen. It's also used in the main program to
!                        make the comb file specific to the combSet.
!  char_i              = a character (of size 8) which is a character conversion of
!                        columns(1:nvar) or (a character conversion of the combination
!                        number of the run) so that newFile can be appended with each
!                        particular variable's corresponding index for that
!                        combination. It is a local variable used in the subroutine
!                        name_gen, and is padded with 0's to ensure it each name is
!                        baseFileName_XXXXXXXX, where XXXXXXXX is EXACTLY 8 digits.
!  charlen             = a local integer in subroutine var_finder which is used to store
!                        the character conversion of the length of a particular var(i).
!                        It is used to re-write line with the variable value substituted
!                        in place of '$VARX'.
!  charvalu            = a local variable (of size 12) in subroutine var_finder which is
!                        the character conversion of valu. It allows valu to be appended
!                        to the file name.
!  columns             = an array of integers (of size nvar- originally set to 55) which
!                        stores the indices of variables in any given combination.
!                        columns(j) corresponds to jth value of variable j.
!  comb_number         = an integer indicating the combination number a particular
!                        generated input file corresponds to.
!  combSet             = an integer of size 3 indicating which particular set of
!                        combinations the given spec_file will need to generate.
!  ex                  = logical flag indicating existance of spec_file and
!                        marked_input_file.
!  exfile              = a character of size 10 indicating the name of the executable
!                        file (usually a.out unless otherwise specified at run time for
!                        input_mod.f90. NOTE: if the executable is changed, the user is
!                        advised to make corresponding changes to the python script
!                        (run_xxxx.py) if tests will be made on the INL cluster.
!  flag                = a character of size 2. It is entered by user after executable file.
!                        If flag is not '-i' program will send error message to user and quit.
!  found               = a logical variable, used in the subroutine copy_file to test whether
!                        or not '$VAR' occured in the line (card) being copied in copy_file.
!  genRun              = character of size '10' which should be either 'run' or 'generate'.
!                        It is found on the first line of the spec_file and if genRun is
!                        'generate' the program will call the subroutine spec_gen. If it's
!                        'run' then the program will run normally.
!  groupNumb           = integer array of size 55 which stores the group number of each var.
!  i                   = integer used in various arguments (i.e. getarg, do loops, etc).
!  ierr                = integer used to detect an input error in reading the spec_file,
!                        marked_input_file, fileName (subroutine copy_file and subroutine
!                        spec_gen), and var_val_comb files.
!  increment_size_var  = array of real numbers indicating the size of increment each
!                        particular variable value will increase by.
!  colindex             = integer indicating the index of a particular column. It is used in
!                        if statements in the subroutine indexgen
!  int_length_var      = array (of size 55)  of real numbers indicating the size of the
!                        range of the values for each particular variable.
!  is                  = integer used to denote the number of command line arguments when
!                        the executable file is run (usually a.out is the executable)
!  j                   = integer used in various arguments (i.e. do loops, etc.)
!  job_inc             = real number which is used to determine the number of jobs that
!                        should be in each combSet. (i.e. job_inc = npossombs/njobs).
!  line                = a character (of size 132) which is used in the subroutine copy_file
!                        to store each string of text from the marked_input_file and then is
!                        used to write the same string into fileName. It is then passed to
!                        the subroutine var_finder which searches line for the variable
!                        markers (ex: $VAR1). If it does contain the specific variable
!                        marker, var_finder will make the variable/value substitution, and
!                        pass line back to the subroutine copy_file. It is also used in the
!                        subroutine spec_gen to read in some of the spec_file lines.
!  marked_input_file   = character of size 25 which corresponds to the name of the input
!                        file which has been "marked" or contains flagsindicating where the
!                        variables are that will be replaced with the values generated by
!                        this program. It is used as a template for generating new input
!                        files.
!  maxpts              = array (of size 55) of integers which indicate the maximum number
!                        of points in any variable's range.
!  mn                  = array (of size 55)  of real numbers indicating the minimum value
!                        for each particular variable.
!  mx                  = array (of size 55) of real numbers indicating the maximum value
!                        for each particular variable. Also used locally in subroutine
!                        spec_gen
!  nameGenType         = a character (of size 4) indicating whether the user would prefer
!                        to generate names based on the total number of combinations
!                        ('comb') or the number of values for each variable ('var').
!                        Ex: 'ed3htrk_99' vs 'ed3htrk_11_3_3'
!  newFile             = a character (of size 80) which corresponds to the name of the new
!                        file which is generated by the subroutine name_gen
!  ngroups             = an integer indicating the number of groups of variables within a
!                        given file. To be more explicit, there are cases in which entire
!                        groups of variables must be changed SIMULTANEOUSLY, and those
!                        variables will be 'grouped' together. e.g. if vars 1-4 must change
!                        simultaneously, var 5 changes independently, vars 6-12 change
!                        simultaneously and var 13 changes independently then there are
!                        4 groups.
!  nposscombs          = integer indicating the total number of combinations that exist for
!                        writing one value for each variable (i.e. this will indicate the
!                        number of input files which will need to be generated.)
!  npts                = array (of size 55) of integers indicating the number of values
!                        each variable will have.
!  nruns               = array (of size 55) of integers which indicate the number of runs
!                        which are in each combSet. (i.e. nruns(2) = the number of runs in
!                        the 2nd combSet. The number of generated files will be
!                        nruns(combSet) - nruns(combSet-1).)
!  nvar                = integer indicating the number of variables which will be replaced
!                        in the marked_input_file. Also used in subroutine spec_gen.
!  originalFile        = a character (of size 10) which is used by the subroutine name_gen
!                        to denote the file which will be copied.
!  prob                = array (of size 55) of characters (of size 8) which indicate the
!                        type of probability distribution that will be used to determine
!                        where the points in the interval for each variable will be.
!  run_layout          = the name of the file containing the run information (i.e the
!                        number of combinations and the number of files created for each
!                        of the combSets. It is a character of size 10.
!  runLayout           = an array of size 55, which holds information about the number of
!                        runs that are in each combSet. It is written to the file run_layout
!                        and then utilized by the python script.
!  sp                  = a character of size 1 (" ") used locally in the subroutine var_finder
!                        to add an extra space in the line (card) after the value of the variable
!                        is inserted AND to add an extra space to the variable name. This prevents
!                        the program from replacing '$VAR11's values with '$VAR1's values, as well
!                        as preventing errors in value replacement- previously, errors occurred
!                        as '$VARX' would replace correctly, but '$VARXY' would add a 'Y' to the
!                        beginning of the value replacement.
!  spec_file           = character of size 20 which corresponds to the name of the file
!                        which contains information about each of the variables which are
!                        going to be replaced in the marked_input_file. This information
!                        includes baseFileName, combSet, genRun, mn, mx, npts, nruns, nvar,
!                        stdev, and var. If genRun = 'generate' spec_file is used as a
!                        template for the generated spec files and serves as a base for the
!                        naming scheme of the newly generated spec files.
!  stdev               = array (of size 55) of real numbers indicating the standard
!                        deviation of each variable.
!  val                 = array of real numbers (of size (20,55)) which corresponds to
!                        val(i,j) = value i of variable j from spec_file formula.
!  valu                = an array of size nvar which holds the specific array of values
!                        for a generated combination of variables and values (i.e.
!                        valu(j) = val(columns(j),j)). It is created in the subroutine
!                        index_gen and then passed to the subroutine copy_file, and then to
!                        the subroutine var_finder so that these values are plugged in for
!                        the specific variable in a particular copyFile.
!  var                 = array (of size 55) of characters (of size 8) denoting each
!                        variable name
!  var_val_comb        = character of size 20 indicating the name of the file created that
!                        contains all the nposscombs of values of the different variables.
!                        In the file, column 1 corresponds to the values of variable 1,
!                        column 2 corresponds to the values of variable 2, etc. Each row
!                        represents a different combination (i.e. nposscombs rows and nvar
!                        columns)
!  where              = an integer corresponding to a local variable in subroutine
!                       var_finder which holds the value returned by the index function
!                       (either a '0' or a '1') which indicates if the variable flag
!                       (ex: $VAR1) is within a particular line.
!------------------------------------------------------------
!
!  files
!
!  file unit number   = file name
!
!  1                  = spec_file (opened and closed in input_mod, utilized in the
!                       subroutine spec_gen if genRun = 'generation'.)
!  2                  = marked_input_file (opened and closed in input_mod, read in
!                       the subroutine copy_file.)
!  3                  = var_val_comb file (opened and closed in input_mod, written
!                       in the recursive subroutine index_gen.)
!  8                  = fileName. This is the file which is a copy all of  information
!                       from the marked_input_file or the information for spec_file if
!                       genRun = 'generate'. (It is opened, written, and closed
!                       in the subroutine copy_file, or in the subroutine spec_gen.)
!  9                  = run_layout. This is the file which contains the information about
!                       the runs it allows the user to double check that the comb files
!                       generated contain the correct number of runs. The first line is
!                       nposscombs. The second line is the array runLayout, with each
!                       integer in the array being size 4. It was originally created to
!                       be used in the python script, but we ended up not using it in
!                       that capacity.
!------------------------------------------------------------
!
!  program/subroutine names and descriptions
!
!  name               = description
!
!  copy_file          = subroutine which is called by index_gen. It is passed (and
!                       returns) fileName, var, nvar, and valu. It is used to generate
!                       a copy of a marked input deck. In order to substitute in correct
!                       variable values, it calls the subroutine var_finder.
!  index_gen          = recursive subroutine called by input_mod. It is passed (and
!                       returns) baseFileName, index, columns, maxcolumn, nvar,
!                       nposscombs, val, var, comb_number, and nameGenType. It
!                       recursively generates all of the possible combinations of
!                       variables and values. It calls the subroutines copy_file (which
!                       calls var_finder) and name_gen to generate all of the
!                       input decks for a particular "input generator" spec file.
!  input_mod          = main program. It opens (and saves information from) spec_file.
!                       It generates the points for a uniform distribution, and it
!                       calls the recursive subroutine index_gen to generate the
!                       files corresponding to the variable/value combinations unless
!                       the first line of the spec file is 'generate' in which case,
!                       the program will call spec_gen to generate all of the 'input
!                       generator' spec files.
!  name_gen           = subroutine called by index_gen. It is passed (and returns)
!                       originalFile, columns, nvar, newFile, comb_number, and
!                       nameGenType. It is used to generate a newFile name which
!                       indicates the particular variable value combination of the
!                       file being created/copied if nameGenType = 'var', otherwise
!                       nameGenType = 'comb' and the generated name  will correspond to
!                       which combination number the file corresponds to.
!  spec_gen            = subroutine which is passed (and returns) spec_file, genRun,
!                        baseFileName, nvar, and combSet. It is called by the main program
!                        if genRun = "generate". It will create njobs specfiles which have
!                        the first line changed to 'run' as opposed to 'generate', so that
!                        when these new spec files are run, they will call index_gen and
!                        are able to generate the new input files. Additionally, it will
!                        generate njobs of them, each of which will have a different
!                        combSet value. (e.g. if njobs = 9, then one file will have combSet=1,
!                        one file will have combSet=2, ..., one file will have combSet=9.)
!  var_finder         = subroutine which is passed (and returns) line, var, nvar and valu.
!                       It is called by copy_file, and uses the above information to find
!                       the VARIABLE MARKERS in the file and substitutes the desired values
!                       into the line and then passes the edited line back to copy_file.
!------------------------------------------------------------
!
!  INFORMATION FOR THE USER:
!  1. THE MARKED INPUT FILE AND SPEC FILE (VARIABLE NAMING):
!     Add a '$' to the beginning of EVERY variable name in your marked input file
!     (i.e. template input file). In the SPEC FILE, place the variable WITHOUT the
!     '$' (i.e. in MARKED INPUT '$VAR1' in SPEC FILE 'VAR1') as a '$' is added in
!     in the main program immediately following the read statement (see sec 2.2.1).
!  2. VARIABLE NAMES:
!     In the program, variable names are set to be of size 8. HOWEVER, due to the
!     addition of '$' to the beginning of the variable name (see #1 directly above)
!     and due to the SPACE added to the end of the variable name (see the subroutine
!     var_finder for a more detailed explanation) variable names MUST be no more
!     than 6 CHARACTERS LONG.
!  3. NUMBER OF VARIABLES OCCURING WITHIN A CARD:
!     Currently ONLY 7 instances of variables may appear in a given line (regardless
!      of which variables- it could be the same variable multiple times plus a few
!      others, or all one variable, or all different variables a single line, i.e.
!      card, may only have 7 substitutions). For more details, see the subroutine
!      var_finder.
!   4. CASE OF A VARIABLE WITH A SINGLE POINT (i.e. stays constant):
!      In the SPEC FILE, simply write the value the user desires to remain constant
!      for that variable in the place where the MIN (i.e. mn) should go.
!   5. MAX NUMBER OF RUNS:
!      Currently ONLY runs of up to 8 digits long are permitted. To change this, and
!      for additional information see NOTE in recursive subroutine index_gen.
!   6. MAXIMUM NUMBER OF VARIABLES SPECIFIED in SPEC FILE:
!      Currently a spec file can ONLY contain information for 55 variables. To
!      change this, the user will need to change the size of several arrays:
!      prob(XX), var(XX), columns(XX), groupNumb(XX), npts(XX), nptsGroup(XX),
!      nruns(XX), runLayout(XX), increment_size_var(XX), int_length_var(XX),
!      mn(XX), mx(XX),stdev(XX), val(20,XX), as well as in a write statement
!      [write (9, '(XXi4)')]. Additionally, changes should be made in the
!      recursive subroutine index gen: val(20,XX), and 62 format(a8,XXes12.4)
!      ALSO should more than 99 variables be in a given file, additional changes will
!      need to be made in  var_finder. See the subroutine for further details.
!-----------------------------------------------------------
!
!   Declarations
!
  implicit none
  character *10 baseFileName, exfile, genRun, run_layout
  character *3 charCombSet
  character *2 flag
  character *25 marked_input_file
  character *4 nameGenType
  character *8  prob(55), var(55)
  character *20 spec_file, var_val_comb
  integer combSet, comb_number, i, ierr, is, j, k, maxpts, nposscombs, ngroups,  njobs, nvar
  integer columns(55), groupNumb(55), npts(55), nptsGroup(55), nruns(55), runLayout(55)
  logical ex
  real increment_size_var(55), int_length_var(55), mn(55), mx(55), stdev(55), val(20,55)
  real job_inc
!-----------------------------------------------------------
!
!   Executable code
!
!   1.0 Initialize
  comb_number = 0
  genRun = 'hello'
  i = 0
  j = 0
  maxpts = 1
  nameGenType = 'type'
  nposscombs = 1
  var_val_comb = 's'
!
!   1.1 Obtain and bypass the name of the executable file
  call getarg (i, exfile)
  is = iargc()
!
!   1.2 Read the command option
  i = 1
  call getarg (i, flag)
  if (flag(1:2) == "-i") then
    i = 2
    call getarg (i, spec_file)
  else
    write (*,*) "Error:_no_-i_was_specified"
    stop "-1"
  endif
!
!   2.0 Open the files
!
!   2.1 Open spec_file
  inquire (file = spec_file, exist = ex)
  if (.not. ex) then
    write (*,*) "Error:_file_not_found,_", spec_file
    go to 100
  endif
  open (unit = 1, file = spec_file, action = "read", status = "old", iostat = ierr)
!
!   2.1.1 Read the spec_file and save the relevant variable information. Then rename
!         each variable to include marker (i.e. '$').
  read (1, "(a10)") genRun
  read (1, "(a10,i3,i3,i3,i3)") baseFileName, nvar, njobs, combSet, ngroups
  do i = 1, nvar
    read (1, "(a8,_3e12.4,_i3,_a8,_i3)", end = 10) var(i), mn(i), mx(i), stdev(i), npts(i),&
prob(i), groupNumb(i)
    var(i) = "$" // trim(var(i))
  end do
10 continue
  rewind 1
```

```
!  2.1.2 Calculate the number of possible combinations, and create an array based on njobs
!        and nposscombs which specifies which runs will take place on each combSet.
!        For example: if there are 27 possible combinations, and njobs = 3, then
!        nruns( ) = [3  6  9], so we know combinations 1-3 will take place if combSet = 1,
!        combinations 4-6 will take place if combset = 2, and combinations 7-9 occur when
!        combSet = 3. Since the ceiling function is used, any number of njobs may be used
!        without concern for the case where  njobs does not evenly divide nposscombs. In
!        that case,  we will just have a different  number, i.e.  +/- 1, of runs generated
!        in one of the 'input deck generator' spec files.)
!
    k = 1
    nptsGroup(1) = npts(1)
    do i = 1, nvar
       if (groupNumb(i) /= groupNumb(k)) then
          k = k + 1
         nptsGroup(k) = npts(i)
       endif
    end do
!   uncomment the line below to write out the group number of each variable
!   write (*,"(20i4)") groupNumb
!
    nposscombs = 1
    do i =1, ngroups
       nposscombs = nptsGroup(i)*nposscombs
    end do
!
   job_inc = (nposscombs)/(njobs)
!
   do i = 1, njobs-1
     nruns(i) = i*ceiling(job_inc)
   end do
   nruns(njobs) = nposscombs
!
!  2.1.3 Open the file "run_layout". It will be used by the python script.
!        For the file, we'll print out the number of possible combinations
!        (i.e. the total number of tests). Then we'll calculate exactly how
!        many runs are expected for each of the nodes (i.e. for each combSet),
!        and then write that onto the second line. (each will be of size 4).
!        Note: nposscombs will be of size 12. The file will then be closed.
!
   run_layout = "run_layout"
   open(unit = 9, file = run_layout, action = "write", status = "replace", iostat = ierr)
   write(9, "(i12)") nposscombs
   runLayout(1) = nruns(1)
   do i = 2, njobs
     runLayout(i) = nruns(i) - nruns(i-1)
   end do
   write(9, '(55i4)') runLayout(1:njobs)
   close(unit = 9)
!
!  2.1.4 Based on genRun, program will follow one of three options.
!        (1) If spec_file is a spec_file with a genRun= "generation", send the spec_file
!            to the subroutine spec_gen which will generate njobs of spec_files. (first if)
!        (2) If spec_file's genRun is not equal to "run" program will terminate
!        (3) Otherwise, execute normally, utilizing the subroutine index_gen.
!
   if (trim(genRun) == 'generate') then
     call spec_gen(spec_file, genRun, baseFileName, nvar, njobs, combSet, ngroups)
     go to 100
   elseif (trim(genRun) /= 'run') then
     go to 100
   endif
!
!  2.2 Use the baseFilename as the marked_input_file
!
   marked_input_file = trim(baseFileName) // '.i'
!
!  2.3 Open the marked_input_file
   inquire (file = marked_input_file, exist = ex)
   if (.not. ex) then
     write (*,*) "Error:_file_not_found,_", marked_input_file
     go to 100
   endif
   open(unit =2, file = marked_input_file, action = "read", status = "old", iostat = ierr)
!
!  3. Generate input files.
!
!  3.1 Calculate the variable values for each interval with 'uniform' dist
!
!  3.1.1 Calculate the interval length and increment size
!
   do i = 1, nvar
     int_length_var(i) = mx(i) - mn(i)
     if (npts(i) <= 1) then
       increment_size_var(i) = 0.0
     else
       increment_size_var(i) = int_length_var(i)/(npts(i)-1)
    endif
  end do
!
!  3.1.2 Calculate each value within the interval for each specific variable.
!        Note: val(rownumber = point in interval, column = variable).
  do i = 1, nvar
    do j = 1, npts(i)
       val(j, i) = mn(i) + (j-1)*(increment_size_var(i))
    end do
!   uncomment the line below to write out all the values for each variable
!     write (*,*) "Values for variable ", trim(var(i)), ": ", val(1:npts(i),i)
  end do
!
  maxpts = 1
  maxpts = maxval(nptsGroup)
!
!  3.2 Input file generation through loops and subroutines.
!
!  3.2.1 Make columns an array of size ngroups with all 1's (ones) inside
  columns(1:ngroups) = 1
!
!  3.2.2 Open var_val_comb, and make the name specific to the combSet by
!        writing a character version of combSet and then appending that
!        to the trimmed version of baseFileName and the string "_comb".
  write(charCombSet,'(i3)') combSet
  charCombSet = trim(charCombSet)
  var_val_comb = trim(baseFileName) // "_comb"// adjustl(charCombSet)
  open(unit = 3, file = var_val_comb, action = "write", status = "replace", iostat = ierr)
!
!  3.3 Input file name generation and file creation through subroutines.
!
!  3.3.1 Generate file names based upon the number of combinations of
!        variables and values or based on the possible values of each
!        variable. nameGenType has two values 'comb' and 'var'.
!        Ex: 'edhtrk_99' for 'comb' vs 'edhtrk_11_3_3' for 'var'.
  nameGenType = 'comb'
!
!  3.3.2 We call the subroutine index_gen. It, in turn, will call the subroutines
!        name_gen and copyfile (copyfile will then call the subroutine var_finder).
!        index_gen will generate all of the possible combinations of input variables
!        and will create all of the input files for the specific 'input file generator'
!        spec file by using name_gen and copyfile/var_finder.
!
  call index_gen(baseFileName, ngroups, columns, nptsGroup, nvar, npposscombs, val, var,&
comb_number, nameGenType, nruns, combSet, groupNumb)
!
!  4. Close the spec_file, marked_input_file, and var_val_comb file
  close (unit = 1, status = "keep")
  close (unit = 2, status = "keep")
  close (unit = 3, status = "keep")
100 stop

contains

recursive subroutine index_gen (baseFileName, colindex, columns, maxcolumn, nvar, nposscombs, val,&
var, comb_number, nameGenType, nruns, combSet, groupNumb)
!
!  AUTHORS: Dr. George L. Mesina and Alexandra E. Gertman
!  CREATED: Mar 02, 2012
!  UPDATED: Apr 18, 2012
!
!  RECURSIVE SUBROUTINE DESCRIPTION:
!  Recursively generate an colindex (columns) which corresponds to the
!  matrix of variable values. ( i.e. val(columns(j), j) corresponds to
!  the column(j) value of variable j.) Then call the subroutines
!  name_gen and copy_file (copy_file will also call var_finder) to
!  generate the corresponding input file names and files.
!
!  CAUTION:
!  Note that combSet == 1 is handled differently than combSet > 1!
!  if changes must be made to the lines written to the combfile,
!  user should ensure that changes are made to the case when
!  combSet == 1 AND combnumber > nruns(combSet-1).
!
!  NOTE:
!  comb_number (specifically the character version of comb_number,
!  charCombNum) is not currently able to handle comb_number > 8 digits.
!  if user wishes to change this, changes need to be made to several
!  places:
!       (i)  index_gen
!                      1. size of character charCombNum in declarations
!                         of local vars from '*8' to '*X'
!                      2. in write statement where comb_number is written to
!                         the charCombNum '(i8.8)' to '(iX.X)' for BOTH elseif
!                         cases.
!                      3. in format 62 and 87 change 'a8' to 'aX'
!       (ii) name_gen
!                       1. size of char_i in declarations to '*X'
!                       2. in the write statement where comb_number is written to
!                          char_i, the format should be changed from '(i8.8)' to
!                         '(iX.X)'.
!
!   Declarations
!
!   Arguments
  implicit none
  character *10 baseFileName
  character *4 nameGenType
  character *80 newFile
  character *8 var(*)
  integer columns(*), groupNumb(nvar), maxcolumn(*), nruns(*)
  integer combSet, comb_number, colindex, nvar, nposscombs
  real val(20, 55)
!
!   Local Variables
  integer i, j
  real valu(nvar)
  character *8 charCombNum
!
! Initialize
  newFile = baseFileName
!
!   Executable Code
!
  if (colindex == 1) then
    do i = 1, maxcolumn(1)
      columns(1) = i
      do j = 1, ngroups
        where (groupNumb == j) valu = val(columns(j),:)
      end do
      comb_number = comb_number + 1
  !     if (comb_number > 6) stop
!  Remove line above to generate more than 2 input files.
      if (comb_number > nruns(combSet)) then
        stop
      elseif (combSet == 1) then
        call name_gen(baseFileName,columns(1:nvar),nvar,newFile, comb_number,nameGenType)
        call copy_file(newFile, var, nvar, valu)
        write (charCombNum, '(i8.8)') comb_number
        write (3,62) charCombNum, valu(1:nvar)
        62 format (a8, 55es12.4)
       ! write (3,621) charCombNum, (columns(j),j=1,ngroups)
       ! uncomment line above (and below) to write the index of each group for each run
       ! 621 format (a8, 55i3)
      elseif (comb_number > nruns(combSet-1)) then
        call name_gen(baseFileName,columns(1:nvar),nvar,newFile, comb_number,nameGenType)
        call copy_file(newFile, var, nvar, valu)
        write (charCombNum, '(i8.8)') comb_number
        write (3,62) charCombNum, valu(1:nvar)
       ! write (3,621) charCombNum, (columns(j),j=1,ngroups)
       ! uncomment line above (as well as the format line) to write the index of each group
       ! for each run
      endif
    end do
  else
    do i = 1, maxcolumn(colindex)
      call index_gen (baseFileName, colindex-1, columns, maxcolumn, nvar, nposscombs, val,&
var,comb_number, nameGenType, nruns, combSet, groupNumb)
      columns(colindex) = columns(colindex) + 1
    end do
    if (columns(colindex) > maxcolumn(colindex)) then
      columns(colindex) = 1
    endif
  endif
  if (comb_number > nposscombs) then
     stop
  end if
return
!
end subroutine index_gen
!-----------------------------------------------------
!
subroutine copy_file(fileName, var,nvar,valu)
!
!  CREATED: Mar 6, 2012
!  UPDATED: Apr 11, 2012
!
!  SUBROUTINE DESCRIPTION:
!  Make (i.e. open and write to) a copy of the marked-up input file
!  with the generated name. Then call var_finder to modify fileName
!  so it's variable values correspond to the file name. Also appends
!  the file name to include '.i'
!
!   Declarations
```

```
!
!  Arguments
!  Arguments
  implicit none
  character *80 fileName
  character *8 var(*)
  integer nvar
  real valu(*)
!
!  Local Variables
  character *132 line
  integer ierr
  logical found
!
!  Executable Code
!
  open (unit = 8, file = fileName, action = "readwrite_", position = "rewind",&
status = "replace", iostat = ierr)
  do
    read (2, "(a132)", end = 999) line
    found = index (line, "$VAR") > 0
    call var_finder(line,var,nvar,valu)
    write (8, "(a)") trim(line)
!
!   UNCOMMENT the text below to print out the line that was altered by
!    the subroutine var_finder and written into the NEW input file.
!
!    if (found) then
!          write (*,*) "Copy File : line = ", line
!    end if
!
  end do

  999 continue

  close (unit = 8)

  rewind 2

  return

end subroutine copy_file
!-----------------------------------------------------
!
subroutine name_gen(originalFile,columns,nvar,newFile, comb_number, nameGenType)
!
!  CREATED: Mar  7, 2012
!  UPDATED: Mar 19, 2012
!
!  SUBROUTINE DESCRIPTION:
!  Creates name for input files based on base name AND variables if
!  nameGenType = 'var', otherwise nameGenType = 'comb' and the name generated will
!  be based on the base name and combination number the var/val corresponds to.
!
!  Declarations
!
!  Arguments
  implicit none
  character *4 nameGenType
  character *80 newFile
  character *10 originalFile
  integer columns(1:nvar)
  integer comb_number, nvar
!
! Local Variables
  character *8 char_i
  integer j
!
! Executable Code
!
  newFile = originalFile

  if (nameGenType == 'comb') then
    write(char_i, '(i8.8)')comb_number
    newFile = trim(newFile) // "_" //trim(adjustl(char_i))
  else
    do j = 1, nvar
      write(char_i, '(i2)') columns(j)
      newFile = trim(newFile) // "_" // trim(adjustl(char_i))
    end do
  endif

  newFile = trim(newFile) // '.i'

  return

end subroutine name_gen
```

```
!------------------------------------------------------------------
!
subroutine var_finder(line,var,nvar, valu)
!
!  CREATED: March 07, 2012
!  UPDATED: April 12, 2012
!
!  SUBROUTINE DESCRIPTION:
!  Find the lines in the input file which contain a variable MARKER
!  (i.e. '$VAR1') and then replace the MARKER with the desired value.
!
!  NOTE:
!  The variable name can have a double digit number, so special care
!  must be taken when forming the new line to prevent part of the
!  name from appearing in the new line. To handle this case, an
!  additional SPACE has been added to the END of the value replacing
!  the marked variable in the line. THIS could potentially cause errors
!  in input processing as it may cause there to be too many characters
!  on a card in the new input file.
!
!  LET THE USER BEWARE:
!  Currently, this subroutine is set to account for up to 7 occurences
!  of a variable within a single card (line) in the input deck.
!  IF more occurences exist in the user's modified deck, CHANGE the
!  outermost do-loop from 1 to 7  TO 1 to X (greatest number of
!  instances of variables occuring on a single card (line) in the marked
!  input deck.
!
!  ADDITIONAL WARNING:
!  ALL VARIABLES ($XXXXXX, where XXXXX is input by the user) are PADDED
!  with a SPACE (i.e. from '$XXXXX' to '$XXXXX ' within this subroutine)
!  so as to ENSURE that the program will not read in variables such as
!  '$VAR1' and '$VAR11' (assuming the user puts them in sequentially in
!  the spec file) and then replace all '$VAR11's values with '$VAR1's
!  values. The user should also be aware that we insert an extra space
!  in the line immediately following the value insertion. If the user
!  should have more than 99 variables, it is recommended that an extra
!  space be inserted for each additional digit of nvar (e.g. for 2000,
!  add two extra spaces, for 999 add one extra space.)
!
!  Declarations
!
!  Arguments
  implicit none
  character *132 line
  character *8 var(*)
  integer nvar
  real valu(*)
!
!  Local Variables
  integer charlen, i, j, loc
  character *12 charvalu
  character *1 sp
!
!  Executable Code
!
  sp = "␣"
  do j = 1, 7
     do i = 1, nvar
        loc = index (line, trim(var(i)) // sp)
        if (loc .ne. 0) then
          charlen = len_trim(var(i))
          write (charvalu,"(es12.4)") valu(i)
!  UNCOMMENT the text below to check which variable and value is being
!  substituted into a particular line. NOTE that it will print out EACH
!  occurence of the variable being replaced.
!          write (*,*) "Var Finder : var(i) = ", var(i), i
!          write (*,*) "Var Finder : line before = ", line
!
          line = line(1:loc-1) // charvalu // sp // line(charlen+loc+1:)
!
!          write (*,*) "Var Finder : line after = ", line
!
        endif
     end do
  end do

  return

end subroutine var_finder
!------------------------------------------------------------------
!
subroutine spec_gen(spec_file, genRun, baseFileName, nvar, njobs, combSet, ngroups)
!
!  CREATED: Mar 15, 2012
!  UPDATED: Apr 18, 2012
!
```

```
!  SUBROUTINE DESCRIPTION:
!  Create njobs specfiles which have the first line changed to 'run'
!  as opposed to 'generate', so that when these new spec files are run,
!  they will call index_gen and actually generate the new input files.
!
!  Declarations
!
!  Arguments
  implicit none
  character *10 baseFileName, genRun
  character *20 spec_file
  integer combSet, ngroups, njobs, nvar
!
!  Local Variables
  character *3 charCombSet
  character *20 fileName
  character *132 line
  integer i, ierr, j
!
!  Executable Code
!
  do i = 1, njobs

    combSet = i
    write(charCombSet,'(i3)') combSet

    fileName = trim(spec_file) // '_' // adjustl(charCombSet)
    fileName = trim(fileName)
    open (unit = 8, file = fileName, action = "readwrite_", position = "rewind",&
status = "replace", iostat = ierr)
!
!  read in the first two lines of spec_file. At this point, we won't be
!  doing anything with them, but after we read them in, we'll be all set
!  to read in the rest of the lines (and print them out) in a loop.
!
    read (1, "(a132)", end = 999) line
    read (1, "(a132)", end = 999) line
!
!  write the first two lines of the new spec_file- replacing "generate" with
!  "run" and changing the combSet number appropriately.
!
    write (8, "(a3)") "run"
    write(8, 23) baseFileName, nvar, njobs, combSet, ngroups
    23 format(a10, i3, i3, i3, i3, i3)
    999 continue
!
!  now we read in the rest of the lines from file 1 and write with no
!  modifications onto the spec_file we're generating.
!
    do j = 1, nvar
      read (1, "(a132)", end = 99) line
      write (8, "(a)") trim(line)
    end do
    99 continue

    rewind 8
    close (unit = 8)
    rewind 1
  end do
   return
end subroutine spec_gen
!-------------------------------------------------
!
end program input_mod
```

B Template Spec File

```
typeOfSpecFile
baseFileName #vars #nodesInStudy currentNode #groups
varX                    min     max    stdDev    #pts       distribution  group#
.                                                                .
.                                                                .
.                                                               .
.                                                               .
varY                       min     max    stdDev    #pts       distribution  group#
```

C Example of a Generator Spec File

```
generate
apsbs      8   9   6   8
VAR1      0.9            1.1          0.1               3   uniform   1
VAR2      0.000135       0.000165     0.000015          3   uniform   2
VAR3      10.63287       12.99573     1.18143           3   uniform   3
VAR4      0.036252       0.044308     0.004028          3   uniform   4
VAR5      0.14688        0.17952      0.01632           3   uniform   5
VAR6      0.078534       0.095986     0.008726          3   uniform   6
VAR7      0.000135       0.000165     0.000015          3   uniform   7
VAR8      .504811        0.616991     0.05609           3   uniform   8
```

D Example of a NodeSpec File

```
run
apsbs      8   9   9   8
VAR1      0.9            1.1          0.0471404520  3   uniform   1
VAR2      0.00135        0.00165      0.0000707106  3   uniform   2
VAR3      8.95959        10.95061     0.4692879144  3   uniform   3
VAR4      0.043749       0.053471     0.0022914973  3   uniform   4
VAR5      0.193752       0.236808     0.0101483965  3   uniform   5
VAR6      0.04419        0.05401      0.0023145962  3   uniform   6
VAR7      0.000135       0.000165     0.0000070710  3   uniform   7
VAR8      .19493874      0.23825846   0.0102105559  3   uniform   8
```

E Python Script for Running Studies on INL's Supercomputer, Quark

```python
#!/apps/local/python/activestate/2.7.1.4/bin/python
#
#  Purpose: Run many input decks from many directories with RELAP5-3D
#  Note:    Pure Python script, no use of Linux scripts or commands.
#           To adjust this run_file for your particular test, go to
#           main program, section 1.0 and adjust the file names (and
#           keyOutputString) accordingly. The user is also encouraged
#           to adjust the walltime request to better fit the user's
#           study.
#  Authors: Dr. George L Mesina, Alexandra E. Gertman
#           J. Shelley
#  Created: Mar 16, 2012
#  Updated: May 02, 2012
#
#  Pro Batch Scheduler (PBS) commands
#PBS -N relap5
#PBS -l select=1:ncpus=12:mpiprocs=12:mem=23gb
#PBS -l place=excl
#PBS -l walltime=10:00:00
#PBS -V
#PBS -j oe
#PBS -q general
#
##############################
#
#  1.0 Imports
import glob, os
from tempfile import mkdtemp
import shutil, subprocess, sys
from string import join
import time
try:
    import forkmap as fm
    FORKMAPavail = "yes"
except Exception, e:
    FORKMAPavail = "no"
##############################

##############################
def CreateDirs(curPath,outPath,filePre, tmpPath):
#  Make the directories where each node's (or combSet's)
#  cases will be run
#
    tmp_dir = mkdtemp(prefix=filePre, dir=tmpPath)
    return tmp_dir
##############################

##############################
def GetInputFiles(inputPre):
#
#  Grab all of files (each of which is named inputPre) from our current path.
#  Then return all of those files.
#  Note: inputPre = specFilePre + '*' for specFiles AND
#        inputPre = inputFilePre%"*"  for inputFiles
#
    inputs = glob.glob(inputPre)
    return inputs
##############################

##############################
def PrepRunGetClean(index, currentPath, relap_exe, tmpPath):
#
#  Runs each input file passed to it (using relap executable rund).
#  Currently, only 12 input files are run at a time (since we're running
#  our tests on quark.) If a different number should be run through each
#  time, appropriate changes should be made in the parallel section, as
#  as in the beginning lines of this program (the PBS -l  select line).
#
    cmd = "%s_%s"%(relap_exe,index)
    output = subprocess.Popen(cmd,shell=True,stdout=subprocess.PIPE).communicate()[0]
    print >> sys.stderr, 'cmd: ', cmd

    return
##############################

##############################
if __name__ == '__main__':
#  DESCRIPTION: Create working directory in temporary-space with subdirs
#               one per cluster node. Run input_mod_gen.f90 to create 1
#               spec-file for each subdirectory by dividing the total no.
#               of combinations equally among the cluster-nodes. Copy the
#                license and template input files, RELAP5-3D, and tpfh2o
#                to each subdirectory. Use input_mod_gen.f90 to generate
#                all input files for the node from template input file.
#                   Find and store key output parameter value from each
#                RELAP5-3D output file along with the sequence number of
#                the file.
#                   When all runs are finished, SAS will be used to
#                analyze the statistics.
#
#  COGNIZANT:    Alexandra E. Gertman, Dr George Mesina, Jon Shelley
#  CREATED: Mar 16, 2012
#  UPDATED: Mar 27, 2012
#
#  PROGRAM OUTLINE:
#   1.0  Initialize and set-up directories
#   1.1 Construct & Populate subdirectories
#       Construct subdirectory structure explained above
#       1.1.1 Change the current working directory to currentPath
#       1.1.2 Collect all of the spec files we'll use in this study.
#       1.1.3 If the directory (tmpPath) does not yet exist, create it.
#             Then create the directory which will hold all of the runs
#             for each combSet (i.e. for each node).
#   1.2 Obtain list of computer nodes
#   1.3 Input file generation
#       1.3.1 Switch the current working directory to tmp_run_dir
#             (i.e. the temporary directory)
#       1.3.2 In each temporary directory, create a link to the needed files:
#             the RELAP5-3D executable (rund), relap5.x, its license and property
#             files, and the appropriate specfile.
#       1.3.4 Collect all of the input files.
#   2.0 Output file generation through parallelization
#   2.1 Run Parallel: All of the input files that were
#       generated will be run with the relap executable.
#       2.1.1 Grab 12 of the input files at a time, passing them
#              to the function PrepRunGetClean where the output
#              for those 12 input files will be created.
# 3.0 Results
# 3.1 Collect the newly generated output files
#      3.1.1 Create a results dictionary where we'll store
#            the run number and the corresponding desired
#            output parameter.
#      3.1.2 Loop through output files to find the key output
#            parameter, and then put that value into the results
#            dictionary along with the corresponding run number
# 3.2 Find the file with the variable information for each
#      run, copy that information and put into a new file,
#      appending the run information with the desired output
#      from the results dictionary. Then save the results file
#      to the current directory.
#      3.2.1 Open and read the comb file for this particular combSet/node
#      3.2.2 Create the list outline.
#            3.2.2.1 Loop through each line of combLine
#                    3.2.2.1.1 Split each line in each of the combFiles
#                    3.2.2.1.2 Append each line with it's corresponding result
#                    3.2.2.1.3 Place modified cline into the list outline
#      3.2.3 Save all of outline to res_text.
#      3.2.4 Open combFile.res, write all of res_text to it, then close the file.
#      3.2.5 Copy the file to the folder 'results' in the home directory
# 4.0 Remove temporary directory and exit program
#
# DATA DICTIONARY:
#   array_idx       = array of cluster nodes. Should correspond to the number of
#                     jobs specified in spec file. (3rd word on line 2 of the
#                     spec file.)
#   cline           = variable denoting each specific line in combLine. It is
#                     stripped and split, appended, and then put back together.
#  cmd              = variable which denotes arguments we'll be passing to the
#                     command line.
#   combFile        = name of the comb file for the particular combSet/node.
#                     It is combFilePre with arry_idx.
#   combFilePre     = base name for each comb file.
#                     For ex: 'apsbs2_comb'. No additional string w/in a string
#                     is specified.
#   combLine        = where each line of the combFile is stored (each line is a
#                     string.)
#   combn           = the run number of cline
#   currentPath     = PBS working directory.
#   data            = where the information for an output file is stored.
#   inputFilePre    = base name used in all of the input files.
#                     For example: 'apsbs_%s.i', where the %s indicates that we
#                     will specify a what string will get placed in between the
#                     underscore and the ".i"
#   index           = variable used to denote 12 different strings (each of which
#                     corresponds to a specific input file.) It is used in the
#                     parallel section, and passed to PrepRunGetClean so that each
#                     of the 12 input files will be run with rund.
#   inputFiles      = all of the generated input files (i.e. spec files which contain
#                       'run' as their first word on line 1) we'll be using in this
#                       study. To get all of these files, we call the function
#                       GetInputFiles, and pass it the string specFilePre + '*'.
#  fixd              = line in data which contains keyOutputString
#  keyOutputParam    = the number in fixd which corresponds to the key output parameter
#                       we are grabbing from the output files.
#  keyOutputString = string which occurs right before the key output parameter
#                       value in the '.p' relap file.
#  markedInputFile = name of the marked input file input_mod_gen.f90 uses as a
#                       template for generating all of the input files.
#  output            = the results generated from running each of the input files with
#                       the relap executable (rund).
#  ouputFilePre      = base name used in all of the output files. It functions in
#                       the same way as the inputFilePre, except it's a '.p' file.
#  outFile           = each individual output file (from outputFiles)
#  outputFiles       = the generated output files are all stored here. They are
#                       outputFilePre%'*'.
#  outline           = list which stores each line of combLine (and appends each of
#                       the lines with the corresponding result from res_dict.)
#  outputPath        = location of output files
#  relap_exe         = name of the relap executable (i.e. how we'll run relap for
#                       each particular input file.)
#  res_dict          = results dictionary. Items in the results dictionary are stored
#                       as run number and then corresponding key output parameter
#                       generated for that particular run.
#                       For ex: ['00000044':'69.5519', '00000045':'69.5519', ... ]
#  resFile           = where the results file is opened, written to (res_text is
#                       written into the file), and closed.
#  resultsFile       = name of the results file (combFile + '.res')
#  res_text          = file where the list outline is written to.
#  runtime           = a call (in parallel section) to the function controller,
#                       passing in all of the input files (12 of them at a time).
#  specFilePre       = base name of the spec file (NOTE: this is not the spec file
#                       generator- rather it is the generated copies which contain
#                      'run' as the first word on line 1 of the spec file.)
#                       Unlike the input/output pre it doesn't include a string
#                       within a string.
#  specFiles         = all of the generated spec files (i.e. spec files which contain
#                       'run' as their first word on line 1) we'll be using in this
#                       study. To get all of these files, we call the function
#                       GetInputFiles, and pass it the string specFilePre + '*'.
#  start2            = floating point representation of time (used to denote when
#                       each set of 12 input files is run with RELAP.) It is used in
#                       the parallel section, in the function controller.
#  tmpPath           = location on temporary disk for the RELAP5-3D code and input
#  tmp_run_dir       = temporary directory where each node's (or combSet's) particular
#                       case will be run. It calls the function CreateDirs() to create
#                       each particular directory by passing currentPath, outputPath,
#                       as well as the specFilePre+str(arry_idx)+'-', and tmpPath)
#
#################################
#
#  1.0  Initialize and set-up directories
   arry_idx = int(os.environ['PBS_ARRAY_INDEX'])
   currentPath = os.environ['PBS_O_WORKDIR']
   tmpPath = "/tmp" + os.sep + "r5stats" + os.sep
   outputPath = currentPath + os.sep + "Output" + os.sep
   inputFilePre = "loftTS_%s.i"
   outputFilePre = "loftTS_%s.p"
   keyOutputString = "pct__________stdfnctn"
   specFilePre = "loftTS_gen_"
   markedInputFile = "loftTS.i"
   relap_exe = "/home/gertae/hpc-runs/rund"
   combFilePre = "loftTS_comb"
#
   print >> sys.stderr, 'arry_idx:_',arry_idx
   print >> sys.stderr, 'currentPath:_',currentPath
   print >> sys.stderr, 'tmpPath:_',tmpPath
   print >> sys.stderr, 'outputPath:_',outputPath
#
#
#  1.1  Construct & Populate subdirectories
#       Construct subdirectory structure explained above
#
#       1.1.1 Change the current working directory to currentPath
   os.chdir(currentPath)
   print "CWD:_",os.getcwd()
   print "ls:_",os.listdir(os.getcwd())
#
#       1.1.2  Collect all of the spec files we'll use in this study.
   specFiles = GetInputFiles(specFilePre+'*')
   print "Spec_Files:_",specFiles
#
#       1.1.3  If the directory (tmpPath) does not yet exist, create it.
#              Then create the directory which will hold all of the runs
#              for each combSet (i.e. for each node).
   if os.path.isdir(tmpPath) == False:
        os.mkdir(tmpPath)
    tmp_run_dir = CreateDirs(currentPath, outputPath, specFilePre+str(arry_idx)+'-',tmpPath)
#
#   1.2 Obtain list of computer nodes
#       Each node has a number of cores
    nodes = open(os.environ['PBS_NODEFILE'],'r').readlines()
    numNodes = len(nodes)
    print "Number of threads: ",numNodes
#
#   1.3 Input file generation
#
#       1.3.1 Switch the current working directory to tmp_run_dir
#             (i.e. the temporary directory)
    os.chdir(tmp_run_dir)
#       1.3.2 In each temporary directory, create a link to the needed files:
#             the RELAP5-3D executable (rund), relap5.x, its license and property
#             files, and the appropriate specfile.
    for lnfile in ['rund','relap5.x','tpfh2o','rellic.bin', specFilePre+'%d'%arry_idx,
                    markedInputFile]:
        cmd = 'ln -s %s/%s'%(currentPath,lnfile)
        print "cmd: %s"%cmd
        output = subprocess.Popen(cmd,shell=True,stdout=subprocess.PIPE).communicate()[0]
#       1.3.3 In each temporary directory, create the command to
#             run the executable from input_mod_gen.f90 (a.out),
#             passing it '-i' (necessary for executable to work),
#             and specFilePre arry_idx.
    cmd = currentPath+os.sep+'a.out -i %s%d'%(specFilePre,arry_idx)
    print "cmd: %s"%cmd
    output = subprocess.Popen(cmd,stdout=subprocess.PIPE,shell=True).communicate()[0]
#       1.3.4 Collect all of the input files.
    inputFiles=GetInputFiles(inputFilePre%"*")
    #inputFiles = inputFiles[:12] #Uncomment for smaller test
    print inputFiles
#
#   2.0  Output file generation through parallelization
#
#
#   2.1  Run Parallel: This is where all of the input files that
#        were generated will be run with the relap executable rund.
#
    ################################
    ## Parallel section
    ################################
#        2.1.1 Grab 12 of the input files at a time, passing them
#              to the function PrepRunGetClean where the output
#              for those 12 input files will be created.
    @fm.parallelizable(12)
    def controller(index):
        print >> sys.stderr, 'index=',index
        start2 = time.time()

        PrepRunGetClean(index, currentPath, relap_exe, tmp_run_dir)

        return time.time() - start2
    runtime = fm.map(controller,[x for x in inputFiles])
    ################################
    ## End Parallel section
    ################################
    #
#
# 3.0 Results
#
# 3.1 Collect the newly generated output files
    outputFiles = glob.glob(outputFilePre%'*')
    print "Output Files: ",outputFiles
#       3.1.1 Create a results dictionary where we'll store
#             the run number and the corresponding desired
#             output parameter.
    res_dict = dict()
#
#       3.1.2 Loop through output files to find the key output
#             parameter, and then put that value into the results
#             dictionary along with the corresponding run number
    for outFile in outputFiles:
        data=open(outFile).read()
        fidx = data.rfind(keyOutputString)
        keyOutputParam = (data[fidx:fidx+80]).split()[2]
        #print "%s Coremin: %s"%(outFile,pct)
        res_dict[outFile.split('_')[1].split('.')[0]]= keyOutputParam
    print "Results Dictionary:\n",res_dict
#
# 3.2 Find the file with the variable information for each
#       run, copy that information and put into a new file,
#       appending the run information with the desired output
#       from the results dictionary. Then save the results file
#       to the current directory.
#
```

```
#       3.2.1 Open and read the comb file for this particular combSet/node
    print "Have not yet found Comb File"
    combFile=combFilePre+str(arry_idx)
    print "Found Comb File"
    combLine=open(combFile).readlines()
#
#       3.2.2 Create the list outline.
    outline = list()
#
#               3.2.2.1 Loop through each line of combLine
    for cline in combLine:
#
#                       3.2.2.1.1 Split each line in each of the combFiles
        cline = cline.strip()
        cline = cline.split()
#
#                       3.2.2.1.2 Append each line with it's corresponding result
        try:
           combn = cline[0]
           cline.append(res_dict[combn])
        except:
           cline.append('noResults')
#
#                        3.2.2.1.3 Place modified cline [joined w/commas and w/ a
#                                  '\n' at the end of the line to indicate a new
#                               line for each result] into the list outline
        outline.append(",".join(cline) + "\n")
#
#      3.2.3 Save all of outline to res_text.
    res_text = join(outline,' ')
#
#      3.2.4 Open combFile.res, write all of res_text to it, then close the file.
    resultsFile=combFile+'.res'
    print "Results File(s): ", resultsFile
    resFile = open(resultsFile,'w')
    resFile.write(res_text)
    resFile.close()
#
#      3.2.5 Copy the file to the folder 'results' in the home directory
    shutil.copy(resultsFile,currentPath)
#
#   4.0  Remove temporary directory and exit program
    try:
        print "Remember to clean up after the run"
        shutil.rmtree(tmpPath)
    except Exception, e:
        print >> sys.stderr, e
        print >> sys.stderr, "Error occurred while removing temporary directories"
    #
        print >> sys.stderr, 'Node %i runtimes (sec):\n  Max %f,\n  Min %f,\n  Avg %f,\n Tot %f '%\
           (arry_idx,max(runtime),min(runtime),sum(runtime)/len(runtime),sum(runtime))
    print >> sys.stderr, "For ",len(runtime)," runs"
    RunTime = time.time() - Runstart2
    print >> sys.stderr, "Total run time =",RunTime
```

F SAS Reports

F.1 AP600 2 inch Break, Top 4 Variables

Thursday, April 12, 2012 4:18 PM 1

Correlation Analysis

The CORR Procedure

1 With Variables:	coremin
4 Variables:	VAR1 VAR8 VAR12 VAR13

SSCP Matrix				
	VAR1	**VAR8**	**VAR12**	**VAR13**
coremin	6607.808270	3294.217900	6618.107340	1.662016

CSSCP Matrix				
	VAR1	**VAR8**	**VAR12**	**VAR13**
coremin	-5.60623000	-12.48935000	4.69284000	-0.00787117

Covariance Matrix, DF = 80				
	VAR1	**VAR8**	**VAR12**	**VAR13**
coremin	-.0700778750	-.1561168750	0.0586605000	-.0000983896

Simple Statistics						
Variable	**N**	**Mean**	**Std Dev**	**Median**	**Minimum**	**Maximum**
coremin	81	81.64709	1.88027	81.67210	78.39860	84.71310
VAR1	81	1.00000	0.04108	1.00000	0.95000	1.05000
VAR8	81	0.50000	0.41079	0.50000	0	1.00000
VAR12	81	1.00000	0.24648	1.00000	0.70000	1.30000
VAR13	81	0.0002525	0.0002033	0.0002525	5E-6	0.0005000

Pearson Correlation Coefficients, N = 81 Prob > |r| under H0: Rho=0				
	VAR1	**VAR8**	**VAR12**	**VAR13**
	-0.90727	-0.20212	0.12658	-0.25734
coremin	<.0001	0.0704	0.2602	0.0204

Spearman Correlation Coefficients, N = 81 Prob > |r| under H0: Rho=0				
	VAR1	**VAR8**	**VAR12**	**VAR13**
	-0.90408	-0.21729	0.13451	-0.25932
coremin	<.0001	0.0513	0.2312	0.0194

Kendall Tau b Correlation Coefficients, N = 81 Prob > |tau| under H0: Tau=0				
	VAR1	**VAR8**	**VAR12**	**VAR13**
	-0.77650	-0.16867	0.10406	-0.20474
coremin	<.0001	0.0516	0.2298	0.0182

Hoeffding Dependence Coefficients, N = 81 Prob > D under H0: D=0				
	VAR1	**VAR8**	**VAR12**	**VAR13**
	0.31322	0.00025	-0.00573	0.00603
coremin	<.0001	0.3472	0.9429	0.1216

Generated by the SAS System ('SASApp', X64_ESRV08) on April 12, 2012 at 4:18:04 PM

F.2 AP600: 4 inch Break, Top 5 Variables

Thursday, April 12, 2012 4:42 PM 1

Correlation Analysis

The CORR Procedure

1 With Variables:	coremin
5 Variables:	VAR1 VAR8 VAR11 VAR12 VAR13

SSCP Matrix

	VAR1	VAR8	VAR11	VAR12	VAR13
coremin	16908.89162	8454.59730	16913.76764	16914.85865	4.26840

CSSCP Matrix

	VAR1	VAR8	VAR11	VAR12	VAR13
coremin	-5.867280000	-2.782150000	-0.991260000	0.099750000	-0.002573307

Covariance Matrix, DF = 242

	VAR1	VAR8	VAR11	VAR12	VAR13
coremin	-.0242449587	-.0114964876	-.0040961157	0.0004121901	-.0000106335

Simple Statistics

Variable	N	Mean	Std Dev	Median	Minimum	Maximum
coremin	243	69.60806	0.76876	69.57290	68.11380	71.52660
VAR1	243	1.00000	0.04091	1.00000	0.95000	1.05000
VAR8	243	0.50000	0.40909	0.50000	0	1.00000
VAR11	243	1.00000	0.24545	1.00000	0.70000	1.30000
VAR12	243	1.00000	0.24545	1.00000	0.70000	1.30000
VAR13	243	0.0002525	0.0002025	0.0002525	5E-6	0.0005000

Pearson Correlation Coefficients, N = 243
Prob > |r| under H0: Rho=0

	VAR1	VAR8	VAR11	VAR12	VAR13
	-0.77092	-0.03656	-0.02171	0.00218	-0.06831
coremin	<.0001	0.5706	0.7364	0.9730	0.2889

Spearman Correlation Coefficients, N = 243
Prob > |r| under H0: Rho=0

	VAR1	VAR8	VAR11	VAR12	VAR13
	-0.76563	-0.03280	-0.01617	-0.00007	-0.05008
coremin	<.0001	0.6109	0.8020	0.9991	0.4371

Kendall Tau b Correlation Coefficients, N = 243
Prob > |tau| under H0: Tau=0

	VAR1	VAR8	VAR11	VAR12	VAR13
	-0.61577	-0.02295	-0.01218	0.00229	-0.03845
coremin	<.0001	0.6436	0.8060	0.9632	0.4382

Hoeffding Dependence Coefficients, N = 243
Prob > D under H0: D=0

	VAR1	VAR8	VAR11	VAR12	VAR13
	0.18111	-0.00235	-0.00234	-0.00240	0.00021
coremin	<.0001	0.9977	0.9974	0.9987	0.3226

Generated by the SAS System ('SASApp', X64_ESRV08) on April 12, 2012 at 4:42:07 PM

F.3 AP600: 6 inch Break, Top 5 Variables

Thursday, April 12, 2012 5:32 PM 1

Correlation Analysis

The CORR Procedure

1 With Variables:	coremin
5 Variables:	VAR1 VAR7 VAR8 VAR12 VAR13

SSCP Matrix

	VAR1	VAR7	VAR8	VAR12	VAR13
coremin	12415.99854	3145.03885	6257.13615	12434.50585	3.12222

CSSCP Matrix

	VAR1	VAR7	VAR8	VAR12	VAR13
coremin	-5.17726000	-0.00286622	46.54825000	13.33005000	-0.01413007

Covariance Matrix, DF = 242

	VAR1	VAR7	VAR8	VAR12	VAR13
coremin	-.0213936364	-.0000118439	0.1923481405	0.0550828512	-.0000583887

Simple Statistics

Variable	N	Mean	Std Dev	Median	Minimum	Maximum
coremin	243	51.11595	0.97550	51.09220	48.97220	53.79780
VAR1	243	1.00000	0.04091	1.00000	0.95000	1.05000
VAR7	243	0.25320	0.02072	0.25320	0.22788	0.27852
VAR8	243	0.50000	0.40909	0.50000	0	1.00000
VAR12	243	1.00000	0.24545	1.00000	0.70000	1.30000
VAR13	243	0.0002525	0.0002025	0.0002525	5E-6	0.0005000

Pearson Correlation Coefficients, N = 243
Prob > |r| under H0: Rho=0

	VAR1	VAR7	VAR8	VAR12	VAR13
	-0.53609	-0.00059	0.48199	0.23005	-0.29558
coremin	<.0001	0.9927	<.0001	0.0003	<.0001

Spearman Correlation Coefficients, N = 243
Prob > |r| under H0: Rho=0

	VAR1	VAR7	VAR8	VAR12	VAR13
	-0.52077	-0.00833	0.45018	0.24533	-0.30260
coremin	<.0001	0.8972	<.0001	0.0001	<.0001

Kendall Tau b Correlation Coefficients, N = 243
Prob > |tau| under H0: Tau=0

	VAR1	VAR7	VAR8	VAR12	VAR13
	-0.41714	-0.00594	0.35799	0.19097	-0.23436
coremin	<.0001	0.9046	<.0001	0.0001	<.0001

Hoeffding Dependence Coefficients, N = 243
Prob > D under H0: D=0

	VAR1	VAR7	VAR8	VAR12	VAR13
	0.07638	-0.00338	0.05917	0.01273	0.01938
coremin	<.0001	1.0000	<.0001	0.0010	<.0001

Generated by the SAS System ('SASApp', X64_ESRV08) on April 12, 2012 at 5:32:43 PM

F.4 AP600: 8 inch Break, Top 4 Variables

Correlation Analysis

The CORR Procedure

1 With Variables:	coremin
4 Variables:	VAR1 VAR8 VAR12 VAR13

SSCP Matrix

	VAR1	VAR8	VAR12	VAR13
coremin	2739.119790	1380.873250	2918.222760	0.615148

CSSCP Matrix

	VAR1	VAR8	VAR12	VAR13
coremin	1.64289000	12.13480000	-1.75260000	-0.00078453

Covariance Matrix, DF = 80

	VAR1	VAR8	VAR12	VAR13
coremin	0.0205361250	0.1516850000	-.0219075000	-.0000098066

Simple Statistics

Variable	N	Mean	Std Dev	Median	Minimum	Maximum
coremin	81	33.79601	1.29386	33.96140	28.58110	37.45090
VAR1	81	1.00000	0.04108	1.00000	0.95000	1.05000
VAR8	81	0.50000	0.41079	0.50000	0	1.00000
VAR12	81	1.06667	0.23717	1.00000	0.70000	1.30000
VAR13	81	0.0002250	0.0001836	0.0002525	5E-6	0.0005000

Pearson Correlation Coefficients, N = 81
Prob > |r| under H0: Rho=0

	VAR1	VAR8	VAR12	VAR13
	0.38637	0.28539	-0.07139	-0.04129
coremin	0.0004	0.0098	0.5265	0.7144

Spearman Correlation Coefficients, N = 81
Prob > |r| under H0: Rho=0

	VAR1	VAR8	VAR12	VAR13
	0.32918	0.46694	-0.23755	-0.18540
coremin	0.0027	<.0001	0.0327	0.0975

Kendall Tau b Correlation Coefficients, N = 81
Prob > |tau| under H0: Tau=0

	VAR1	VAR8	VAR12	VAR13
	0.25619	0.31788	-0.19474	-0.14414
coremin	0.0032	0.0002	0.0255	0.0983

Hoeffding Dependence Coefficients, N = 81
Prob > D under H0: D=0

	VAR1	VAR8	VAR12	VAR13
	0.01622	0.04746	0.01046	0.00160
coremin	0.0242	0.0003	0.0588	0.2683

Generated by the SAS System ('SASApp', X64_ESRV08) on April 13, 2012 at 4:41:36 PM

F.5 LOFT: 3 Values per Variable

Tuesday, April 24, 2012 10:26 AM 1

Correlation Analysis

The CORR Procedure

1 With Variables:	PCT
6 Variables:	VAR1 VAR25 VAR26 VAR44 VAR45 VAR47

SSCP Matrix

	VAR1	VAR25	VAR26	VAR44	VAR45	VAR47
PCT	35809.288	3424.870	6205105.659	808202.822	681860.601	73642.606

CSSCP Matrix

	VAR1	VAR25	VAR26	VAR44	VAR45	VAR47
PCT	-34.88999	-3.78793	-10562.55619	-20307.45525	-3041.22853	-2.75210

Covariance Matrix, DF = 728

	VAR1	VAR25	VAR26	VAR44	VAR45	VAR47
PCT	-0.04792581	-0.00520320	-14.50900575	-27.89485611	-4.17751171	-0.00378036

Simple Statistics

Variable	N	Mean	Std Dev	Median	Minimum	Maximum
PCT	729	1010	192.24533	975.73000	723.60100	1475
VAR1	729	0.04867	0.0005781	0.04867	0.04796	0.04938
VAR25	729	0.00466	0.0000624	0.00466	0.00458	0.00473
VAR26	729	8.44000	0.68960	8.44000	7.59600	9.28400
VAR44	729	1.12500	0.30640	1.12500	0.75000	1.50000
VAR45	729	0.93000	0.15197	0.93000	0.74400	1.11600
VAR47	729	0.10000	0.04085	0.10000	0.05000	0.15000

Pearson Correlation Coefficients, N = 729
Prob > |r| under H0: Rho=0

	VAR1	VAR25	VAR26	VAR44	VAR45	VAR47
	-0.43126	-0.43386	-0.10944	-0.47357	-0.14299	-0.00048
PCT	<.0001	<.0001	0.0031	<.0001	0.0001	0.9896

Spearman Correlation Coefficients, N = 729
Prob > |r| under H0: Rho=0

	VAR1	VAR25	VAR26	VAR44	VAR45	VAR47
	-0.33134	-0.70081	-0.16088	-0.39616	-0.21612	0.00031
PCT	<.0001	<.0001	<.0001	<.0001	<.0001	0.9934

Kendall Tau b Correlation Coefficients, N = 729
Prob > |tau| under H0: Tau=0

	VAR1	VAR25	VAR26	VAR44	VAR45	VAR47
	-0.25922	-0.59409	-0.12552	-0.30960	-0.16669	0.00034
PCT	<.0001	<.0001	<.0001	<.0001	<.0001	0.9906

Hoeffding Dependence Coefficients, N = 729
Prob > D under H0: D=0

	VAR1	VAR25	VAR26	VAR44	VAR45	VAR47
	0.02860	0.20104	0.00631	0.04027	0.01090	-0.00119
PCT	<.0001	<.0001	<.0001	<.0001	<.0001	1.0000

Generated by the SAS System ('SASApp', X64_ESRV08) on April 24, 2012 at 10:26:38 AM

F.6 LOFT: 6 Values per Variable

Monday, April 30, 2012 11:57 AM 1

Correlation Analysis

The CORR Procedure

1 With Variables:	PCT
6 Variables:	VAR1 VAR25 VAR26 VAR44 VAR45 VAR47

SSCP Matrix

	VAR1	VAR25	VAR26	VAR44	VAR45	VAR47
PCT	741345.7	70880.1	128445869.5	16928250.1	14118361.3	1523872.0

CSSCP Matrix

	VAR1	VAR25	VAR26	VAR44	VAR45	VAR47
PCT	-354.6168	-66.9270	-170739.5504	-215551.4121	-53847.9969	-21.4771

Covariance Matrix, DF = 15624

	VAR1	VAR25	VAR26	VAR44	VAR45	VAR47
PCT	-0.02269692	-0.00428360	-10.92803062	-13.79617333	-3.44649237	-0.00137462

Simple Statistics

Variable	N	Mean	Std Dev	Median	Minimum	Maximum
PCT	15625	975.29182	138.06833	960.68800	723.60100	1476
VAR1	15625	0.04867	0.0005002	0.04867	0.04796	0.04938
VAR25	15625	0.00466	0.0000540	0.00466	0.00458	0.00473
VAR26	15625	8.44000	0.59682	8.44000	7.59600	9.28400
VAR44	15625	1.12500	0.26517	1.12500	0.75000	1.50000
VAR45	15625	0.93000	0.13153	0.93000	0.74400	1.11600
VAR47	15625	0.10000	0.03536	0.10000	0.05000	0.15000

Pearson Correlation Coefficients, N = 15625
Prob > |r| under H0: Rho=0

	VAR1	VAR25	VAR26	VAR44	VAR45	VAR47
PCT	-0.32863	-0.57473	-0.13262	-0.37682	-0.18979	-0.00028
	<.0001	<.0001	<.0001	<.0001	<.0001	0.9719

Spearman Correlation Coefficients, N = 15625
Prob > |r| under H0: Rho=0

	VAR1	VAR25	VAR26	VAR44	VAR45	VAR47
PCT	-0.25152	-0.80646	-0.17079	-0.31533	-0.25236	-0.00048
	<.0001	<.0001	<.0001	<.0001	<.0001	0.9517

Kendall Tau b Correlation Coefficients, N = 15625
Prob > |tau| under H0: Tau=0

	VAR1	VAR25	VAR26	VAR44	VAR45	VAR47
PCT	-0.18691	-0.66938	-0.12528	-0.23445	-0.18548	-0.00035
	<.0001	<.0001	<.0001	<.0001	<.0001	0.9523

Generated by the SAS System ('SASApp', X64_ESRV08) on April 30, 2012 at 11:57:40 AM