AN ABSTRACT OF THE DISSERTATION OF

Kevin A. Makinson for the degree of Doctor of Philosophy in Radiation Health Physics

presented on April 19, 2013.

Title: Preliminary Framework for the Run-Ahead Predictive Simulation Software (RAPSS)

Abstract approved:

Andrew C. Klein

The Run-Ahead Predictive Simulation Software (RAPSS) is an architecture designed for faster-than-real-time decision support for operators of complex networks. To enable further development of the RAPSS methodology, the necessary proof of principle is illustrated in two applications: decision support for shift technical advisors in nuclear power plant control rooms (RAPSS-STA), and in the event of a release outside of containment, decision support for emergency operation centers (RAPSS-EOC).

Preliminary Framework for the Run-Ahead Predictive Simulation Software (RAPSS)

by
Kevin A. Makinson

A DISSERTATION

submitted to

Oregon State University

in partial fulfillment of
the requirements for the
degree of

Doctor of Philosophy

Presented April 19, 2013
Commencement June 2013

Doctor of Philosophy dissertation of Kevin A. Makinson presented on April 19, 2013.

APPROVED:

____

Major Professor, representing Radiation Health Physics

____

Head of the Department of Nuclear Engineering and Radiation Health Physics

____

Dean of the Graduate School

____

Kevin A. Makinson, Author

ACKNOWLEDGEMENTS

I would first like to express my sincerest appreciation to Dr. Andrew Klein, my advisor, for his guidance and support throughout this process. To my other committee members, Drs. Kathryn Higley, Qiao Wu, Henri Jansen, and Ken Krane, thank you for all you've done and support throughout the process. My parents, Clyde and Gretchen Makinson, thank you for your unconditional love and for supporting my participation in recreational extracurricular activities (e.g., ski coaching). Thank you Robyn Mills for the love, patience, and tolerance while I finished my degree. Thank you Diego Mandelli for many thought provoking discussions and handfuls of dimensionality reduction and clustering emails. Bob Youngblood also deserves thanks for suggesting the connection with weather prediction, and for mentoring me during my time at Idaho National Lab. Thanks also to Curtis Smith, Nam Dinh, and many other important researchers at INL for their guidance as well as financial support for this project. Thanks to Tom Riley for building the RELAP5 models and being my go-to guy for RELAP questions. I'd particularly like to give a shout out to my undergraduate computer science, math, and physics professors: Drs. Lynda Danielson, Robin Cruz, and James Dull for putting up with me for four years and teaching me the foundations that allowed me to be successful now. Thanks to Bert Martin for his work on the LiteFTA module, and for helping me through computer science labs as an undergrad. Finally, Chris Thompson's guidance has been incalculably beneficial in the last few years. This department would come to a grinding halt if not for Chris's computer expertise. Thanks also to everyone else that I've had a pleasure getting to know in the last six years. Peace and love!

TABLE OF CONTENTS Page

TABLE OF CONTENTS (Continued) Page

TABLE OF CONTENTS (Continued) Page

TABLE OF CONTENTS (Continued) Page

TABLE OF CONTENTS (Continued) Page

TABLE OF CONTENTS (Continued) Page

Figure LIST OF FIGURES Page

Figure LIST OF FIGURES (Continued) Page

LIST OF TABLES

LIST OF ACRONYMS

Acronym	Definition
ADAPT	Analysis of Dynamic Accident Progression Trees
ADS	Accident Dynamic Simulation
AEC	Atomic Energy Commission
ALAIS	Automated Linear Approximation Interval Sequencer
API	Application Programming Interface
APS	American Physical Society
ATWS	Anticipated Transients Without Scram
BDBA	Beyond Design Basis Accident
BWR	Boiling Water Reactor
CAMP	Code Applications and Maintenance Program
CATHARE	Code for Analysis of THermalhydraulics during an Accident of Reactor and safety Evaluation
CDF	Cumulative distribution function
CDF	Core Damage Frequency
CFR	Code of Federal Regulation
CPU	Central Processing Unit
DET	Dynamic Event Tree
DOE	Department of Energy
DYLAM	DYnamic Logical Analytical Methodology
EF	Ensemble Forecasting
EnsKF	Ensemble Kalman Filter

LIST OF ACRONYMS (Continued)

Acronym	Definition
FCM	Fuzzy C-Means
FormalFTA	Formal Fault Tree Analysis
FR	Federal Register
FRAMES	Framework for Risk Analysis in Multimedia Environmental Systems
F-V	Fussell-Vesely
GIP	Generic Issues Program
GNU	General Public License; synonymous with GPL
GPL	General Public License synonymous with GNU
GPM	Gaussian Plume Model
GUI	Graphical User Interface
ICAP	International Code Assessment and Applications Program
IDAC	Information Decision and Action in Crew
INPO	Institute of Nuclear Power Operations
IPE	Individual Plant Examination
KF	Kalman Filter
LB	Licensing Basis
LB LOCA	Large-Break Loss of Coolant Accident
LERF	Large Early Release Frequency
LiteFTA	Lite Fault Tree Analysis
LLEnsF	Local-Local Ensemble Filter
LWR	Light Water Reactors

LIST OF ACRONYMS (Continued)

Acronym	Definition
MASLWR	Multi Application Small Light Water Reactor
MAUT	Multi-Attribute Utility Theory
MCDET	Monte Carlo Dynamic Event Tree
MELCOR	Methods for Estimation of Leakages and Consequences of Releases
MPI	Message Passing Interface
NPP	Nuclear Power Plant
NS-EnsKF	Normal-Score Ensemble Kalman Filter
NUREG	NUclear REGulation
NWP	Numerical Weather Prediction
OpenFTA	Open Fault Tree Analysis
OpenMP	Open Multi-processing
PDF	Plant Damage Frequency
PDS	Plant Damage States
PRA	Probabilistic Risk Assessment
PVM	Parallel Virtual Machine
PWR	Pressurized Water Reactor
R7	RELAP 7
RAP	Reliability/Availability/Performance
RAPSS	Run-Ahead Predictive Simulation Software
RAPSS-EOC	Run-Ahead Predictive Simulation Software for Emergency Operations Centers

LIST OF ACRONYMS (Continued)

Acronym	Definition
RAPSS-STA	Run-Ahead Predictive Simulation Software for Shift Technical Advisors
RELAP	Reactor Excursion and Leak Analysis Program
RHRS	Residual Heat Removal System
RISMC	Risk Informed Safety Margin Characterization
RPV	Reactor pressure vessel
SB LOCA	Small-Break Loss of Coolant Accident
SDP	Significance Determination Process
SGTR	Steam Generator Tube Rupture

Preliminary Framework for the Run-Ahead Predictive Simulation Software (RAPSS)

1 Introduction

The Fukishima Diiachi accident in April 2011 was the most recent reminder of what catastrophic failure of nuclear power generating stations looks like. In the aftermath of the disaster, the American Nuclear Society (ANS) published a report outlining their recommendations for upgrades to the current generation of nuclear power plants to decrease the probability of another Fukishima. Recommendation V.D. for accident diagnostics tools from the ANS Committee Report (March 2012) recommends that plants:

> "Provide operators with information regarding the accident progression which can then allow them to identify the most effective strategy to manage a prolonged [station black out] or [beyond design basis accident] sequence. This information might be provided in the form of pre-prepared charts or generated for the actual conditions of the NPP by a *faster-than-real-time simulator* that can predict the gross behavior of the essential NPP subsystems under beyond-design basis conditions, especially before substantial core damage occurs, so that core damage can actually be prevented."

The Run-Ahead Predictive Simulation Software (RAPSS) is an architecture designed specifically for this purpose: faster-than-real-time decision support for operators of complex networks.

1.1 RAPSS-STA

The first and most developed application of the RAPSS methodology was designed to assist the senior members of a nuclear power plant (NPP) operating crew (i.e., the plant's Shift or Unit Supervisor and the Shift Technical Advisor (STA)) in the assessment of current and potential future reactor system conditions. This application has thus been named RAPSS-STA. This tool generates a set of scenarios to predict what could happen in the near future (with associated probabilities) by continuously performing a faster-than-real-time probabilistic risk assessment including outcome and consequence analyses.

When fully implemented and connected to a reactor system, RAPSS-STA will utilize current plant data to generate a set of inputs for an advanced systems modeling code, such as: TRACE, RELAP5, MELCOR, or CATHARE (see Section 2.3). In contrast to the slower-running comprehensive dynamic probabilistic risk analysis codes, these parallel “potential futures” calculations are determined utilizing a small number of streamlined, risk-informed algorithms that repeatedly initiate, identify, analyze, and disposition possible near future scenarios in a probabilistic manner as plant conditions evolve. Results are presented to senior operating staff (i.e., Unit Supervisor and STA), who can make use of these risk-informed projections to help guide plant operations decisions. Such an approach provides a degree of safety margin monitoring by presenting the unit leadership with a real-time predictive analysis of future plant conditions. An outline of both the existing NPP structure and the new RAPSS-STA methodology is presented in Figure 1.1.

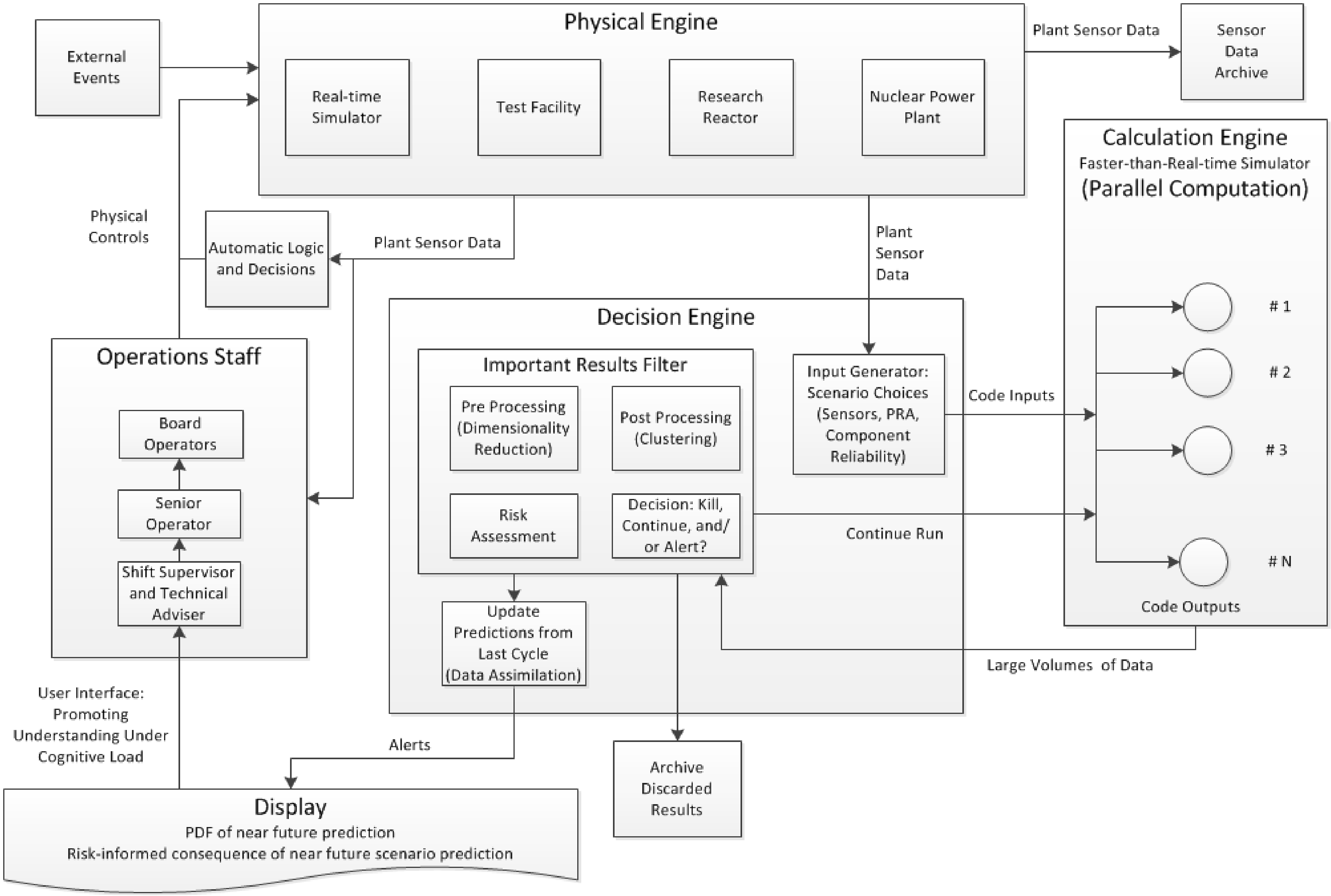


Figure 1.1 Conceptual outline of RAPSS-STA implementation in a nuclear power plant

The diagram in Figure 1.1 starts with a "physical engine", which can be the reactor itself, a test facility, a research reactor, or a reactor simulator. While the physical engine can be represented in a number of ways, it is important that the faster-than-real-time "calculation engine" provide a reasonable fidelity physical engine simulation, and very quickly. The "physical engine" will be influenced both by external events, such as off-site power failures, earthquakes, tornados, and tsunamis, or by internal conditions and the physical controls, such as opening valves or starting pumps. The physical engine will continuously feed data to its physical sensors (e.g., temperature, pressure gauges, etc.), which are archived, displayed, and used to drive automatic logic and actions. Those automatic decisions activate the physical controls, which, in turn, change plant configuration and conditions. The physical displays and alerts are read by plant operators, the unit supervisor, and the STAs who can influence the operators, but the operators otherwise follow procedures, which utilize the physical controls to change conditions in the plant.

The RAPSS methodology incorporates a “decision engine,” which digests current plant data into a suitable input format for the “calculation engine,” which performs simultaneous outcome assessments across several parallel computing nodes. The calculation engine is a systems modeling code capable of faster-than-real-time performance. The decision engine first decides which future plant scenarios to run using combinations of sensor input data, the plant’s own probabilistic risk assessment (PRA), component availability and reliability data, and other useful inputs. The calculation engine continuously runs an ensemble of parallel calculations with appropriately perturbed initial conditions, projecting a short time in the future based on current plant conditions. The decision engine takes these outputs, compiles and organizes the large volume of output data from the calculation engine utilizing dimensionality reduction techniques among others, and decides which of these scenarios is different enough from current conditions or might lead to a sufficiently consequential, negative outcome to warrant alerting the STA or unit supervisor. The scenarios are flagged, organized by risk, and displayed to the shift supervisors and STA in a way that promotes understanding under high cognitive load.

1.1.1 RAPSS-STA Simulation Timeline

Figure 1.2 is useful to conceptually visualize the series of cycles RAPSS-STA performs.

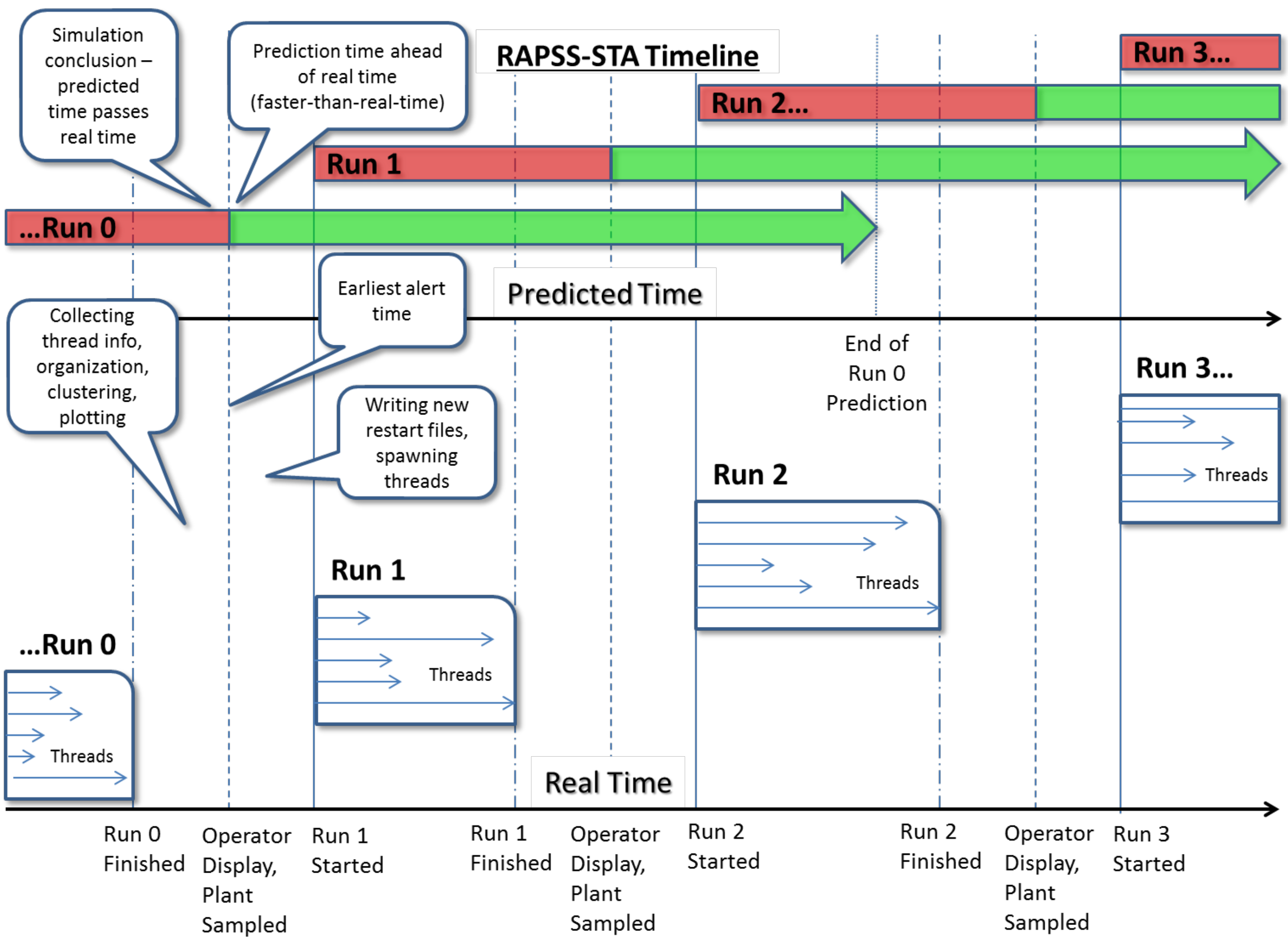


Figure 1.2 RAPSS-STA conceptual timeline

The timeline begins with run 0, whose initialization is not displayed on the chart. Run 0 predicts events happening past the point where run 1 starts, providing plenty of overlap to avoid any dead time. The lag time required to run the simulation is colored in red. If the simulation did not predict past physical time to run the program, it would be considered slower-than-real-time. The point at which the predicted time passes real time is colored green. This is what is referred to as faster-than-real-time.

After a run is finished, there is a small amount of processing time to collect and organize the thread information, pass the information to R (the tool used for statistical analysis) perform data analysis, write to output files, and plot the results. Immediately following is the earliest time that a senior operator could be alerted to RAPSS-STA's predictions of deleterious future events. The beginning of each cycle also involves some processing time to sample the plant, spawn new threads, load information on the threads, and write the appropriate scripts to run RELAP5 with the given conditions on each thread. Because the threads run with slightly, or drastically different conditions, each simulation takes different amounts of physical time to complete. If one finishes early, the master thread will wait until all slave threads have finished before performing the data analysis (see Section 2.9 for parallel computing terminology). While it does take some time to complete the RELAP5 simulations, the previous cycle's prediction should overlap the current simulation by plenty of time.

1.2 RAPSS-EOC

The second application described in this dissertation focuses on applying the RAPSS architecture to plume modeling for emergency operations centers in the event of a release of radioactive material outside of a nuclear power plant, spent fuel storage pools and casks, fuel cycle facilities, and radioactive handling facilities (See Figure 1.3). While the majority of the research focused on RAPSS-STA, it is important to realize that there are infinite possibilities for future applications of the RAPSS methodology in a similar fashion to what was illustrated in RAPSS-EOC. See Section 9 for more information on RAPSS-EOC.

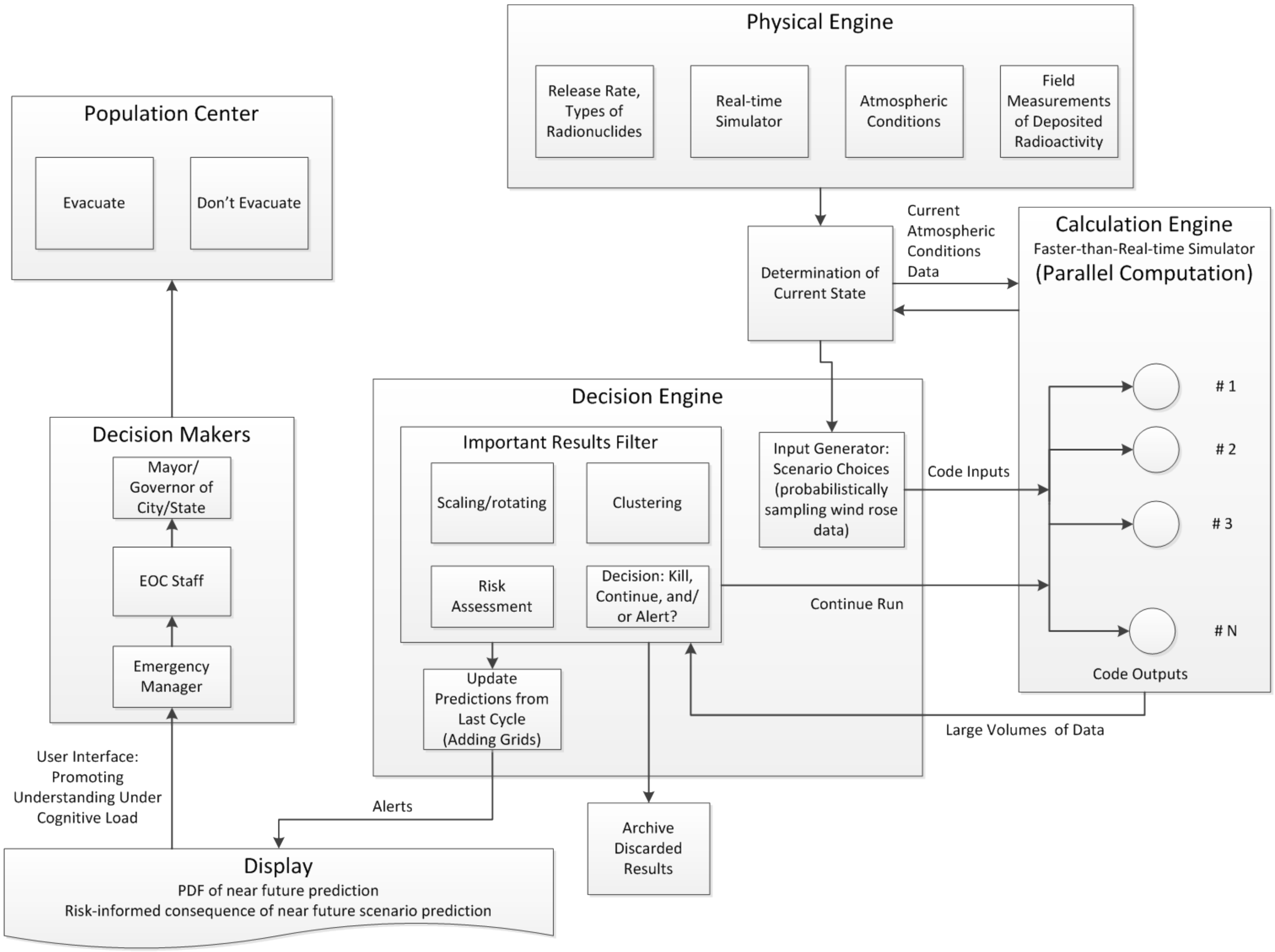


Figure 1.3 Conceptual outline of RAPSS-EOC implementation in an emergency operation center

The flow diagram for RAPSS-EOC (Figure 1.3) is very similar to the one for RAPSS-STA (Figure 1.1). The physical engine, in this case, is the atmosphere conditions in addition to estimates of release rate and type of radionuclides from the plant. An estimate of the current state is generated from sampled atmospheric conditions using the plume modeling program, labeled as the "calculation engine" in Figure 1.3. After an estimate of the current state is generated, wind rose data are sampled to determine probabilities for future wind speeds and directions. The possible future wind speeds and directions are run ahead in time across many parallel computational nodes. These data are output in the form of a grid of ground-level concentrations per unit area. These grids are added to the current state estimate, clustered with other similar scenarios, and checks are run to determine if a given population center is at risk. Once these determinations of risk are made, they are presented to senior emergency operations staff, who can notify prominent authority figures, such as city mayors, or state governors. These authority figures can then make the call of whether or not to evacuate the population center.

1.2.1 RAPSS-EOC Simulation Timeline

Figure 1.4 is useful to conceptually visualize the series of cycles RAPSS-EOC performs.

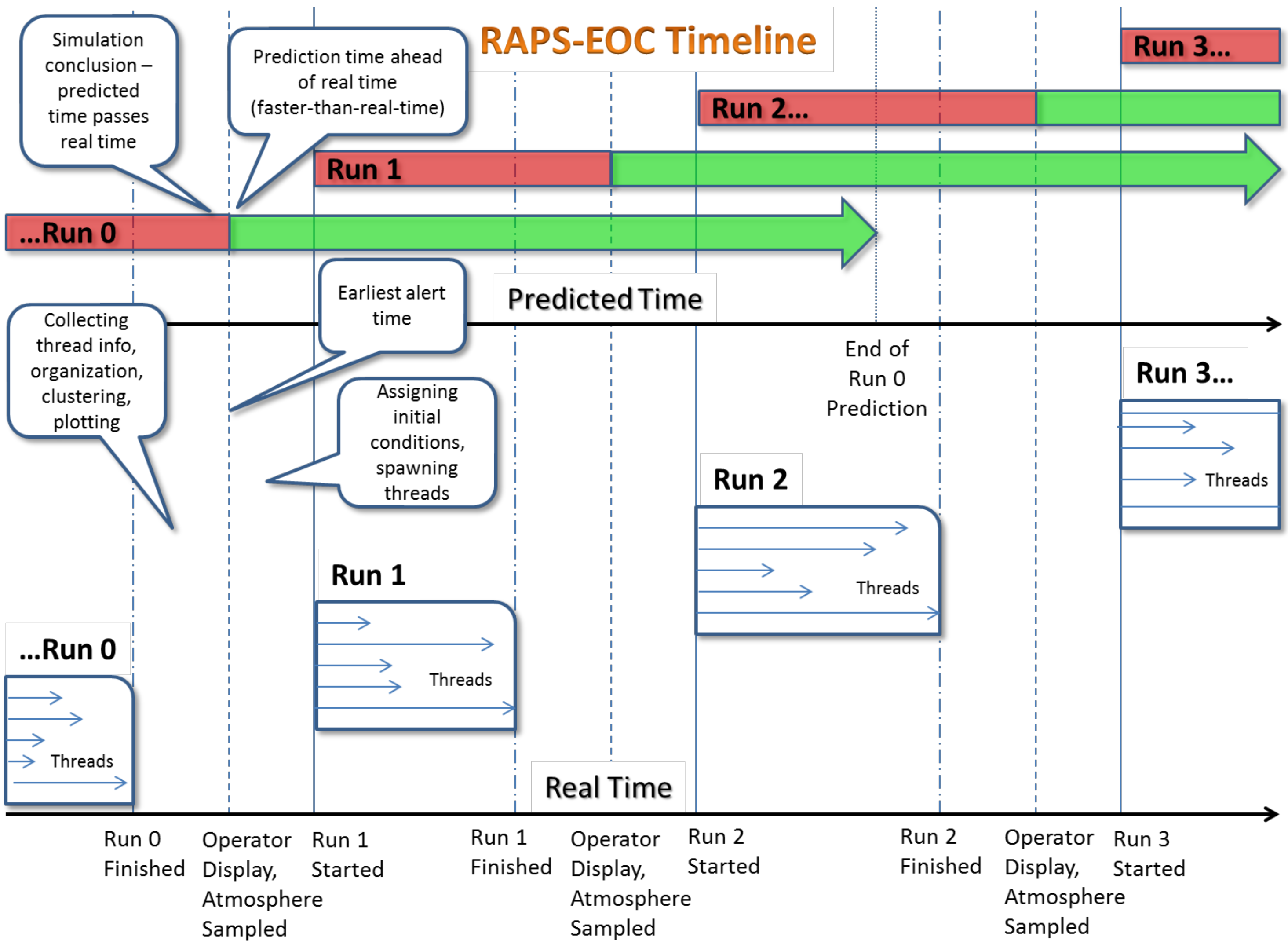


Figure 1.4 RAPSS-EOC conceptual timeline

While the majority of the RAPSS-EOC timeline displayed in Figure 1.4 is identical to the one displayed for RAPSS-STA in Figure 1.2, it is presented again to illustrate how similar the architecture is under different systems. The change in the case of RAPSS-EOC comes with sampling atmospheric conditions instead of plant conditions, and predicting ground level concentrations using a plume modeling program instead of future plant conditions using a thermal hydraulic simulation software.

1.3 Programming Languages

RAPSS was written in a combination of C++, Java, html, and R. The majority of the program and primary control structure was written in C++. It was compiled using g++ (GCC) 4.4.6, although other similar versions of g++ can be used.

R was used for data processing and for plot generation. R was convenient because it had the statistical tools such as principal component analysis and matrix multiplication already imbedded, allowing the researcher to avoid "reinventing the wheel."

The user interface for RAPSS-STA was written in html. This was convenient because it allowed the user to interact with the display by "clicking" on certain sections to obtain more information.

One module of RAPSS-STA was written in Java, LiteFTA, the stripped down version of OpenFTA (from http://www.openfta.com/) (see section 2.2.3). It served as the engine for calculating cuts sets, and probabilities based on the fault tree information provided by the user in the form of .fta and .ped files. LiteFTA generated .prp (cut sets and probability) and .mrp (Monte Carlo) files, which were read by RAPSS-STA and used to determine which transients to run. LiteFTA was precompiled using Java Runtime

Environment 1.6.0, and is not intended for modification. LiteFTA has similar functionality to OpenFTA, but operated by a UNIX terminal window instead of a Windows GUI.

What follows is an exhaustive literature review of the history of formal safety assessment in nuclear power, probabilistic risk assessment (PRA), thermal hydraulic simulation software, data management tools, atmospheric transport modeling techniques, numerical techniques used in numerical weather prediction, an introduction to parallel computing, a brief description of R, an overview of formal decision making, and risk and perception of risk. Discussion of RAPSS continues in Section 3. RAPSS-STA begins with Section 4, and RAPSS-EOC is detailed in Section 9.

2 Literature Review

2.1 History of Formal Safety Assessment in Commercial Nuclear Power

Prior to 1975, nuclear safety regulations in the US were written from deterministic conservative margins and models based on experience, test results, and expert judgment. Specifically, WASH-740, or the "Brookhaven Report," (U.S. Atomic Energy Commission (USAEC), 1957) estimated the maximum possible damage from a meltdown at a large reactor with no containment building, the worst possible meteorological conditions, and half the reactor core released into the atmosphere as 1 μm particles without much explanation of how this might occur. Needless to say, the results these assumptions yielded were unrealistic (i.e., 45,000 deaths, 100,000 injuries, and $17 billion in property damage). The industry was ready for a more detailed and realistic look at NPP risk.

2.1.1 WASH-1400 and Event/Fault Trees

WASH-1400, or the "Rasmussen Report" (USNRC, 1975), was a pivotal event in reactor safety analysis because it established the pattern for future nuclear power plant probabilistic risk assessments (see Section 2.2). WASH-1400 provided comparison with other non-nuclear risks, identified transients (loss of flow, rod withdrawal, etc.) and small-break loss of coolant accidents (SB LOCA) as major risk contributors (rather than just large-break (LB) LOCAs). It also identified human error as a major contributor, and showed the impact of testing, maintenance and common mode interactions. The Rasmussen Report further predicted that radiological risks from nuclear power plants were small when compared to societal risks. However, The American Physical Society (APS, 1984) later criticized WASH-1400's handling of radiological risks noting that the fatality estimates had considered only deaths during the first 24 hours after the accident, completely neglecting cancer deaths and radiation poisoning deaths after several weeks.

None the less, WASH-1400 was the first attempt to apply the methods of fault-tree/event-tree analysis to a nuclear reactor to determine the overall probability and consequences of an accident. The fault tree approach is a deductive process where an undesirable event, called the top event, is postulated, and the possible ways for this event to occur are systematically deduced. The fault tree does not necessarily contain all possible components failure modes; only the failure modes contributing to the top event occurrence are modeled (Modarres et al. 2010).

For instance, a top event may be, “no water delivered,” from a simple pumping system, such as the one displayed in Figure 2.1. This system consists of five valves, two pumps, a water source, and a sensing and control system, all which must run on AC power. Valves V-1 and V-2, V-4 and V-5, and pumps P-1 and P-2 are in parallel with one another, meaning, the failure of one doesn’t necessarily predict the failure of the entire system. However, failure of the water source (T-1), the sensing and control system (S), or the AC power would result in water delivery failure.

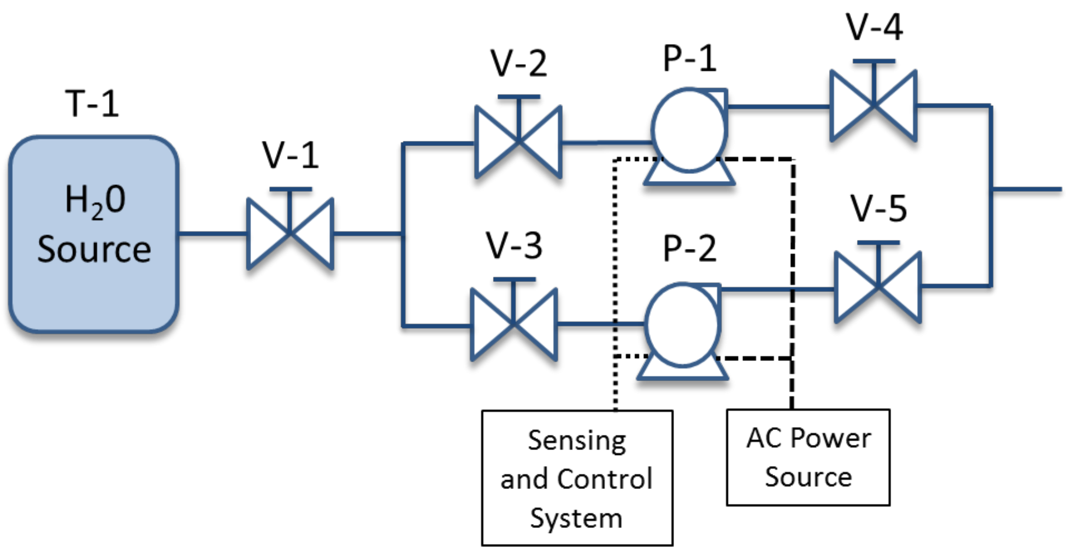


Figure 2.1 A simple example of a pumping system (from Modarres et al., 2010)

To construct a fault tree, one would write the top event, in this example, "no water delivered," at the top of the tree, and proceed down by writing the logical statements (if, and, or, xor, etc.) leading to the top event. An example is shown below in Figure 2.2: basic events are illustrated as circles, intermediate events are represented by rectangles, and undeveloped events are shown as rhombuses; logic gates are represented by their standard symbols.

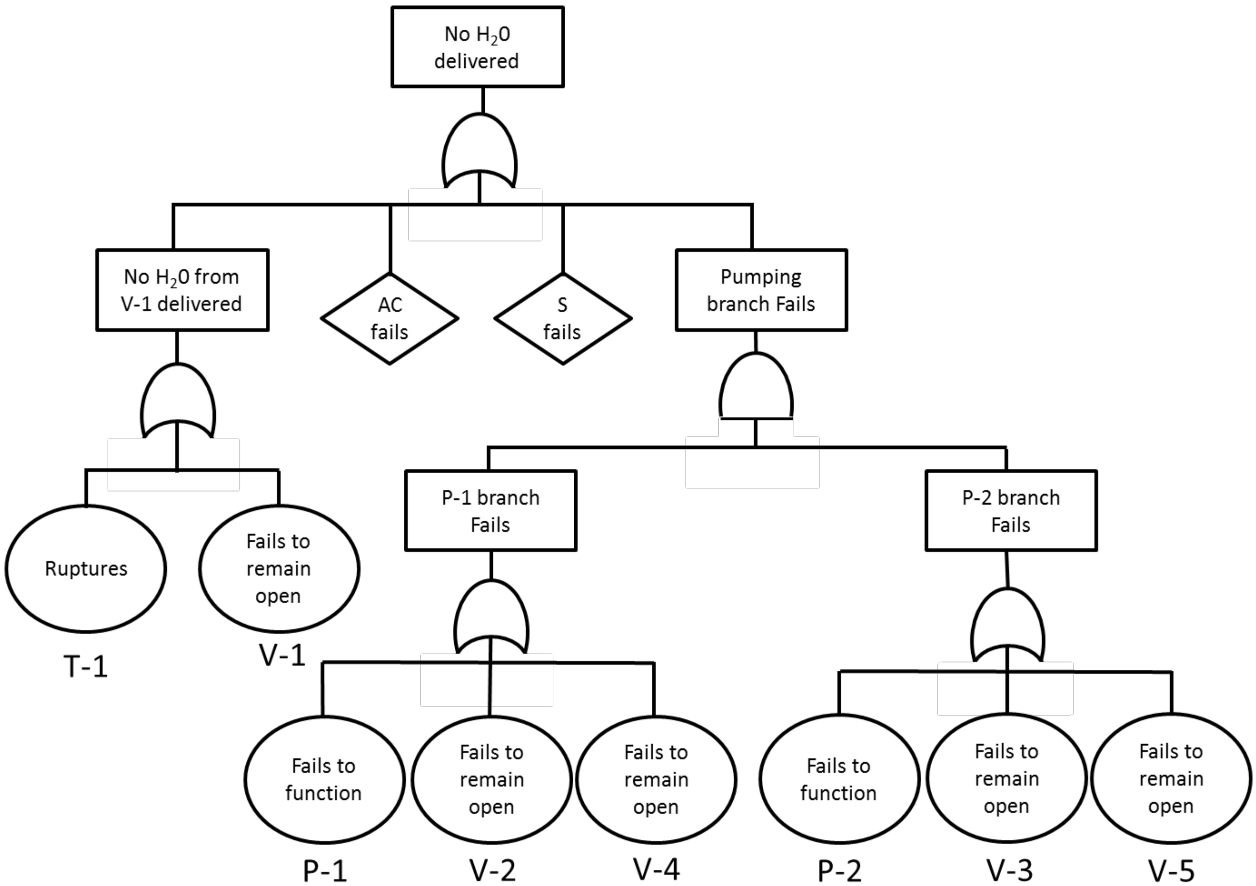


Figure 2.2 A simple example of a fault tree constructed from the pumping system in Figure 2.1 (from Modarres et al., 2010)

By starting with “no water delivered,” there are four immediate paths to follow: either the AC fails, the sensors fail, there is no water at V-1, or no water is delivered from the pumping branch. While there are further reasons for the AC or the sensors to fail, they are outside the scope of this analysis and thus are represented by the undeveloped rhombuses. In order for no water to be delivered at V-1, either the tank ruptures, or V-1 fails. The pumping branch will only fail to deliver water if both the P-1 and P-2 branches fail. For this to happen in the P-1 branch, either P-1, V-2, or V-4 would have to fail. In the P-2 branch, either P-2, V-3, or V-5 would have to fail. Fault trees such as this assist decision makers by explicitly detailing the possible ways to arrive at an undesirable end state.

The evaluation of fault trees is anything but straight forward. Development of a simple fault tree requires only a minimum understanding of the system; however, development of a more compact version for computational efficiency sake requires a much better understanding of the overall system logic. This involves the determination of *cut sets* which represent a single path that leads to the occurrence of the top event. A *minimum cut set* represents the minimum path that leads to the occurrence of the top event. Top event probability determination from cut sets involves the use of Boolean logic. The tree OR-gate represents the union of the input (e.g., A, B) events (AUB), where the probabilities of A and B are added (A+B), and the tree AND-gate represents the intersection of the input events ($A \cap B$), where the probabilities of A and B are multiplied ($A \cdot B$). Determining the probability of top events is challenging, especially when the number of cut sets is large. In general, there are 2^n-1 such terms in cut sets, where n is the number of cut sets (Modarres, Kaminskiy, & Krivtsov, 2010). For example, for the 13 cut sets generated for the pumping example (Figure 2.2) there are 8191 such terms (2^{13}-1). For larger fault trees, the exponential growth of cut sets can be challenging for even the most powerful mainframe computers.

Similar to fault trees, event trees help deduce the logical sequence of events leading to a failure. However, unlike fault trees, event trees start with an initiating event and show many different end states. To construct an event tree, one would start on the left with an initiating event, and proceed chronologically to the right passing through several “branch points” or points where systems could either succeed or fail. At each branch point, the upper branch shows the success of the event at that branch, and the lower shows failure. An example is shown in Figure 2.3. For the example sequence logic, components are shown in failure mode (e.g., AC implies the AC failed), while their compliments illustrate the component not failing (e.g., $\overline{AC}$ implies the AC did not fail).

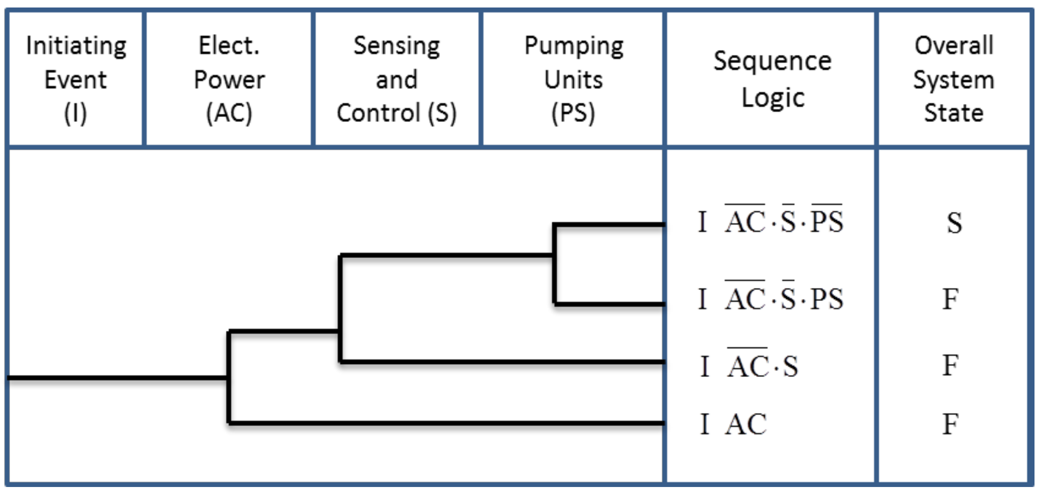


Figure 2.3 A simple example of an event tree (from Modarres et al., 2010)

Both fault trees (such as Figure 2.2), and event trees (such as Figure 2.3) are used in WASH-1400 to detail the sequence of events leading to core damage in nuclear power plants.

2.1.2 Post WASH-1400

The Energy Reorganization Act of 1974 created the Nuclear Regulatory Commission (NRC) and the Department of Energy (DOE) out of the old Atomic Energy Commission (AEC). The NRC appointed a review group to assess the quality of WASH-1400. This group concluded that “The uncertainties in WASH-1400’s estimates of the probabilities of severe accidents were… greatly understated” (Lewis et al., 1978). Reasons given were inadequate data base, a poor statistical treatment, lack of peer review, and an inconsistent propagation of uncertainties throughout the calculation. "In summary, we find that the fault-tree/event-tree methodology is sound, and both can and should be more widely used by the NRC. The implementation of this methodology in WASH-1400 was a pioneering step, but leaves much to be desired."

The Three Mile Island (TMI) accident in March, 1979 prompted many changes in the field of safety assessment. Up until this point, only design basis accidents (DBAs) were considered in the licensing process. TMI forced the industry and the regulators to take a closer look at severe, or beyond design basis accidents (BDBAs). A design basis accident is a postulated accident that a facility is designed to withstand without exceeding the offsite exposure guidelines of 10CFR100.11 (25 rem whole body dose or 300 rem to the thyroid from radioactive iodine) (USNRC 2002a). Beyond design basis accidents are more challenging to quantify because they usually involve multiple simultaneous failures and are defined by everything not planned for that results in significant core damage.

TMI caused the industry to rethink its safety goals. While the TMI release did not exceed the 10CFR100 limits, it did cause intense public outrage, which effectively undermined 10CRF100. As a result, the NRC set out to answer the question, "How safe is safe enough?" and published NUREG-880 (USNRC, 1983), which gave qualitative goals and suggested a quantitative goal of less than one core melt per 10,000 years. The NRC then issued a formal statement in 51 FR (Federal Register) 30028 (USNRC, 1986) which specifically defined two qualitative goals and two quantitative goals for 10CFR50 (USNRC 2002a). The quantitative goals of 51FR30028 were:

- The risk to an average individual in the vicinity of a nuclear power plant of prompt fatalities that might result from reactor accidents should not exceed 0.1%

of the sum of prompt fatality risks from other accidents to which members of the U.S. population are generally exposed; and

- The risk to the population in the area near a nuclear power plant of cancer fatalities that might result from nuclear power plant operation should not exceed 0.1% of the sum of cancer fatality risk resulting from all other causes.

And the qualitative goals were:

- Individual members of the public should be provided a level of protection from the consequences of nuclear power plant operation such that individuals bear no significant additional risk to life and health; and
- Societal risk to life and health from nuclear power plant operation should be comparable to or less than the risks of generating electricity by viable competing technologies and should bear no significant addition to other societal risks.

51FR20028 also recommended that, "The overall mean frequency of a large release of radioactive materials to the environment should be less than 1 in 1,000,000 years of reactor operation," which is now used in 10CFR50.109 (USNRC 2002a) for evaluating facility changes and updates.

2.1.3 NUREG-1150 and Accident Progression Event Trees

In 1988, the NRC requested each plant to assess its severe accident vulnerabilities. This Individual Plant Examination (IPE) would supplement the replacement for WASH-1400, NUREG-1150 (USNRC 1990). NUREG-1150 surveyed five plants, Surry Power Station near Newport News, Virginia, Peach Bottom Atomic Power Station near Lancaster, Pennsylvania, Zion Nuclear Power Station near Chicago, Illinois, Sequoyah Nuclear Generating Station near Soddy-Daisy, Tennessee, and Grand Gulf Nuclear Generating Station near Port Gibson Mississippi. The intent was to survey the spectrum of U.S. nuclear generating stations including three- and four-loop Westinghouse Pressurized Water Reactors (PWR, W3 & W4), as well as a variety of four- and six-loop Boiling Water Reactors (BWR- 4 & 6).

NUREG-1150 used the Accident Progression Event Tree (APET) approach to quantify accident progression and containment response. An APET identifies the variety of ways in which containment failure or bypass can occur, as well as the various severe accident processes that affect the mode of failure, timing of failure, and magnitude of environmental radioactive material release (Hakobyan et al. 2008). Unlike the WASH-1400 event trees, where branchings are based on the failure or success of safety systems in demand, APETs address questions such as, “Type of vessel breach?”, and “Amount of hydrogen released in-vessel during core damage?”, etc. Each question in an APET analysis has two or more answers, creating two or more branches to follow after each branch point. APETs are intended to determine environmental radiological release due to containment failure or bypass. In order to initiate an APET, prior analysis is necessary about the Plant Damage States (PDS) to be used as initial conditions for the analysis. Normally, fault tree analysis is not used to estimate branching probabilities in APETs; instead, branching probabilities are determined by comparing physical conditions obtained in the severe accident scenario with the branching criteria (Hakobyan, 2006). However, because of the uncertainties (epistemic and aleatory) in the analysis, there is not a deterministic outcome of “failure” or “no failure” for a scenario. Thus, uncertainty analysis is used to determine failure probability by performing several accident progression calculations using different modeling or input assumptions.

2.1.4 Post NUREG-1150

NUREG-1150 was eventually replaced by the NRC with the State-of-the-Art Reactor Consequence Analyses (SOARCA) report (USNRC 2011). The SOARCA analyzed two plants that the NRC believes are typical of the two basic types of U.S. commercial nuclear power plants: The Peach Bottom Atomic Power Station, and the Surry Power Station. This report greatly builds on NUREG-1150, incorporating onsite and offsite actions – including the implementation of mitigation measures and protective actions for the public (such as evacuation and sheltering) – that may prevent or mitigate accident consequences. It also used computer modeling techniques (MELCOR and MACCS2, see section 2.3) to understand how a reactor might behave under severe accident conditions, and how a release of radioactive material might impact the public.

2.2 Probabilistic Risk Assessment (PRA)

Probabilistic Risk Assessment (PRA) is a systematic procedure for investigating the ways in which complex systems are built and operated (Modarres et al. 2010). Kaplan and Garrick (1981) reduce the definition of PRA to three[1] basic questions, commonly referred to as "the set of triplets" definition:

1. What can go wrong that could lead to the exposure of hazards?
2. How likely is this to happen?
3. If it happens, what consequences are expected?

---

[1] Some authors (Garrick 2006) have added additional questions such as, "What are the uncertainties?" and "What corrective actions should be taken?"

The most significant result of the PRA is not the so-called bottom line value of the risk computed, but the determination of the system elements that substantially contribute to the risks of that system, the uncertainties associated with such estimates, and the effectiveness of various available risk reduction strategies (Modarres et al., 2010).

PRA, however, does contain some inherent limitations, struggling to quantify the items listed below (Apostolakis, 2004).

- Human error during accident conditions. These are both errors of omission (the crew failed to take prescribed actions) and errors of commission (the crew did something that worsened the situation). These errors are not handled well by PRA and research is underway to better quantify these sources. Some examples include the Technique of Human Error Rate Prediction (THERP) (Swain & Guttman, 1983), the Accident Sequence Evaluation Program (ASEP) (Wilson, 1993), and more recently, the Standardized Plant Analysis Risk (SPAR) (Gertman et al., 2005).
- Digital software failures. Historically, software systems were seen as black boxes with ascribed failure rates. While research is still ongoing (Li et al., 2005; Li et al., 2006; Stutzke & Smidts, 2001, Tumer & Smidts, 2011), traditional methods such as requiring extensive testing and the use of diverse software systems are making progress toward a more complete understanding of software failure modes.
- Safety culture. Defined by the Institute of Nuclear Power Operations (INPO, 2004) as "An organization's values and behaviors… that serve to make nuclear

safety the overriding priority." While it is relatively easy to blame an accident on a "bad safety culture," identifying the indicators of a "bad" or "good" safety culture is much more challenging. INPO, made progress towards safety culture quantitation by outlining the generic principles for a strong nuclear safety culture. INPO states that while safety culture is an intangible quantity, when thought of as a continuum, it is possible to determine, based on observable attributes (e.g., safety role modeling by leaders, cultivation of a questioning environment, the embracement of organizational learning, constant nuclear safety examination, etc.), whether a station tends toward one end of the continuum or the other.

- Design and manufacturing errors. Traditional safety methods of testing and equipment qualification address these errors; however, these are become especially challenging to quantify for equipment operating under unusual conditions, such as accident environments.

Surprisingly, there are no PRA requirements for the current generation of Light Water Reactors (LWRs). But in an odd bit of regulation, Regulatory Guide 1.174 (USNRC 2002b) requires the use of PRA for risk-informed decisions regarding changes to the plant's Licensing Basis (LB). Licensing basis changes are modifications to plant's design, operation, or other activities that require NRC approval. Since it is safe to assume that all plants will have many LB changes throughout their lifetime, all current generation LWRs are essentially required to keep an updated PRA. For all next generation (Gen III and beyond) reactors, 10CFR52 (USNRC 2009) requires the original license application to contain a PRA.

2.2.1 PRA Levels 1, 2, and 3

PRA is divided into three levels to help narrow the scope for the user's intent. Level 1 PRA contains accident frequency estimation only. It involves event/fault trees, which are used to define plant damage states in terms of scenarios leading to core damage and estimate the plant damage state frequencies (Core Damage Frequency (CDF)) based on success criteria for assuring core coolability. It starts with an initiating event (e.g., station black-out or loss of coolant accident) and proceeds until the reactor core is damaged. In comparison to the other two levels, once the right data are obtained, it is quick and cheap.

Level 2 PRA starts from the situation of core damage, and is carried out until containment is breached. It includes accident progression and radioactive materials transport analysis. Event/fault trees primarily address the occurrence of phenomenological events, such as hydrogen explosion or containment building failure. Accidents can be quantified by the severity of the radioactive material release. This determines the frequency and timing of core damage. The goal is to quantify probabilities and progression of the accident scenarios. Because accident progression differs for each plant damage state, accident progression analysis is necessary for each of the Plant Damage States (PDS). Regulatory Guide 1.174 (USNRC 2002b) provides a basis for making level 2 PRA risk-informed regulatory decisions using CDF and LERF to determine the acceptability of changes in risk.

Level 3 PRA starts from a radioactive release outside of containment. In conjunction with levels 1 and 2 PRA, a level 3 PRA estimates the health effects from radiation doses to the population around the plant, and land contamination from radioactive material released. These depend on several factors. For example, health effects depend on the population in the plant vicinity, evacuation conditions, and the path of the radioactive plume. The plume, in turn, is affected by wind speed and direction, as well as rain and snowfall. In a similar manner, land contamination depends on the characteristics of the radioactive release and how the surrounding land is used (NRC Website, 2011b). Level 3 PRA estimates the final measure of risk by combining consequence analysis with frequency. It is expensive, and thus performed only when the most accurate and detailed assessment of risk is required.

Apostolakis, (2004) cautions that PRA results should never be the sole basis for decision making by responsible groups. PRA is meant to inform human decision makers, not replace them.

2.2.2 SAPHIRE

SAPHIRE (Systems Analysis Programs for Hands-on Integrated Reliability Evaluations) (Smith et al., 2008) is a PRA software tool designed for reliability assessment (e.g., fault trees) and risk/safety assessment (e.g., event trees, core damage frequency), used by agencies such as the NRC, NASA, and the DOE for their risk-informed activities. SAPHIRE can be used for Level 1 PRA analysis to model a plant's response to initiating events, quantify associated core damage frequencies, and identify important contributors to core damage. It can also be used for Level 2 PRA severe accident evaluations by starting with the core damage state, and evaluating containment failure and/or release models. It can assume the reactor is at full power, low power, or in shutdown conditions. SAPHIRE's capabilities for performing PRA are:

- Graphical event/fault tree construction;

- Rule-based fault tree linking;
- Fast cut set generation;
- Fault tree flag sets;
- Failure data;
- Uncertainty analysis;
- Cut set editor, slice, display and recovery analysis tools;
- Cut set path tracing;
- Cut set comparison;
- Cut set and end-state partitioning;
- End-state analysis; and
- User-defined analysis types.

The primary functionality for most users of SAPHIRE lies in generating minimal cut sets for extremely large and complex event/fault tree logic models. Once the dominant cut sets are determined, they are used to quantify the overall probability of basic events. Three methods are available for this function. Fist, is the “rare event” approximation where the cut set values are simply summed. Second is the “minimal cut set upper bound approximation,” which is used in SAPHIRE as the default quantification method. Third, is the exact calculation method termed the “inclusion approach.” However, this method is exact only if the number of iteration passes is equal to the total number of cut sets, which can be achieved only for a limited number of cut sets.

After cut sets are generated, they can be used to obtain standard importance measures (e.g., Fussell-Vesely importance, Birnbaum importance, or the Risk Increase Ratio (see Section 2.11.1) for each basic event. They also can be used to propagate the epistemic uncertainty through the use of Monte Carlo or Latin Hypercube sampling. However, one limitation of SAPHIRE is lack of functionality for models explicitly capturing dynamic or time-dependent situations. For a discussion of software intended for this purpose, see Section 2.4.

2.2.3 OpenFTA and LiteFTA

OpenFTA is an advanced tool for fault tree analysis, similar in nature to SAPHIRE. OpenFTA is the open source product name for Formal-FTA, a product developed by Auvation. OpenFTA has the distinct advantage of being open source and uncopywritten, which made it a prime candidate for RAPSS integration.

OpenFTA is not hindered by artificial limitations such as a maximum number of gates of events. Events may appear in any number of transferred-in trees because during analysis transferred-in trees are treated as one large fault tree.

Minimal cut set generation is also very fast, and have been verified by the implementation of two independent cut set generators as well as by Monte Carlo Simulation (OpenFTA Website, 2012). After minimal cut sets are determined, the logically reduced tree can be quantitatively analyzed. OpenFTA provides the probability of system failure as well as the importance to the failure of each minimal cut set and event.

Fault trees can be built in the GUI, or typed manually. Once the tree is built in the GUI, OpenFTA generates *.fta and *.ped files, The *.fta file contains the information about the shape of the tree, and the *ped files contain the probability information about basic and undeveloped events. After the tree has been constructed, OpenFTA will determine minimal cut sets, both by Boolean logic and Monte Carlo simulation. The output files from these calculations are *.prp and *.mrp files.

While most users utilize the OpenFTA's GUI, RAPSS takes a different route. Because the code is open source, the cut set and Monte Carlo engines were extracted and pared down to only the necessary components and compiled to run via a UNIX terminal window. This new version of OpenFTA is appropriately named, LiteFTA. RAPSS calls shell scripts that runs LiteFTA with user specified conditions. LiteFTA reads *.fta and *.ped files and outputs *.prp and *.mrp files in the same directory. RAPSS then reads the *.prp and *.mrp files to determine the most probable transients to run.

2.3 Severe Accident/Thermal Hydraulic Codes:

While there are plenty of simulation codes that have historically been used to model thermal hydraulics and severe accidents in NPPs, the two heavyweights in the U.S. (serving slightly different functions) are MELCOR (Methods for Estimation of Leakages and Consequences of Releases) and RELAP5 (Reactor Excursion and Leak Analysis Program). Internationally, CATHARE (Code for Analysis of THermalhydraulics during an Accident of Reactor and safety Evaluation) is also widely used.

2.3.1 MELCOR/MACCS2

MELCOR is an engineering-level computer code that models the progression of severe accidents in LWRs. It is a successor to the Source Term Code Package (STCP) (Soffer et al., 1995). A broad spectrum of severe accident phenomena in both BWRs and PWRs are treated in MELCOR. These include thermal-hydraulic (TH) response in the reactor coolant system, reactor cavity containment, and confinement buildings; core heatup, degradation and relocation; core-concrete attack; hydrogen production, transport, and combustion; and fission product release and transport behavior. Current uses of MELCOR include estimation of severe accident source terms and their sensitivities/uncertainties in a variety of applications (Sandia National Lab, 2000).

If users of MELCOR are interested in simulating the accident progression outside of the containment structure, MACCS2 (MELCOR Accident Consequence Code System) (Sandia National Lab, 1998) is the answer. MACCS2 is a successor to MACCS, and CRAC2 (Calculation of Reactor Accident Consequences) (Aldrich et al., 1982). MACCS2 facilitates level 3 PRA analyses by considering atmospheric transport, short- and long-term mitigative actions/exposure pathways, deterministic/stochastic health effects, and economic costs of nuclear power plant disasters.

2.3.2 RELAP/SCDAP

RELAP5 is a thermal-hydraulic simulation code for LWRs developed at Idaho National Lab under sponsorship by the USNRC and USDOE, and a consortium of several countries and domestic organizations that were members of the International Code Assessment and Applications Program (ICAP) and its successor, the Code Applications and Maintenance Program (CAMP).

Specific applications include simulations of transients such as loss of coolant, anticipated transients without scram (ATWS), and operational transients such as loss of feedwater, loss of offsite power, station blackout, and turbine trip. In addition to calculating the behavior of the reactor coolant system during a transient, it can be used for simulation of a wide variety of hydraulic and thermal transients in both nuclear and nonnuclear systems involving mixtures of vapor, liquid, non-condensable gases, and nonvolatile solute (Idaho National Lab, 2003). While REALP5 still enjoys widespread use across the nuclear community, active maintenance will be phased out in the next few years (NRC Website, 2011) as usage of the more modern TRACE code grows (see Section 2.3.4).

Because RELAP5 was limited to transients that do not result in core damage, RELAP/SCDAP (Severe Core Damage Analysis Package) was developed by ISS (Innovative Systems Software) to model core damage in conjunction with TH phenomena as part of the international SCDAP Development Training Program (SDTP). SDTP consists of nearly 60 organizations in 28 countries supporting the development of technology, software, and training materials for the nuclear industry (Allison & Hohorst, 2008). SCDAP includes detailed modeling LWR core components, upper plenum structures, and is capable of modeling core debris and molten pools as well as lower plenum debris and vessel structures (ISS website, 2011), allowing a RELAP/SCDAP to serve similar functions to MELCOR.

2.3.3 TRAC

TRAC (Transient Reactor Analysis Code) (Spore et al., 1981), is a legacy thermal hydraulics simulation software and was, in 1980, split into two flavors, TRAC-P (for PWRs) and TRAC-B (for BWRs). TRAC-P analyzed LB LOCAS as well as modeled TH phenomena in 1- or 3-D components for PWRs. TRAC-B could also model TH phenomena in 1 or 3-D (for BWRs), and could analyze SB as well as LB LOCAS. The original intent of TRAC was to include significantly more detail than RELAP, and as a consequence of increased computational time, only be used to spot check RELAP results. However, over the years, TRAC became much faster without loss of detail, and RELAP became significantly more detailed. As a result the codes evolved similar capabilities.

2.3.4 TRACE

In the mid-nineties, the NRC sought to consolidate RELAP5, TRAC-P, TRAC-P, and a special purpose BWR code, RAMONA (H. S. Cheng & Rohatgi, 1996) due to the overhead involved in maintaining these codes. The result was a software package named TRAC/RELAP Advanced Computational Engine (TRACE). TRACE is a component-oriented reactor systems analysis code designed to analyze reactor transients and accidents up to the point of significant fuel damage, and is considered the NRC's current flagship thermal hydraulics code (NRC Website, 2011). TRACE is a finite-volume, two-fluid compressible flow code utilizing a combination of one-, two-, and three-dimensional flow geometries to model heat structures and control systems that interact with component models and the fluid solution (Murray, 2007).

2.3.5 CATHARE

In the international community, French researchers at AREVA, CEA (French Atomic Energy Commission), EDF (French utility), and IRSN (French Nuclear Safety Institute) released the Code for Analysis of THermalhydraulics during an Accident of Reactor and safety Evaluation (CATHARE) (CEA Website, 2011). CATHARE is a system code for safety analysis, accident management, definition of plant operating procedures, and research and development. It is also used to quantify conservative analysis margins and for licensing. Since France does not have any BWRs, CATHARE only deals with PWR analysis. The main objectives of CATHARE are to model LB LOCAS, SB LOCAS, intermediate break LOCAS, Steam Generate Tube Rupture (SGTR), and as well as other transients (i.e., loss of residual heat removal system (RHRS), loss of SG feed water, etc.)

2.3.6 Assessment of Existing Codes for RAPSS application

While TRACE, RELAP5, MELCOR, or CATHARE could all theoretically be used to in RAPSS, the question would be if any of the codes could run fast enough with enough precision to provide useful information to the user. Since these codes were not written for this intent, it is not believed at this time that the full implementation of any of the aforementioned codes will have the ability to run in a faster-than-real-time environment. However, opportunities still exist for running ensembles of streamlined, lower-resolution versions of the codes, or using newer, faster, cutting edge simulations software.

2.4 Dynamic Probabilistic Risk Assessment (DPRA)

The word, "dynamic" has several different meanings when applied to probabilistic risk assessment. Some use it to describe a "living PRA," or periodic updates of the plant's PRA to reflect any changes in the plant configuration. Another version is used to explicitly account for equipment aging. The third is a PRA that can be used as an instantaneous or average "risk meter" to help operators and plant personnel in making daily decisions regarding plant configuration changes and possibly as a decision aid in accident conditions (Hsueh & Mosleh, 1996). And the fourth is used to describe an approach that includes explicit modeling of deterministic dynamic processes taking place during plant system evolution combined with stochastic modeling (Hakobyan et al., 2008). It is the fourth definition that will be the focus of the majority of this section.

In the Dynamic Event Tree (DET) analysis, event trees are run simultaneously starting from a single set of initial conditions. In most cases, DETs are generated by direct coupling with a dynamic model of the plant using system simulation codes (e.g., MELCOR, RELAP5), probabilistic behavior of system components and parameters, and human action (Mandelli, 2011). Branching occurs either when specified by the user, or when action is required by the system or operator (Hakobyan et al., 2008). The plant simulator evaluates the temporal behavior of the plant and determines the timing and natures of each branch. The results of these analyses are usually very difficult to organize without risk contributor identification algorithms for each initiating event (see Section 2.5).

2.4.1 DYnamic Logical Analytical Methodology (DYLAM)

Software development for DETs began in the 1980s at the Joint European Center in Ispra, Italy. They developed the DYnamic Logical Analytical Methodology (DYLAM) (Cojazzi, 1996), which was not only used for nuclear power plant simulations, but also in the chemical, aeronautical and other industries (Hakobyan et al., 2008). The intent of DYLAM was to couple the probabilistic and physical behavior of a system for more detailed reliability analysis. DYLAM acted as a driver for a system simulation code by assigning initial states to each branch and triggering stochastic transitions in the component states. For each branch, the probability of the system achieving that branch was evaluated from the user-provided branching probabilities. The probability of consequence occurrence (or top event) was the sum of all branch probabilities leading to the top event (Cojazzi, 1996).

2.4.2 Accident Dynamic Simulation (ADS)

Accident Dynamic Simulation (ADS) methodology was developed by Hsueh and Mosleh in the early 90s (Hsueh & Mosleh, 1996). ADS was novel because it broke down the accident analysis model into different paths according to the nature of the processes involved, simplifying each part while retaining its essential features. ADS was originally designed to run in serial, following a single path to an end point, then retrace to the last branch point, and choose a new path to follow.

Performance was greatly improved when Zhu et al. (2008) proposed a multiprocessor version of ADS. Aside from parallel processing capabilities, this version had several efficiency improvements. For instance, a reduction of the number of risk scenarios was achieved by combining system and operator states that lead to similar end states, and biasing the system and operator states toward interesting or risk significant end states.

ADS had another leap forward when researchers at the University of Maryland paired it with the Information, Decision and Action in a Crew (ADS-IDAC) cognitive model (Chang & Mosleh, 2007; Coyne, 2009; Coyne & Mosleh, 2009), which assisted in predicting situational contexts that might lead to human errors. ADS-IDAC generated discrete DETs by applying branching rules to reflect variations in crew response to plant events, for example, slow or fast procedure execution speed, skipping steps, reliance on memorized information, and activation of mental beliefs among others. ADS-IDAC provided a more realistic assessment of human error events by directly determining the effect of operator behaviors on plant parameters.

2.4.3 Monte Carlo Dynamic Event Tree (MCDET)

In the early 2000s, German researchers developed a novel Monte Carlo technique of DET simply named Monte Carlo Dynamic Event Tree (MCDET) (Hofer et al., 2004; Kloos & Peschke, 2008). Its intent was to model the response of the safety features of the plant and the reaction of the operating crew during severe accident progression.

MCDET was implemented as a stochastic model that could be operated in tandem with any deterministic dynamics code, (e.g., MELCOR, RELAP, see Section 2.3). The dynamics code would generate a discrete DET and compute the time histories of all variables along each path together with the path probability. MCDET focused on transitions (or branching points) of the event trees. Each transition had two characteristics: "when" it occurred, and "where to" it went, which may be either deterministic, discrete and random, or continuous and random. MCDET sampled all combinations of the "when" and "where to" for each transition. Discrete and random "when and/or "where to" were generally accounted for by dynamic DET analysis, while continuous and random "when" and/or "where to" were sampled with Monte Carlo simulation. Probabilistic "cutoff" values were utilized to allow termination of any braches below the specified probability.

2.4.4 Analysis of Dynamic Accident Progression Trees (ADAPT)

More recently, researchers at the Ohio State University developed Analysis of Dynamic Accident Progression Trees (ADAPT) (Hakobyan, 2006; Hakobyan et al., 2008). This methodology sought to account for uncertainties that arise from lack of experience and knowledge (i.e., epistemic), as well as stochastic phenomena such as creep rupture and hydrogen burn (i.e., aleatory). Similar to the other DETs mentioned above, the philosophy was to let a system simulation code determine the pathway of the scenario within a probabilistic context. When conditions were achieved leading to alternative accident pathways, a driver generated new scenario threads (or branches) for parallel processing. To avoid unacceptable run times due to exponential growth of branches, there were user defined truncation rules such as branch probabilities falling below a cutoff value, or the simulation exceeding a given time limit. ADAPT is plant simulator independent as long as the simulator had the following four features: (1) it reads itDims input from command-lines and/or text file, (2) it has check-pointing feature, (3) it allow user-defined control-functions (e.g., stopping if a certain condition is true), and (4) its output can be utilized to detect stopping condition.

2.5 Data Management

The major challenge in using DETs is the heavy computational and memory requirements; each new branch can contain the time evolution of a large number of variables. This yields hundreds to thousands of scenarios that often lead to very similar end-states.

2.5.1 Principal Component Analysis (PCA)

It becomes necessary to preprocess the data in most practical applications to reduce the dimensionality. Due to the often correlated nature of the data, methods such as Principal Component Analysis (PCA) (Ramsey & Schafer, 2002) or Multi-Dimensional Scaling (MDS) (Borg & Groenen, 2005) are often used to transform the original set of possibly oblique coordinate axes to a new set of orthogonal axis. The variables of interest may differ in units (e.g., temperature, pressure, etc.), as well as range. This can be overcome by either: normalizing each dimension onto the [0, 1] interval, or normalizing each dimension by dividing it by its standard deviation (Mandelli, 2011). In PCA, a new set of orthogonal axes is obtained by finding the eigenvectors of the covariance matrix of the data (dimension *x* observation) in order to project the data onto the new coordinate system. This not only reduces the dimensionality, but also allows for traditional Euclidean distances (Equation (2.1)) to be used for clustering analysis.

The steps to perform PCA are fairly straightforward:

- Get some data.
- Subtract the mean from each dimension. This produces a data set whose mean is zero.
- Normalize the data by dividing by diving by the standard deviation of each dimension.
- Calculate the covariance matrix.
- Calculate the eigenvectors and eigenvalues of the covariance matrix.
- Trim dimensions that represent smaller than a given threshold of variance (e.g., 5% or less). The eigenvectors that correspond to the largest eigenvalues represent the most variance in the data. To determine the proportion of variance, add the eigenvalues and divide by the one of interest. The remaining matrix of eigenvectors is called the feature vector and is not square.
- Multiply the feature vector matrix by the normalized data to project the data onto a new set of axes.

This yields the original data only in terms of the axes that represent the most variability. To obtain the original data back

- Multiply the inverse of the feature vector by the projected data. This yields a data set that is the same size as the original data.
- Multiply by the standard deviation of each dimension.

- Add back in the mean of each dimension.

2.5.2 Linear Approximation Intervals

However, both PCA and MDS have the disadvantage of only allowing for linear correlations. Mandelli et al. (2011) sidestepped this issue by dividing the data into small subintervals for linear approximation. The interval size was determined by comparing the covariance matrices between the intervals; when the covariance was similar between intervals, the interval size was increased; if the covariance differed significantly, the interval could be decreased.

2.5.3 The Mean Shift Algorithm

Once the initial dimensionality is reduced, grouping can be performed. Milano et al. (2009) use a probabilistic Fuzzy C-means (FCM) approach to group data from an ADS-REALP5 DET simulation. Because FCM is based on fuzzy sets, it allows a data point to belong to more than one cluster. However, FCM is only able to identify a predetermined number of clusters, having ellipsoidal or spherical geometry. This and other similar approaches (Zio et al., 2009) implement classification rather than clustering algorithms, which imply that the number of clusters have been set *a priori* by the user and the algorithm simply performs the group membership.

In clustering, however, the algorithm determines the number of clusters based on a set of similarity rules specified by the user (Mandelli, 2011). Similarity can be found by measuring the distance, $d(\vec{x}_i, \vec{x}_j)$, between two data points, $\vec{x}_i$, and $\vec{x}_j$ using the Euclidian distance formula:

$$d(\vec{x}_i, \vec{x}_j) = \left(\sum_{d=1}^{\delta} \left| x_i(d) - x_j(d) \right|^2 \right)^{1/2} \qquad (2.1)$$

where δ is the dimensionality.

The idea of clustering can be summarized as the process of finding partitions of the original data set and characterizing each partition by a representative data point. Scenario clustering aims to:

- Identify the scenarios that have similar behavior (i.e., identify the most evident classes); and
- Decide cluster membership for each event sequence (i.e., classification).

Mandelli (2011) applied clustering analysis to DETs through the use of the Mean Shift Algorithm (Cheng, 1995) to drastically reduce the amount of information yield from an ADAPT-MELCOR simulation. The Mean-Shift algorithm is a kernel-based, non-parametric density estimation technique used to find the modes of unknown distributions, which correspond to regions with high data density, separated by areas of low density. It is a fairly simple iterative procedure that shifts each data point to the average of data points in its neighborhood (Cheng, 1995). The Mean Shift Algorithm is able to identify clusters of arbitrary shapes, and hence, clusters are not limited by topological figures such as spheres or ellipsoids. The cluster centers obtained by Mandelli (2011) illustrated the most representative scenarios from a ADAPT-MELCOR simulation, allowing the analysis to be carried out on a much smaller set of representative scenarios.

2.6 Atmospheric Transport Modeling

Modeling a radioactive material release outside of a nuclear power plant is a complex process. The ultimate goal is to determine the quantity of radionuclides reaching man or other biota. This is calculated by estimating quantities such as external submersion dose from a contaminated cloud, external dose from contaminated soil deposition, internal inhalation dose from the cloud, and internal dose from ingestion of contaminated water and/or foodstuff.

2.6.1 Gaussian Puff/Plume Modeling

The most widely used diffusion model is the Gaussian Plume/Puff model (GPM) (Figure 2.4).

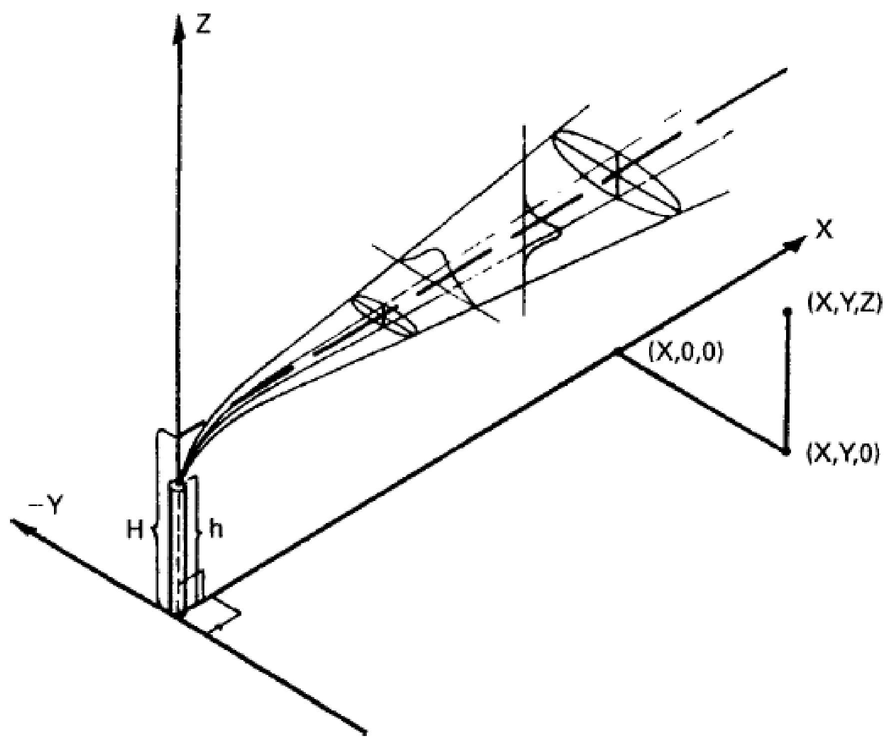


Figure 2.4 Gaussian plume dispersion model for a continuous point source (Cember & Johnson, 2009)

For the purposes of this discussion, a plume is defined as a continuous release from a point source for an arbitrarily long amount of time. The most common form of the GPM is expressed similarly to Equation (2.2) (Martin, 2006):

$$\mathrm{X(x,y,z)} = \frac{\mathrm{Q}}{2\pi\sigma_y\sigma_z u} e^{-\frac{1}{2}\left(\frac{y}{\sigma_y}\right)^2} \left[e^{-\frac{1}{2}\left(\frac{z-h_e}{\sigma_z}\right)^2} + e^{-\frac{1}{2}\left(\frac{z+h_e}{\sigma_z}\right)^2} \right] \tag{2.2}$$

Where:

- X(x,y,z) is the steady state concentration at a point (x, y, z), expressed in g m^{-3} or Ci m^{-3};
- Q is the source emission rate (Bq/s or Ci/s);
- σ_y, σ_z are crosswind and vertical plume standard deviations of distances, usually determined by the Pasquill Stability Class (m) (see section 2.6.3) ;
- *u* is the average wind speed (m/s); and
- h_e is the effective stack height, as described by Equation (2.3).

For the ground contamination (z=0) case, the concentration is effectively doubled by combining the exponential terms in the brackets of Equation (2.2). This can be thought of as accounting for reflection of the plume with the ground. The plume essentially folds over on itself to double the ground-level concentration (Martin, 2006).

To account for variables such as the exit velocity of the gas, the term, effective stack height, h_e, is calculated by Equation (2.3):

$$h_e = \mathrm{h} + \mathrm{d}\left(\frac{v}{\mu}\right)^{1.4}\left(1+\frac{\Delta \mathrm{T}}{\mathrm{T}}\right) \tag{2.3}$$

Where:

- h is the actual chimney height (m);
- d is the chimney outlet diameter (m);
- v is the exit velocity of the gas (m s^{-1});
- μ is the mean wind speed at the top of the chimney (m s^{-1});
- ΔT is the difference between ambient and effluent gas temperatures (K); and
- T is the absolute temperature of the effluent gas (K).

A puff is defined as a single point source release, modeled with respect to time and position. The Gaussian Puff Model is commonly used for a single, instantaneous release, illustrated in Figure 2.5.

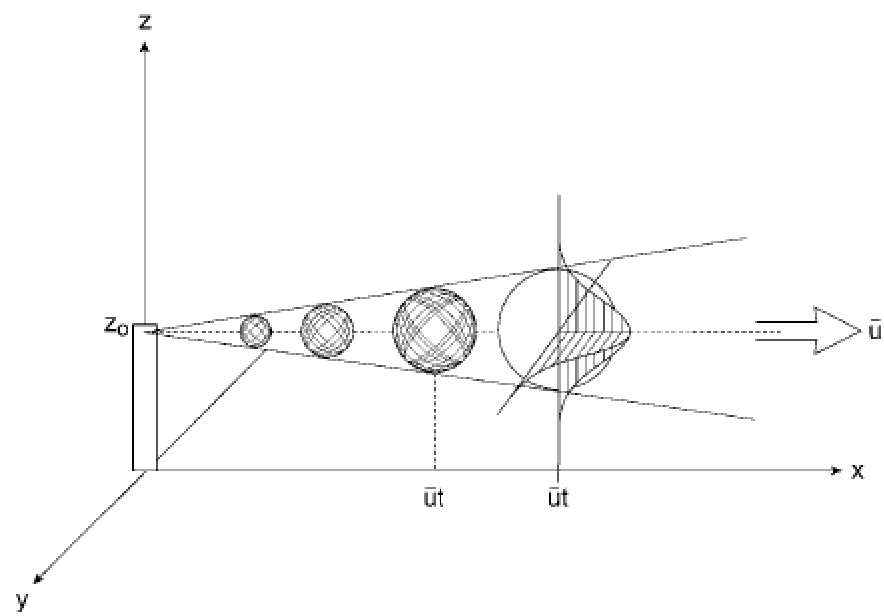


Figure 2.5 Gaussian puff model (Martin, 2006)

Another way to look at the Gaussian Puff Model is as the derivative of the plume model with respect to time. In other words, the plume model is made up of an infinite number of arbitrarily small puffs for an infinite amount of time. The puff model is described analytically by Equation (2.4) (Martin, 2006):

$$X(x,y,z,t) = \frac{Q_p}{(2\pi)^{\frac{3}{2}}\sigma'_y\sigma'_z\sigma'_x} e^{-\frac{1}{2}\left(\frac{y}{\sigma'_y}\right)^2} \cdot e^{-\frac{1}{2}\left(\frac{x-ut}{\sigma'_x}\right)^2} \cdot e^{-\frac{1}{2}\left(\frac{z-h_e}{\sigma'_z}\right)^2} \tag{2.4}$$

Where:

- X(x,y,z,t) is concentration at a point and time (x, y, z, t), expressed in g m^{-3}, Ci m^{-3}, or Bq m^{-3};
- Q_p is the total release of material (Bq or Ci);
- σ'_y, σ'_z σ'_x are crosswind, vertical, and horizontal (respectively) puff standard deviations of distances (m). These are not the same as for the plume model;
- *u* is the average wind speed (m/s); and
- h_e is the effective stack height (m).

In order to calculate a plume over a certain time interval (t_1,t_2), one simply integrates the puff model, as shown in Equation (2.5).

$$X_{plume} = \int_{t_1}^{t_2} X_{puff}(x,y,z,t)\,dt \tag{2.5}$$

2.6.2 Extensions of the Gaussian Plume/Puff Models

One limitation of the puff and plume models is that the equations are limited to expressing the plume behavior only in the direction of the wind. With real-world situations, involving changing wind direction, a few adjustments need to be made.

In certain situations it is useful to convert the GPM into cylindrical coordinate system, similar in nature to Green et al. (1980). Assuming that the observation line is at angle θ with respect to the direction of the wind, the following approximations can be used:

$$x = r\cos\theta \cong r\left(1 - \frac{\theta^2}{2}\right) \tag{2.6}$$

And

$$y = r\sin\theta \cong r\theta\left(1 - \frac{\theta^2}{6}\right) \tag{2.7}$$

Retaining terms in expansions through θ^2 it is possible to represent the diffusion terms as:

$$\sigma_y = \frac{Kr\left(1 - \frac{\theta^2}{2}\right)}{\left[1 + \frac{r}{a}\left(1 - \frac{\theta^2}{2}\right)\right]^p} \approx \frac{Kr}{\left[1 + \frac{r}{a}\right]^p} e^{-\frac{1}{2}\theta^2\left(1 - \frac{pr}{a+r}\right)} \tag{2.8}$$

and

$$\sigma_z = \frac{Lr\left(1 - \frac{\theta^2}{2}\right)}{\left[1 + \frac{r}{a}\left(1 - \frac{\theta^2}{2}\right)\right]^q} \approx \frac{Lr}{\left[1 + \frac{r}{a}\right]^q} e^{-\frac{1}{2}\theta^2\left(1 - \frac{qr}{a+r}\right)} \tag{2.9}$$

By substituting Equations (2.8) and (2.9) into Equation (2.2), after some reduction, the GPM can be expressed in cylindrical coordinates as:

$$\chi(r,\theta,z,h_e) = \frac{Q}{u\pi T\Xi} e^{-\frac{r^2\theta^2}{2T^2}} e^{-\Omega\frac{\theta^2}{2}} \left[e^{-\frac{1}{2}\left(\frac{z-h_e}{\sigma_z}\right)^2} + e^{-\frac{1}{2}\left(\frac{z+h_e}{\sigma_z}\right)^2} \right] \tag{2.10}$$

where Ω is a correction term given by:

$$\Omega = \alpha(r) + \beta(r)\frac{H^2}{\Xi^2} \tag{2.11}$$

with

$$\alpha(r) = -2 + \frac{pr}{a+r} + \frac{qr}{a+r} \tag{2.12}$$

$$\beta(r) = 1 - \frac{qr}{a+r} \tag{2.13}$$

And T and Ξ are functionally identical to the dispersion parameters σ_y and σ_z but with x replaced by r.

However, for the sake of simplicity, a rotated Cartesian coordinate system in the direction of the wind for a single wind direction at a time was chosen instead of the GPM in cylindrical coordinates. Where the rotations:

$$\begin{aligned} x' &= x \cdot \cos(\theta) - y \cdot \sin(\theta) \\ y' &= x \cdot \sin(\theta) + y \cdot \cos(\theta) \end{aligned} \tag{2.14}$$

are used with θ equaling the wind direction in radians. There are two particularly interesting features of these rotations. First, while it may make sense to some that θ describes the direction the wind is *blowing*, wind is usually expressed in the direction the wind is *coming from*, or 180 degrees different than the value of θ. Second, common practice is to describe North as 0 degrees, and East as 270 degrees. In mathematics, however, the direction commonly thought of as East is along the positive X axis, referred to as 0 degrees, and north as 90 degrees. While it ultimately doesn't matter which system (mathematical, or directional) the plume model user takes, it is important that one decides on one system and sticks with it (similar to driving on the right or left hand side of the road). For the sake of this dissertation, the mathematical model of East equaling 0 degrees was used, and everything else was converted accordingly.

2.6.3 Pasquill Stability Classes

Turbulence in ambient air greatly affects the rise and dispersion of plumes. Turbulence can be categorized into increments, or "stability classes." These Pasquill Stability Classes range from A-F, where A represents the least stable/most turbulent conditions, and F represents the most stable/lest turbulent conditions. Table 2.1 is used as a rule of thumb for determining Stability classes:

Table 2.1 Pasquill Stability Classes

Surface Wind Speed		**Daytime Incoming Solar Radiation**			**Nighttime Cloud Cover**	
(m/s)	(mi/hr)	Strong	Moderate	Slight	>50%	<50%
<2	<5	A	A-B	B	E	F
2-3	5-7	A-B	B	C	E	F
3-5	7-11	B	B-C	C	D	E
5-6	11-13	C	C-D	D	D	D
>6	>13	C	D	D	D	D

For a given stability class, Figure 2.6 and Figure 2.7 are used to determine horizontal and vertical diffusion standard deviation coefficients (respectively).

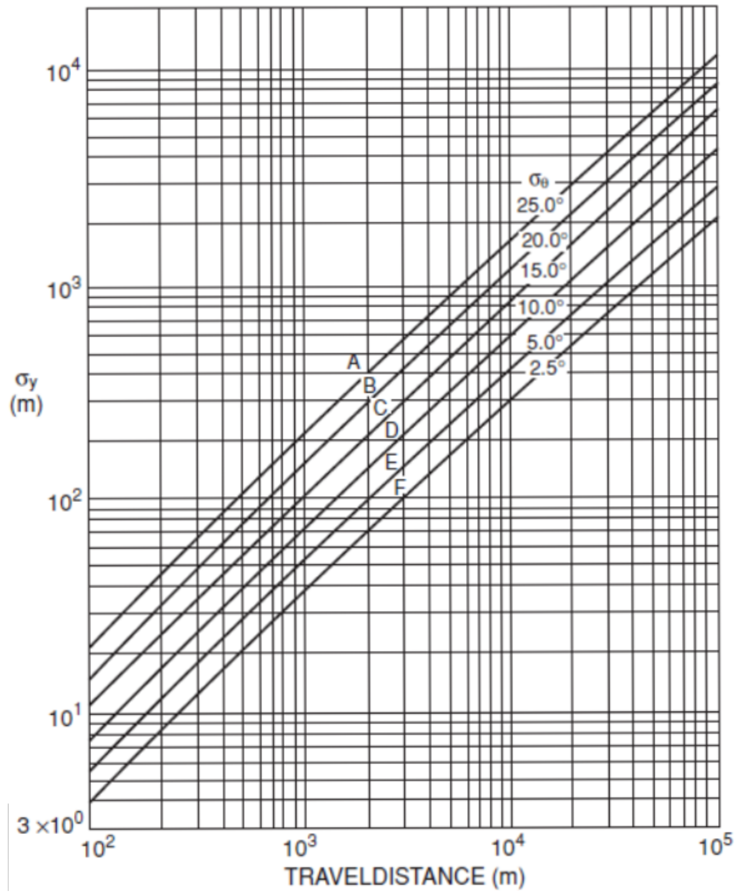


Figure 2.6 Horizontal diffusion standard deviation versus downwind distance from a point source (Cember & Johnson, 2009).

The curves in Figure 2.6 correspond to the analytic approximation given in Equation (2.15):

$$\sigma_{z_{plume}} = \frac{Lx}{\left[1 + \frac{x}{a}\right]^q} \tag{2.15}$$

Where x is the downwind distance in meters, and the values of L, a, q, are given in Table 2.2.

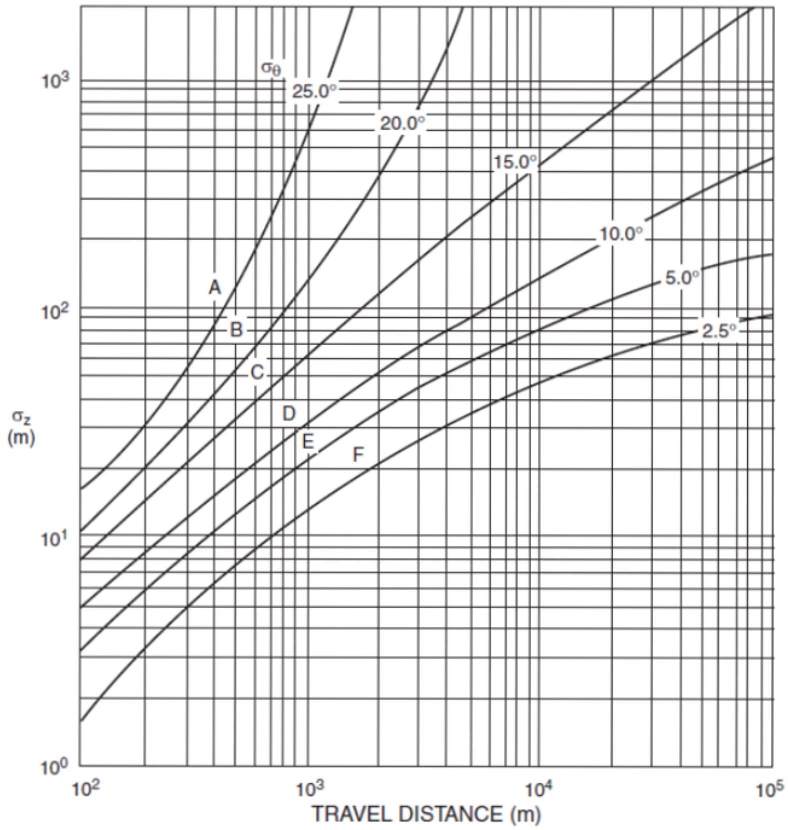


Figure 2.7 vertical diffusion standard deviation versus downwind distance from a point source (Cember & Johnson, 2009).

The curves in Figure 2.6 correspond to the analytic approximation given in Equation (2.16):

$$\sigma_{y_{plume}} = \frac{Kx}{\left[1+\frac{x}{a}\right]^{p}} \quad (2.16)$$

Where *x* is the downwind distance in meters, and the values of *K*, *a*, *P*, are given in Table 2.2.

Table 2.2 Gaussian Plume Model Dispersion Parameters (Green et al., 1980)

	a (km)	L (m/km)	q	K (m/km)	p
A	0.927	102.0	-1.918	250	0.189
B	0.370	96.2	-0.101	202	0.162
C	0.283	72.2	0.102	134	0.134
D	0.707	47.5	0.465	78.7	0.135
E	1.07	33.5	0.624	56.6	0.137
F	1.17	22.0	0.700	37.0	0.134

For the Gaussian Puff model, the curves are generated from a different set of simple exponential equations (2.17), (2.18), and (2.19).

$$\sigma_{x_{puff}} = p_x \cdot x^q \quad (2.17)$$

$$\sigma_{y_{puff}} = p_y \cdot x^q \quad (2.18)$$

$$\sigma_{z_{puff}} = p_z \cdot x^q \quad (2.19)$$

Where q, p_x, p_y, and p_z are given by Table 2.3:

Table 2.3 Values used in the Gaussian Puff Model for q, p_x, p_y, and p_z

	q	p_x, p_y	p_z
A	0.92	0.14	0.53
B	0.92	0.14	0.53
C	0.92	0.06	0.15
D	0.92	0.06	0.15
E	0.92	0.02	0.04
F	0.92	0.02	0.04

2.6.4 RASCAL

The current flagship code used by the NRC's emergency operations centers is RASCAL (Radiological Assessment System for Consequence AnaLysis). It was designed for making dose projections for atmospheric releases during radiological emergencies (McGuire, Ramsdell, & Athey, 2007). RASCAL evaluates releases from nuclear power plants, spent fuel storage pools and casks, fuel cycle facilities, and radioactive handling facilities. RASCAL is compiled for a Windows environment using a graphical user interface.

2.6.5 GENII

GENII (Napier, 2012) is a computer code developed for the Environmental Protection Agency (EPA) at Pacific Northwest National Laboratory (PNNL) to incorporate state-of-the-art internal and external dosimetry models into updated versions of environmental pathway analysis models. The primary purpose of GENII is for calculating radiation doses for individuals or populations following chronic or acute releases. This is accomplished by modeling radionuclide transport via air, water, or biological activity. Air transport options include puff or plume models, calculation of effective stack height, plume rise from buoyant or atmospheric releases, and building wake effects. GENII is compiled for a Windows environment, and is run through a user interface using the Framework for Risk Analysis in Multimedia Environmental Systems (FRAMES).

2.7 Risk Informed Safety Margin Characterization (RISMC)

As part of the Department of Energy's (DOE) Light Water Reactor Sustainability Program (LWRSP), a team at Idaho National Lab (INL) is working on Risk-Informed Safety Margin Characterization (RISMC) (Youngblood et al., 2010) to evaluate long term changes in plant safety margins, with special emphasis on the integrated treatment of aleatory and epistemic uncertainty.

The overarching objectives of RISMC are to support plant life-extension decision-making by providing a state of knowledge characterization of safety margins in key Systems, Structures, and Components (SSCs) (Fleming et al., 2010), and to develop and apply advanced analysis methods to predict and manage plant safety margins as an essential part of operational and regulatory decision making for commercial NPPs (Hess et al., 2009).

The RISMC team uses the terms “load” and “capacity” to refer respectively to the magnitude and nature of the physical challenge imposed on particular SSCs and the capability of the SSC to withstand a given challenge. Margin is explicitly related to the probability that a load applied to an item exceeds the capacity of that item to withstand the load without failing. However, it is not enough to characterize margin as a distance between two mean values. Load and capacity are both distributions with associated uncertainties, which yield an overlap in the low probability/high distribution consequence tails.

The term “risk-informed” has several meanings depending on its intended use. The NRC describes risk informed regulation as, “An approach to regulation… which incorporates an assessment of safety significance or relative risk. This approach ensures that the regulatory burden imposed by an individual regulation or process is appropriate to its importance in protecting the health and safety of the public and the environment” (USNRC 2011a). Because considering only design basis accidents and single failure events leads to over-investment in some areas, and under-investment in others, a risk-informed approach considers the margins in the context of the full scenario set (i.e., including non-design basis accidents) (Youngblood et al., 2010).

2.7.1 The Determinator

In 2011, as under the umbrella of the RISMC goals, researchers at Idaho National Lab began work on a concept called the "Determinator." This was originally designed to be a component of the next generation safety analysis code, RELAP7 (also known as R7). While its applicability has evolved over the last few years, the concepts that make up the Determinator are still novel. Aside from simply simulating physical plant properties, a next generation safety analysis code should also have the ability to simulate human (operator) behavior. The goal of the Determinator is to represent plant procedures, guidelines, and the plant operator priorities that inform goal-seeking behavior within the constraints imposed by the procedures (Nourgaliev et al., 2011). In other words, the Determinator uses artificial intelligence engines (similar to those employed by the by the video game industry) to put a simulated operator in the simulated plant.

A future nuclear power plant simulation code will encompass everything inside a Gaussian surface that includes the plant, the license commitments, procedures, PRA, and thermal fluids codes. While these areas typically have sockets where one would insert models of particular actions one finds relevant to the situation, the Derminator ideally will be capable of simulating a more complete and comprehensive set of actions than one's own imagination may be capable of. For instance, the operators at Fukushima shut off the isolation condensers in some of the units during the accident. The Determinator could conceivably have predicted this because written procedures outline this action in the event of an overcooling transient (Youngblood, 2011).

2.8 Numerical Weather Prediction (NWP)

The tools from Numerical Weather Prediction (NWP) are essential for the construction of the RAPSS decision engine. In NWP, the earth is sampled through a combination of weather balloons, ocean buoys, and satellite images among others. These measurements are fed into NWP models that project a short time into the future by use of ensemble modeling. These forecasts are continuously updated by the use of data assimilation. RAPSS uses similar methodology, except the power plant takes the place of the earth as the system, and a severe accident code is used instead of NWP models.

2.8.1 Ensemble Forecasting

Because the atmosphere is a chaotic system (Lorenz 1963), small errors in initial conditions of any NWP model will amplify as the forecast evolves. Since all atmospheric measurements inherently contain some error, an infinite spectrum of plausible initial conditions and hence possible (often drastically) different futures exist. Running a single NWP model is insufficient because it only shows one of many possible futures.

A common way to account for this is through the use of Ensemble Forecasting (EF) (Du et al., 1997; Ghile & Schulze, 2009; Pozo et al., 2010; Roebber et al., 2009; Snyder & Zhang, 2003; Stensurd et al., 2000; etc.), which involves running multiple forecasts simultaneously from equally probable initial conditions, physical parameterization, or numerical models, or a combination of two or more of the aforementioned methods. The output is a spectrum of possible forecasts that gives a much better picture of possible futures than a single, highly detailed run combined with uncertainty.

Roebber et al. (2004) point out a number of important questions to be addressed when utilizing EF tools; the sub-bullets were added later.

- What is the best way to construct an ensemble?
    - How are the initial conditions going to be varied? What else will be varied?
- What is the relative role of initial conditions versus model formulation in constructing ensembles?
    - Because the ensemble is generally made up of lower-resolution models with more simplified physics packages, the initial conditions may not provide enough information to obtain any sharpness in the probability distribution.
- For what temporal scales are ensembles best suited?
    - Typically, ensemble forecasts have been used for medium range and longer time scales. Do the models have sufficient resolution on the proposed temporal scale?
- What is the best way to produce probabilistic forecasts from the ensemble output?
    - What are the model biases? Do the ensemble members need to be weighted using various statistical processing techniques?
- What is the source of the underdispersion of ensemble system, and how can this best be corrected?
    - Generally, it is difficult to encompass all of reality, especially for low probability, high impact scenarios. How does one account for this?

A primary advantage of ensembles is that they are inherently probabilistic and hence can express uncertainty directly. Therefore, users can make informed decisions based on these probabilities and their own cost/loss ratios (Roebber et al., 2004).

2.8.2 Data Assimilation

Assimilation of new data into models is an essential feature of NWP, giving models the ability to update predictions in real time. Data assimilation, according to Mackenzie (2003) is the glue that binds raw data with the physics-based equations that go into computer weather models. The most basic form of mathematical updating is derived from Bayesian networks and serves as the foundation for more recent data assimilation techniques, such as Kalman filters.

2.8.3 Bayesian Networks

Bayesian networks constitute a class of probabilistic models for modeling logic and dependency among variables representing a system. According to Ayyub (2003), Bayesian networks consist of the following:

- A set of variables;
- A graphical structure connecting the variables; and
- A set of conditional distributions.

Bayesian networks are commonly represented graphically consisting of a set of nodes and arcs (see Figure 2.8). The nodes represent the variables and the arcs represent the conditional dependencies in the model. If there is no arc between nodes, it implies the variables are conditionally independent.

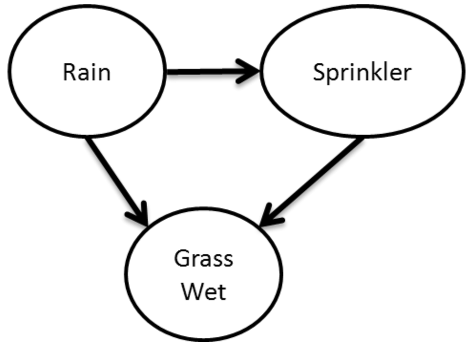


Figure 2.8 A Simple Bayesian network

For example, say that there are two reasons for the grass to be wet. Either it could be raining, or the sprinkler is on. Suppose further that rain influences how much the sprinkler is used, namely, that the sprinkler is used less when it is raining. Models such as this can be used to answer questions such as, "Given that the grass is wet, what is the probability it is raining?"

In mathematical form, Bayes Theorem (Bayes, 1763) states:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} \quad (2.20)$$

where P(A) and P(B) are the probabilities of events A and occurring, respectively. P(A|B) is the conditional probability, or the probability of A occurring given that B has occurred, and P(B|A) is the probability of event B occurring given that A has occurred.

P(B) can be computed based on the compliment of P(A) in the following manner:

$$P(B) = P(A|B)P(B) + P(A|\bar{B})P(\bar{B}) \quad (2.21)$$

where $\bar{A}$ and $\bar{B}$ are the compliments of A and B respectively, also expressed as (1-A) and (1-B). Equation (2.20), then can then be rewritten as:

$$P(A|B) = \frac{P(B|A)P(A)}{P(A|B)P(B) + P(A|\overline{B})P(\overline{B})} \tag{2.22}$$

The "prior" probability in Equation (2.20), P(A), represents relevant prior knowledge, or beliefs originally encoded in the model. Application of other relevant information such as tests and observations generate the "posterior" probability, P(A|B); they reflect the levels of beliefs computed in light of the new evidence, illustrated below in Figure 2.9.

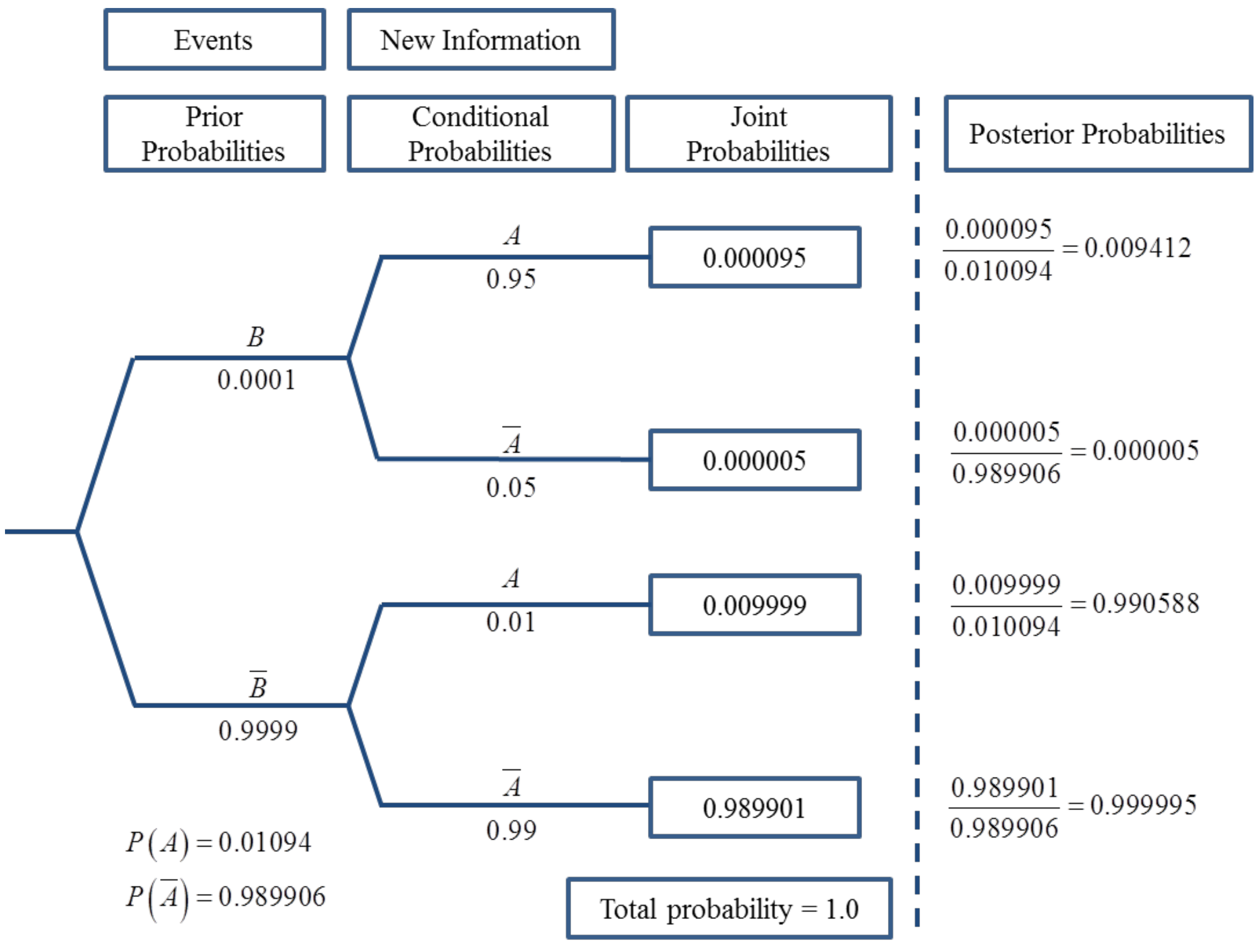


Figure 2.9 Probability tree representation of a Bayesian model; from Ayyub (2003)

The logic to follow one branch in Figure 2.9 is as follows. In this case, we will start with the prior probability of event B occurring as 0.0001. The prior probability of A occurring is 0.01094, shown in the lower left of Figure 2.9. However, if B does occur, then the probability of A occurring, P(A|B), is 0.95, totaling a joint probability of 0.00095, (P(B) *x* P(A|B)). The posterior probability is the updated probability of B occurring if A occurs, or P(B|A), and is calculated using of Equation (2.20) as follows:

$$P(B \mid A) = \frac{P(A \mid B)P(B)}{P(A)} = \frac{(0.95)(0.0001)}{(0.01094)} = 0.009412 \qquad (2.23)$$

Bayesian networks allow for one to update probability calculations as new data arrives.

2.8.4 Kalman filters

A Recursive Bayesian Estimator, also known as a Bayes filter is a general probabilistic approach for estimating an unknown Probability Density Function (PDF) recursively using indirect, inaccurate, or uncertain incoming measurements and a mathematical process model. When the differential predication equations are linear, yielding Gaussian PDFs, the Bayes filter becomes the more widely used Kalman filter (Kalman, 1960). Kalman filters are commonly used in a variety of fields from robotics and computer graphics, to weather and economic modeling, and even to famously guiding the Apollo spacecrafts to the moon (Andreasen, 2008; Huntley & Miller, 2009; Mackenzie 2003; Strid & Walentin, 2008; Welch & Bishop).

The idea behind Kalman filters can be traced back to a simple statistics problem. As explained by Mackenzie (2003), given two measurements, x_1 and x_2, of an unknown variable, x, what combination of x_1 and x_2 gives the best estimate of x? The answer depends on how much uncertainty (expressed by the variances σ_1^2 and σ_2^2) one would expect in each of the measurements. The combination of x_1 and x_2 that yields the least variance, then, is expressed by Equation (2.24):

$$\hat{x} = \frac{\sigma_2^2}{\sigma_1^2 + \sigma_2^2} x_1 + \frac{\sigma_1^2}{\sigma_1^2 + \sigma_2^2} x_2 \tag{2.24}$$

If the variance of one measurement is infinite, then one would only use the other variable as the estimate. In most cases, however, when the variances are finite, Equation (2.24) shows the correct weights to assign to each variable.

In Kalman filtering, the first measurement, x_1, comes from sensor data at the current time, t_k, and the second “measurement,” x_2, is actually not a measurement but the last prediction, made at time t_{k-1}, of the model at time t_k. In real applications, these measurements (i.e., x_1 and x_2) are not numbers, but vectors with, in the case of NWP, millions of components. The variances, σ_1^2 and σ_2^2, are replaced by covariance matrices, which represent the uncertainty in the measurements and the uncertainty in the forecasting model, respectively. The groundbreaking discovery by Kalman (1960) was that when the weights attached to the data and the previous forecast are chosen in a way that minimizes the new forecast variance, the weights attached to all previous data are automatically optimized as well. It is this unique feature that allows users to update future predictions based only on the previous prediction, and current sensor data, completely eliminating the need for data older than t_{k-1}.

Kalman filters are limited by size and non-linearity. Size limitations in NWP comes from the immense state vectors (i.e., on the order of 75 million components) and the even larger covariance matrices (i.e., 75 million *x* 75 million). The user also needs the predictive computer model errors to be Gaussian. While measurement errors are typically Gaussian, prediction errors are only Gaussian if the differential equations used to make the predictions are linear. Unfortunately, weather pattern equations are notoriously non-linear in nature. As time passes, and many state observations are assimilated by the Kalman filter, the predictive distributions start to become more Gaussian, even if the initial distributions were clearly non-Gaussian.

Evensen (1994) proposed (later clarified by Burgers et al., 1998) a way around the size issue by use of an Ensemble Kalman Filter (EnKF). EnKFs represent the distribution of the system state using ensembles (a.k.a., random samples, or Monte Carlo methods) and replacing the enormous covariance matrix by the much smaller sample covariance of the ensemble.

EnKFs, however, still rely on the Gaussian assumption, though they are still used in practice for nonlinear differential predictors. The extended Kalmen filter (Ribeiro, 2004) attempted to correct for non-linearity by linearizing the prediction equations via Taylor expansions. However, for highly non-linear prediction equations, this too deteriorated as time progressed. Other work involved filters such as the Local-Local Ensemble Filter (LLEnsF) (Bengtsson et al., 2002). While the LLEnsF produced more accurate state estimates than the EnsKF when the forecast distributions were sufficiently non-Gaussian, the LLEnsF also to broke down after many cycles because it ignores spatial continuity (smoothness) between local state estimates (Bengtsson et al., 2003). Recently, however, Zhou et al. (2011) proposed a promising correction method for EnsKFs called the Normal-Score Ensemble Kalman Filter (NS-EnsKF) which transforms the original state vector into a new univariate Gaussian vector, performs the filtering using an EnsKF, and back transforms the vector ensuring state-vector components are preserved throughout.

2.9 Parallel Computing

Due to the large amount of computer power required to implement the above methodologies, parallel computing strategies must be utilized to make predictions with any resolution in an acceptable amount of time.

Ordinary computer programs run sequentially, meaning a program command is executed only if the former command is finished. It can be thought of as a single worker performing tasks step by step. If instead there was a whole team of workers, they would be analogous multiple processors working simultaneously. Knaus and Porzelius (2009) provide a useful analogy:

> "Imagine the building of a wall. A single worker places the bricks one after another. A team of workers can place several bricks at the same time. But there is a limit: the wall will not be done faster if adding a huge number of workers, there is a limit – in this case the amount of bricks per wall row (which is called 'scalability' of the problem, which basically describes the benefit of adding additional processors)."

2.9.1 Parallelism Vocabulary

The average computer user performs parallel computing every day, usually without knowledge of it. This is because most modern programs are written using *implicit parallelism*, meaning the system controls parallelism details such as task division or process communication. Matlab M-code is an example of a computation code that utilizes implicit parallelism (Burkardt & Cliff, 2009; Moler, 2007). While this is very convenient for the user, it oftentimes produces less-than-optimal parallel efficiency.

To increase the efficiency, one could utilize *explicit parallelism,* meaning the user controls the parallelism details (e.g., spawning of computation nodes, task division, synchronization of concurrent processes, etc.). Message Passing Interface (MPI) and OpenMP are examples of explicit parallel structures (see Section 2.9.3). While this can produce highly efficient codes, it is often times difficult for non-computer scientists (e.g., health physicists) to code.

Parallel computing makes use of a workstation *cluster* or a set of machines (called *cores* or *processors*[2]) that are interconnected and share resources as if they were one larger computer. The *master* in the cluster is the system that controls the cluster and the *slave*[3] (also called a *worker*) is a machine that performs computations and responds to the master's requests. *Threads* are the smallest units of processing that can be scheduled by the operating system. On a single processor, *multithreading* occurs when the processor switches between different threads fast enough that the user perceives simultaneous execution. However, when systems have multiple cores, the threads can actually run at the same time, which is generally called *multiprocessing* to distinguishing it from multithreading.

2.9.2 Task Division

Task division is divided into three categories: *brute force, task push,* and *task pull.* Brute force is commonly used for "embarrassingly parallel" problems, which is slang in the computation world for dividing the task into *n* subproblems and assigning one task to each slave. The glaring disadvantage of this technique is that the number of tasks must be less than or equal to the number of slaves. If the number of tasks are greater than the number of slaves, one could use task push, where the number of tasks is equally divided between slaves *a priori* from the master. The slaves loop and retrieve these tasks until there are no messages left to process. Task pull is commonly used when the number of

---

[2] While some refer to the "processor" the physical chip, possibly containing multiple cores, it is also used synonymously with the term "core", making its use ambiguous and context dependent.
[3] In Los Angeles, officials have asked that manufacturers, suppliers, and contractors stop using the terms "slave" and "master," saying they are unacceptable and offensive (CNN 2003).

tasks is much greater than the number of slaves. In this case, the tasks are only delivered to the slaves when they have completed their previous task. This is especially useful when the tasks are either not load-balanced, or the slaves have unequal computing power. Task pull cuts down on the wait time task push creates when one or more slaves complete their tasks while others are still working.

2.9.3 Application Programming Interfaces (APIs)

Similar to the way user interfaces facilitate interaction between humans and the computer, Application Programming Interfaces (APIs) provide a medium for communication between programs. APIs for parallel processing can be divided into two basic categories: *shared memory* and *distributed memory*. Shared memory offers a single memory space that may be simultaneously accessed by multiple programs or processors. Distributed memory allows each processor to have its own private memory, enabling computational tasks to operate solely on local data until remote data are required.

2.9.4 Open Multi-Processing (OpenMP)

One of the most common examples of parallel processing API using shared memory is OpenMP (Open Multi-Processing). OpenMP supports multi-platform multiprocessing in C/C++ and Fortran using open source compilers in many architectures (e.g., Unix, Solaris, MacOS X, Windows, etc.) and across a wide variety of platforms (e.g., laptops, desktops, super computers, etc.) (OpenMP website, 2011). It consists of a set of compiler directives, library routines, and environment variables that influence runtime behavior (Mandelli, 2011). For example, when using OpenMP, the section of code that is meant to run in parallel is marked with a preprocessor directive causing the threads to form before the section is executed (Schmidberger et al., 2009). OpenMP is straight-forward because the user does not dictate message passing between programs; instead, there is a set location that all programs look to for information.

2.9.5 Message Passing Interface (MPI)

The distributed memory parallel processing API of choice for applications running on large-scale clusters is Message Passing Interface (MPI) (Sur et al., 2006). MPI defines an environment in Fortran or C/C++ where programs can run in parallel and communicate with each other by passing messages (Snir, 1996). Simply speaking, MPI creates mail boxes for each program, called *queues*. Any program can put messages into another program's mailbox. When a program is ready to process a message, it receives a message from its own mailbox. Like OpenMP, MPI is also capable of running on a variety of platforms and architectures. However, unlike OpenMP, the processors need not be of similar architecture; the MPI implementation will automatically do any necessary data conversion and use the correct communications protocol (Snir et al., 2006).

2.10 R

R is a language and environment for statistical computing and graphics. It is a General Public License (GPL, commonly called GNU, a free software license) project, similar to the S language and environment, developed by Bell Laboratories. The software (in source code form) compiles and runs on a wide variety of UNIX platforms, as well as Windows and MacOS. While the R environment is most often used on its own, for computationally-intensive tasks, C/C++ and Fortran code can be linked and called at run time. Advanced users can even write C code to manipulate R objects directly (The R Project Webpage, 2011).

One major limitation of R is that regardless of the number of cores in the Central Processing Unit (CPU), R will only use one on a default build. According to *The R Journal (*Knaus et al., 2009):

> "R itself does not allow parallel execution. There are some existing solutions… However, these solutions require the user to setup and manage the cluster on his own and therefore deeper knowledge about cluster computing is needed. From our experience this is a barrier for lots of R users…"

The obvious solution to this problem, then, is to obtain a deeper understanding of cluster computing (see Section 2.9). Thankfully, R is highly extensible through the use of *packages*, which are libraries for specific functions or specific areas of study, frequently created by R users and distributed under suitable licenses (Schmidberger et al., 2009).

2.10.1 Parallel Computing in R

The Simple Network Of Workstations (SNOW) (Tierney et al., 2011) is an R package that provides a high-level interface for using a workstation cluster for parallel computing. It allows users to implement explicit parallelism without interfacing with C or Fortran. However, most tend to use a wrapper package for easier development: SNOWfall (Knaus, 2011). Aside from syntax, SNOW and SNOWfall differ because SNOWfall does not require the user to create and handle the R cluster object directly. Both SNOW and SNOWfall can be executed in sequential mode, making debugging easier when no cluster is present (Schmidberger et al., 2009). Using SNOW/SNOWfall, a variety of parallel APIs are possible including socket, Parallel Virtual Machine (PVM), and MPI.

While some view R as a simple tool for statistical modeling, the above discussion has shown that through the use of packages, it can be used to serve a variety of purposes.

It is precisely the simplicity of this program that makes it useful. As will be described in Section 2.12, oftentimes, simpler is better.

2.11 Decision Making

Decision making has historically focused on guessing the potential outcomes of the apparent choices at hand. With the realization of probabilistic phenomena, modern statistical theory was born out of the European Renaissance, circa 1600 to 1700. However, many important decision making concepts did not become widely regarded until much later with the publication of von Neumann and Morgenstern's text on game theory and economic behavior (von Neumann & Morgenstern, 1944), where Expected Utility Theory (EUT) was defined for the first time. In EUT, a utility function is meant to represent a decision maker's beliefs about the value of a particular attribute of a decision outcome. In EUTs most basic form, subjects are presented with "betting preferences" with regard to uncertain outcomes, or gambles. For example, suppose there is a gamble where the probability of receiving $100 is 1 in 50. The alternative is to receive nothing, with much higher odds. The expected utility of this gamble would be $2.00, suggesting that it is worthwhile to take this gamble. Later, Raiffa and Schlaifer (1968) introduced the concept of decision trees and Baysian analysis (see Section 2.8.3) to decision theory, which laid the foundation for the event trees of WASH-1400 (see Section 2.1.1). A PRA event tree can be seen as simply a decision tree without decisions. Today, decision trees are essential to formalized decision analysis in a variety of business and engineering fields (Clemen, 1997).

2.11.1 Decision Making in Nuclear Power Plants

Every day, numerous decisions are made at nuclear power plants. Since NPPs have both large capital and operational costs, even small, or routine decisions can lead to potentially large economic consequences. Until fairly recently (Horng, 2004; Pagani et al., 2004; Smith, 2002; Weil & Apostolakis, 2001) there has been little investigation into formal decision making at NPP; instead ad hoc, or informal decision making is mostly used. These take the form of three risk metrics advocated by the NRC: Significance Determination Process (SDP), the Generic Issue Program (GIP), and Risk-Informing Special Treatment Requirements.

The Significance Determination Process is a method of evaluating the significance of a previous plant condition that has since been corrected but could have threatened the plant's safety. The SDP is used to determine the change (or *delta*, Δ) in the CDF and LERF with and without the condition. Based on the Δ CDF and Δ LERF, the NRC determines the significance of the condition, which is color-coded as green, white, yellow, or red in increasing order of significance as illustrated below:

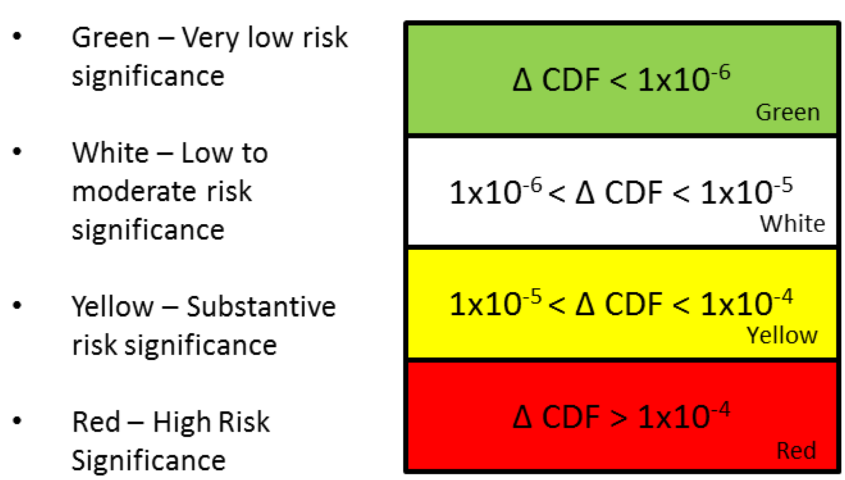


Figure 2.10 Color coding of the Significance Determination Process used by the NRC (Dwivedy et al., 2007)

The color of the finding identifies the severity of the condition and is used by the NRC to determine the scope and extent of future inspections, enforcement actions, and communications in order to help avoid similar future conditions in the plant (Dwivedy et al., 2007).

The NRC's Revised Oversight Process uses these risk metrics to determine an "action matrix" of outcomes for measures such as initiating events and mitigating systems. However, Smith (2002) pointed out that these action matrices raise questions about the consistency between categorical outcomes. For example, are two "white" outcomes in one quarter comparable to a "yellow" outcome in another category?

The NRC also employs a Generic Issues Program, which uses a slightly more quantitative approach to not only asses the CDF, but also an impact/value ratio (USNRC, 2011). The impact/value ratio, R, reflects the relationship between the risk reduction value expected and the associated cost impact in dollars, C:

$$R = C = NFTD \tag{2.25}$$

Where:

- N is the number of reactors involved;
- F is the accident frequency reduction (in event per reactor-year);
- T is the average remaining life (in years) of the affected plants, based on an original license period of 40 years; and
- D is the public dose from the radioactive material released from containment (in person-rem).

Using this methodology, it is possible to generate a conversion factor for the monetary worth of radiation exposure. The NRC currently recommends a conversion factor of $2000 per person-rem (USNRC 2004).

The Risk-Informing Special Treatment Requirements process is a fairly new methodology being used by the NRC to categorize certain SSCs as "safety-significant." In order for an SSC to be classified as safety-significant, according to 10CFR50.2 (USNRC 2003), it must be relied upon to remain functional during and following design basis events to assure:

- The integrity of the reactor coolant pressure boundary;
- The capability to shut down the reactor and maintain it in a safe shutdown condition; or
- The capability to prevent or mitigate the consequences of accidents which could result in potential offsite exposures comparable to the applicable guideline exposures set for in10CFR50.34, or 10CFR100.11.

The NRC further clarifies this by suggesting the safety-significant classification of an SSC can be determined using factors such as the Fussell-Vesely (F-V) importance and the Risk Achievement Worth (RAW) (Travers, 2000) for either the CDF or LERF. The F-V importance of an SSC is defined as the fractional decrease in total risk level when the plant feature is assumed perfectly reliable. The RAW of an SSC is the increase in risk if the feature is assumed to be failed at all times, and is expressed as the ratio of the risk with the event failed to the baseline risk level (NUCE Glossary, 2002). If the F-V or RAW of an SSC exhibit a larger value than the target:

- F-V > 0.005; or
- RAW > 2,

then the component is deemed safety significant.

In general, Smith (2002) points out several issues that all ad-hoc methods of decision making suffer from:

- Only focusing on a single metric (e.g., CDF) as a primary decision-driver;
- Lacking consideration of other decision alternatives outside the initial focus;
- Ignoring decision maker preferences for key attributes; and
- Not using methods such as “sanity checks” to question the validity of decision results.

Formal decision making, Smith (2002) argues, while still subjective, forces one to not only indicate what attribute is important, but also why it is important, and how much emphasis should be paid to the attribute as it relates to decision making.

As a possible future alternative, Smith (2002) developed a novel internet-based incident management (i.e., prior to core melt) advisory system tool for formalized decision making, simply named, “The Prototype.” The goal of The Prototype was to facilitate selection of a preferential decision alternative in response to an incident and provide technical justification for the decision. Interface with the Prototype involves the user manually entering data such as the type of incident (component or initiator related), the current reactor state, the time until the next outage, impacts to the plant operations through component degradations, as well as a variety of other incident specific information. The Prototype then analyzes the data using precompiled knowledge base containing a variety of potential decision alternatives such as shutting the plant down to fix the problem, repairing the problem at power, etc. Next, The Prototype constructs a decision model to evaluate a generic influence diagram/decision tree via a static, sequence based “roll-back” calculation (Clemen, 1997) and recommends decisions with the goal of assisting human judgment.

In a further evolution of Smith’s (2002) work, MIDAS (Minor Incident Decision Analysis Software) (Horng, 2004) is a standalone, window-driven GUI that adds further options, models, and a more modulator analysis structure. MIDAS uses Muli-Attribute Utility Theory (MAUT) (Keeney & Railla, 1993) with the intent of assisting decision making in a wide spectrum of minor incidents ranging from preventative maintenance to minor failure and repair of various components. While MIDAS is written specifically for nuclear power plant incident management, the intent was to lay the foundation for applications in other decision making situations. However, the decision making architecture of MIDAS was embedded in the source code and thus cannot be modified by general users.

Although The Prototype and MIDAS are not feasible in it their current forms for implementation in control rooms (due to the time involved in manual data entry) they do reflect the desire to shift the industry to more formalized decision making processes.

2.12 Risk and Perception of Risk

While some would argue that more information is always better for decision making, those in the psychology literature would disagree. They argue: to avoid biases, simpler is better, especially under high cognitive load.

2.12.1 Probability Aided Decision Making

Previous research has shown that people err when making judgments aided by probability information (Edwards, 1962; Ibrekk & Morgan, 1987; Kleinmuntz, 1990; Van Dijk & Zeelenberg, 2003). Many biases, such as the tendency to incorrectly estimate the probability of low likelihood, high consequence events have been identified among both experts and non-experts (Dawes, 1998; Tversky & Kahneman, 1974). This may surprise some who believe that experts are immune to this bias. For example, physicians who have considerable knowledge of incidence rates of diseases have been shown to reliably overestimate the annual number of deaths due to rare conditions such as Botulism (Christensen-Szalanski et al., 1983; Lichtenstein et al., 1978)

Cognitive resource constraints from time pressure or cognitive load have also been shown to further hinder people's ability to process decision making information (Ariely & Zakay, 2001; Edland & Svenson, 1993; Snyder et al., 2010). Gerhardt et al. (2011) explained that because risk is the outcome of (at least) two interacting systems (i.e., emotional/instinctive, and cognitive), when cognitive load is increased, the brain more actively relies on emotional responses, which leads to an increase in risk aversion tendencies. Another consequence of a cognitive load increase is a decrease in self-control. For example, Fudenberg & Levine (2009) showed that when participants were given a choice between cake and fruit salad for dessert, an increase in cognitive load led to a significant increase in cake choice. This is not surprising when one considers the cranial dual-system approach described above. Cake most likely generates a greater emotional response. When the cognitive brain is busy, emotions can take over without the interference of cognitive thoughts. This is especially evident in harrowing personal stories of the Chernobyl disaster in the Chernobyl Notebook (Medvedev, 1989), where reactor crew chief, Alexander Akmov, believed the reactor was still intact (after the explosion) despite indisputable evidence to the contrary (i.e., reactor graphite and fuel lying around the building).

2.12.2 Optimizing the Presentation of Uncertainty for Decision Makers

Research on optimizing the presentation of uncertainty information to decision makers has primarily focused on presenting probability in different ways. For example, researchers have tried presenting probability information as color variations, verbal expressions, frequencies, odds, visual objects with varying degrees of degradation, and graphical presentations (Ibrekk & Morgan, 1987; Johnson & Slovic, 1995; Kirschenbaum & Arruda, 1994; Schapira et al., 2001; Schwarz & Howell, 1985; Wickens et al., 2000). While most methods have met equivocal success, certain trends have been identified. For instance, graphical displays of uncertainty are superior to verbal descriptions (Kirschenbaum & Arruda, 1994; Stone et al., 1997).

Simply displaying a probability and consequence is not sufficient to communicate the risk. Several psychological studies of anxiety show the relatively small role probability plays in anticipatory emotions (Loewenstein et al., 2001). One might assume that people would use probability to optimize outcomes; however, Slovic (1995) showed that no optimization principles of any sort lie behind even the simplest of human choices.

Because risk is also feeling, the probability of an event is not necessarily related to the moral dimension of acceptability (Drottz-Sjoberg & Persson, 1993; Loewenstein et al., 2001). One approach that hasn't gotten much attention outside of the psychology literature is, in conjunction with probability and consequence, facilitating an emotional reaction from the display. It has been shown (Dougherty, Gettys, & Thomas, 1997) that imagining these outcomes makes them appear more likely. Perhaps showing nuclear power plant shift supervisors and technical associates the spectrum of consequences from decisions will invoke a greater emotional response, than simply following procedures.

A recent psychology study at Oregon State University investigated a variety of uncertainty display methods and found that in general, simpler portrayals of probability work better than complex multivariate or familiar approaches (Snyder et al., 2010). And consistent with Ibrekk and Morgan (1987), background knowledge of statistics did not enhance performance.

While the bulk of this work is focused on detailed prediction of future events, it is not lost on the author of this work that it is useless without the proper communication method to the humans who will use it to make decisions. Biases naturally are generated by any display method, especially under high cognitive load. However, work can be done to correct for these biases, as long as they are identified before the decision take place.

3 RAPSS Philosophy

When building a structure as large as RAPSS, it is important to segregate the process into a series of important but achievable steps, that when completed in succession, incrementally progress the user closer to the goal. RAPSS was developed in this manner. As RAPSS should be primarily thought of as a sampling and simulating technique, independent of a system, the first iteration was to sample and simulate a system of non-linear differential equations. From there, the first physical system was chosen (i.e., RAPSS-STA for NPPs). The main requirement for the system was having access to a method of modeling with reasonable enough fidelity and speed to make useful predictions about the system. Once the prediction data were generated, data analysis techniques were developed and implemented to parse the useful and representative information out for communication to the user. After data and plots were generated, a user interface was developed to communicate important information to aid in decision making. A second application (RAPSS-EOC) was developed in a similar fashion to RAPSS-STA to demonstrate the generalizability of the methodology (see Section 9 for more on RAPSS-EOC).

3.1 Preliminary research

The equation chosen to demonstrate a prediction method for non-linear differential equations was borrowed from insect outbreak modeling of spruce budworms, shown in Equation (3.1).

$$\frac{dx}{dt} = r(t) \cdot x \cdot \left(1 - \frac{x}{k}\right) - \left(\frac{x^2}{1 + x^2}\right) \tag{3.1}$$

Where:

- $r(t)$ is the rate of bug population increase, proportional to the forest growth rate. As the leaves grow, more food is available for the bugs, which causes the bugs to grow more;
- x is the population of bugs at time t; and
- k is the carrying capacity of the forest, arbitrarily set at 30 for this exercise, and was held constant.

Equation (3.1) was chosen both for its simplicity, and ability to demonstrate bifurcations (a.k.a., tipping points) where small perturbations in initial conditions cause the bug population to either explode into an outbreak condition, or contract to the refuge state. The behavior of Equation (3.1) is best explained by setting the derivative equal to zero to find equilibrium points by identifying intersections between the two remaining functions, as illustrated in Figure 3.1.

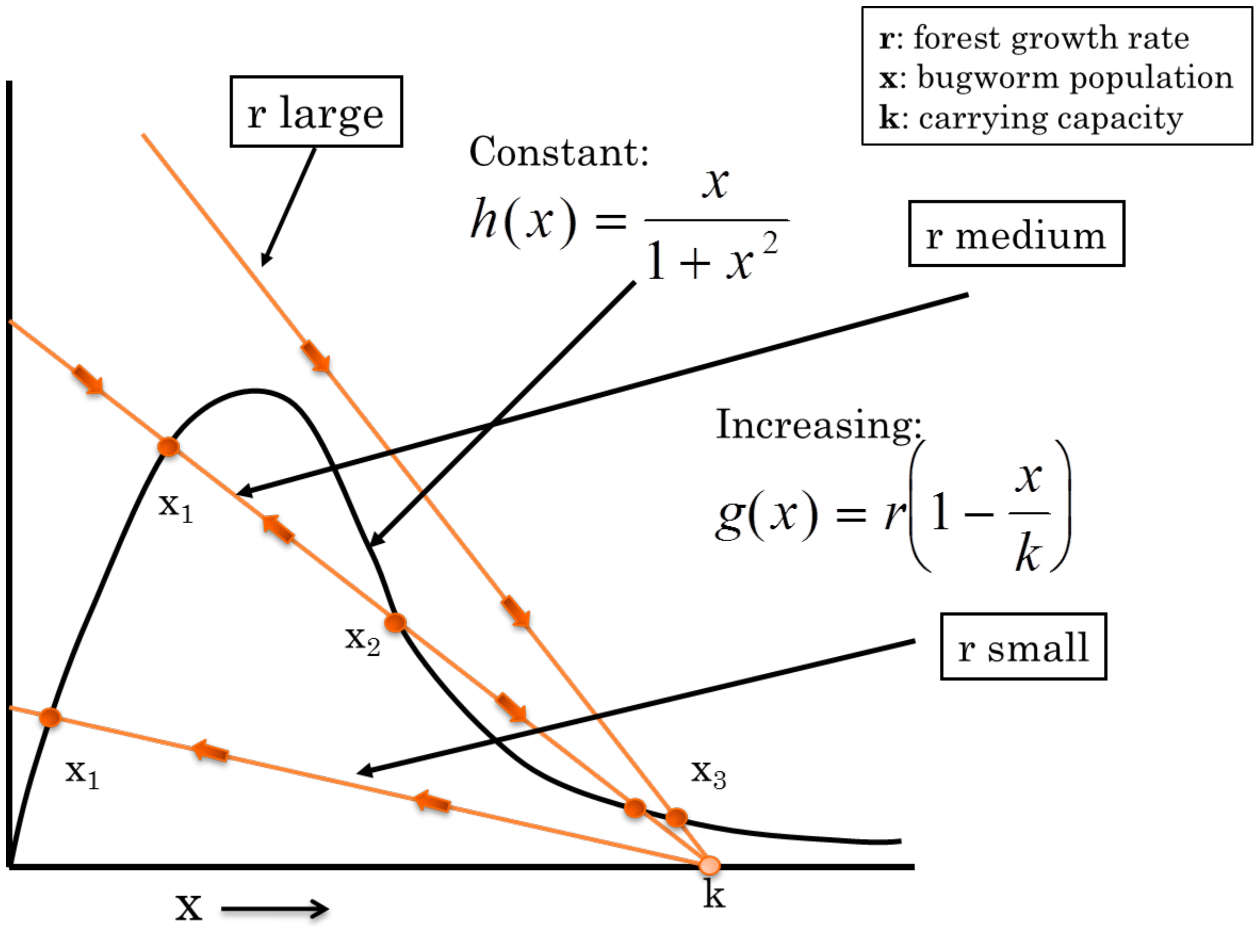


Figure 3.1 A visual representation of the behavior of Equation (3.1)

Figure 3.1 shows the two curves from Equation (3.1) after the derivative was set to zero, labeled *h(x)* and *g(x)*. The curve *h(x)* is only dependent on the bug population while *g(x)* is dependent on both the carrying capacity, *k*, and the bug/forest growth rate, *r*. The carrying capacity was fixed, so *r* was the only parameter besides the bug population, *x*, varying with time. By starting at the bottom of the figure, when *r* is small, any population of bugs tended towards the stable refuge state, x_1. As *r* increased, two other equilibrium points appeared: the stable outbreak condition of x_3, and the unstable bifurcation point of x_2. As r increased further, x_1 and x_2 disappeared and all populations, according to this model, tended towards outbreak conditions.

Equation (3.1) was coded in R (see Section 2.10), with randomly sampled initial conditions. Each run was therefore unique from the last, illustrating the population either passing a tipping point and increasing to outbreak conditions, or remaining at the refuge population. At each time step, the script fit a linear projection of the curve to predict when the curve would pass a "critical value," arbitrarily set at k-5. A typical graphical output of this demonstration is illustrated in Figure 3.2:

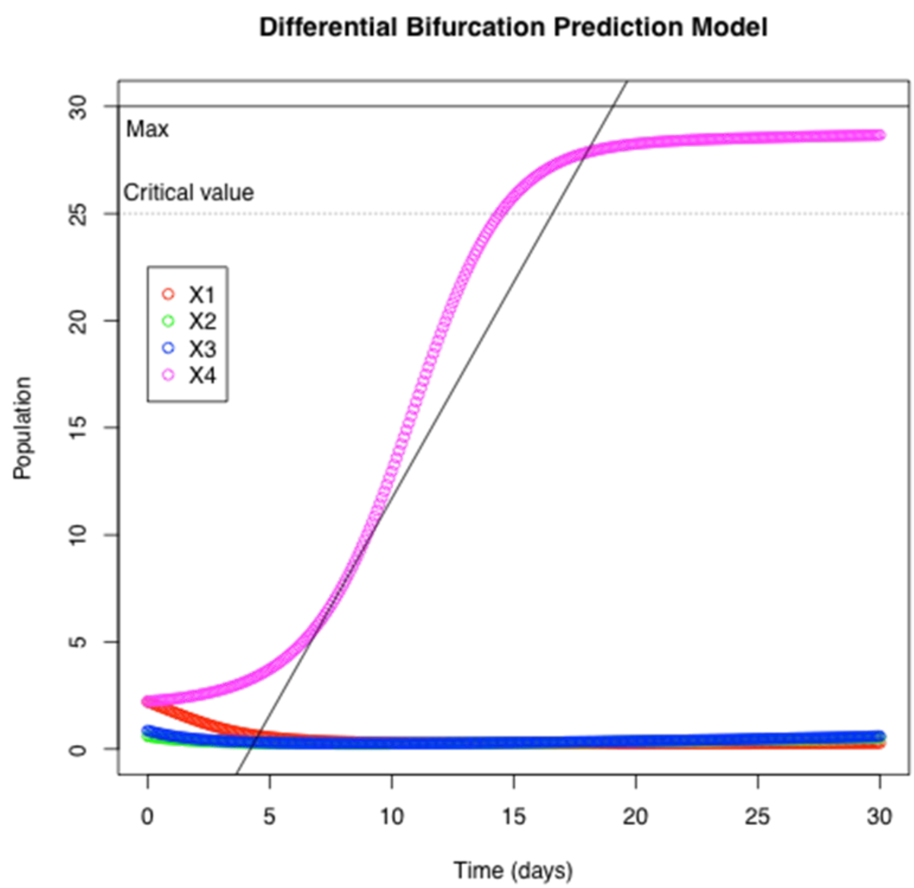


Figure 3.2 A typical graphical output of the first demonstration of the RAPSS philosophy

In this case, the non-linear differential equations functioned as the system, and the linear approximation at each point functioned as the predictive mechanism. At each timestep, the system was sampled, and projected ahead to determine if any thresholds were exceeded in the future. It was this simple structure that functioned as the foundation of the RAPSS methodology.

3.2 Implementation Path and Challenges

The first physical system that was chosen to apply the RAPSS architecture was for nuclear power plants (RAPSS-STA). In this case, the standard code for simulating these systems was RELAP5 (see Section 2.3.2). It should be noted that the version of RELAP5 that was integrated into RAPSS-STA was not designed for speed, or to handle any type of large changes in component conditions mid-simulation. These are tasks that reactor *simulators* are designed for. In this context, a reactor simulator is the underlying software in a reactor simulator control room (used for operator training). Softwares such as these handle operator actions, such as opening valves or starting pumps, by robustly translating the change in the model to the thermal fluids modeling software. RELAP5, on the other hand, is primarily used for detailed simulations where initial conditions are set and the user “lets it run”, with little opportunity for operator action modeling.

RELAP5 has always been the limiting factor in RAPSS-STA, primarily because the code was not originally designed for the type of implementation that RAPSS requires. However, as a proof of principle, RELAP5 acts as the perfect placeholder until more modern software is made available to the RAPSS team.

During a RAPSS-STA cycle, the software samples the system to define the initial conditions of the system, and runs ahead a given amount of time. Due to the limitations of RELAP5, RAPSS-STA is not capable of robustly simulating operator actions, especially over small time scales. When an operator acts, the physical properties of the model changes, and thus makes all previous predictions invalid. This is an especially challenging aspect of the RAPSS methodology that requires more robust reactor simulation software to address.

In the future, RAPSS-STA would be integrated directly into the physical controls of the facility. In the event of operator action, RAPSS-STA would stop the current prediction cycle, archive old predictions, update the physical properties of the model to match the new system properties, and begin a new ensemble of predictions. If the operator constantly “tweaked” system properties, even robust simulation software would have a difficult time predicting very far ahead before those predictions were invalidated by operator action. That is, unless the operator action could be predicted in a reliable manner by coding procedures combined with sampled human reliability data. The name given to this type of modeling is the “Determinator” (See Section 2.7.1).

The second system that the RAPSS architecture was applied to was for modeling a release of radioactive material outside of a nuclear power plant, spent fuel storage pools/casks, fuel cycle facilities, or radioactive handling facilities. In each of these cases an emergency operations center would be set up to monitor the release of material. RAPSS-EOC is intended to be used in this setting to give the staff of the emergency operations center an idea of where the current state of contamination plume is, and where it could be in the near future.

Because the system was inherently different than a nuclear power plant, some of the limitation of RAPSS-STA did not transfer to RAPSS-EOC. For instance, operator actions in RAPSS-EOC do not change the model, as human actions do not influence near-term meteorological conditions. Therefore, updating the physical properties of the model is only necessary for changing atmospheric conditions, which are predicted probabilistically from current and historic meteorological conditions data.

4 RAPSS-STA Facility Models

During the construction of RAPSS-STA, two primary RELAP5 models were used. The Cook model and the MASLWR (Multi Application Small Light Water Reactor) model. A description of each follows.

4.1 The Cook Model

For initial design of the RAPSS-STA system, an in-house RELAP5 model, written by Thomas Riley, of a generic pressurized water reactor was used (see Figure 4.1). This model was based on the Donald C. Cook Nuclear Generating System near Bridgman, Michigan. The RAPSS-STA system ran the model from a set of initial conditions under a variety of conditions looking for scenarios that could compromise the safety of the plant.

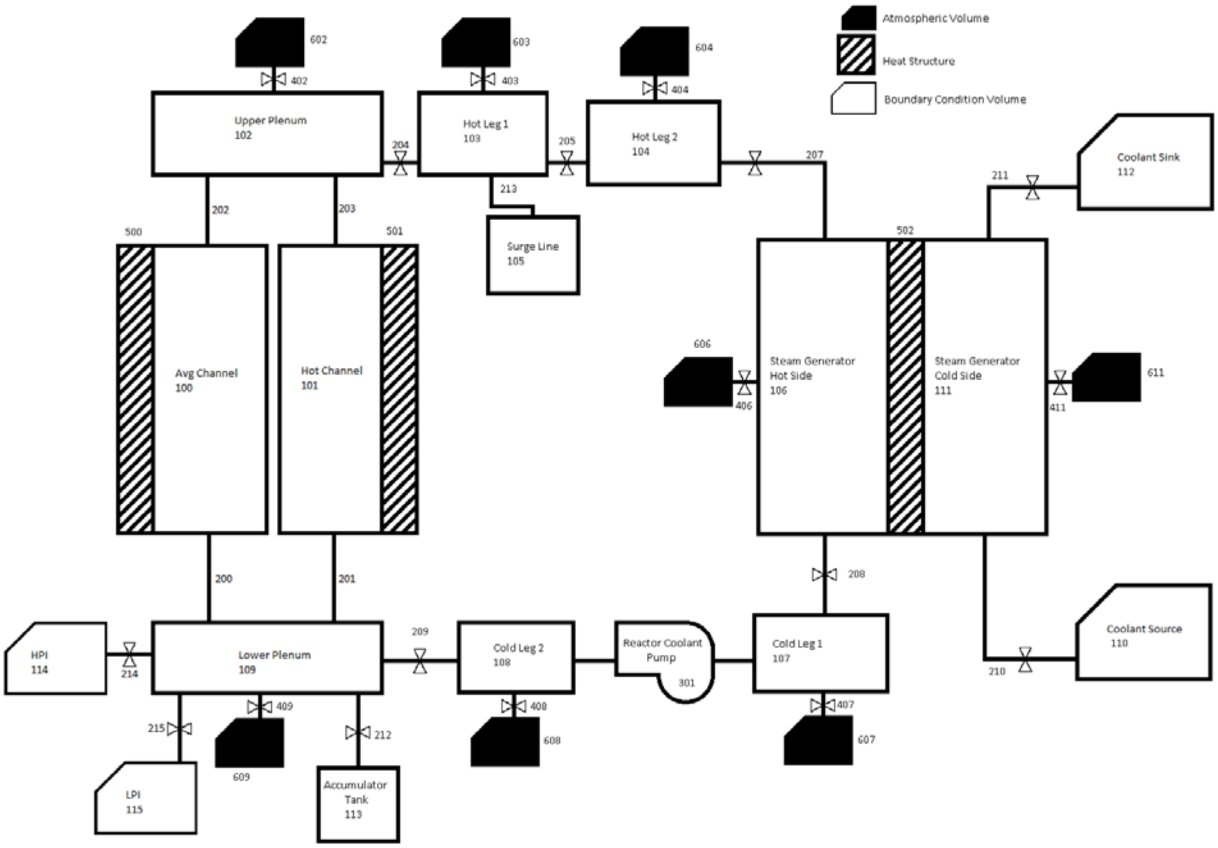


Figure 4.1 Schematic of the R5 Cook model used for the first-generation RAPSS-STA architecture

The model was built as a four leg plant, with a reactor coolant pump, steam generator, and connections to the reactor pressure vessel on each leg. For speed of simulation, the model was kept simple, to allow it to run significantly faster than real time. This was done so that multiple simulations of various conditions could be run simultaneously, producing a variety of predictions for scenarios before they happen. The model produced data for temperature and pressure of several key components of the plant that the RAPSS-STA system examined as part of the basis for its predictions.

To break the model in various ways, a number of valves were included to enable the RAPSS-STA system to disconnect various components from each other. Under normal circumstances, valves connecting functioning components to each other were set to open and valves connecting functioning components to atmospheric conditions were closed. To simulate a loss of coolant accident, valves could be opened to vent coolant into containment. Additionally, these “breaker valves” could be resized to allow for leaks of all sizes to occur. Similarly, to simulate a loss of flow accident, normally open valves could be partially or fully shut to restrict the coolant flow through a leg of the plant. To simulate Emergency core cooling system (ECCS) failures, ECCS subsystems could be shut by setting the connecting valves to the core to stay closed, rather than opening at appropriate pressure levels.

RAPSS-STA is capable of simulating eleven flavors of transients and safety system failures for the Cook model. Each transient was given a code (e.g., HPI_F for high pressure injection failure) that was used in the fault tree (see Figure 4.2):

- High pressure injection failure (HPI_F);
- Low pressure injection failure (LPI_F);
- Blockage of accumulator tank to lower plenum connection (ACUM_F);
- Partial blockage of accumulator tank to lower plenum connection (ACUM_F1);
- Small break loss of coolant accident in eight locations (SB_LOCA) ;
- Traditional large break loss of coolant accident in eight locations (LB_LOCA);
- Double guillotine large break loss of coolant accident in two locations (LB_LOCA1);
- Partial double guillotine large break loss of coolant accident in two locations (LB_LOCA1);

- Loss of heat sink accident in two locations (LOHA);
- Loss of flow accident in five locations (LOFA);
- And partial loss of flow accident in five locations (LOFA1).

4.1.1 The Cook Plant Fault Tree

To determine which transients to activate, RAPSS-STA used a generic PWR fault tree (Figure 4.2). It consisted of a top event, "core damage," and an *and* gate separating the tree into two legs. The left leg consisted of safety systems such as high-pressure injection among others. The right side consisted of various common transients that could occur in multiple locations within the plant. If a transient occurred, and the safety systems operated normally, core damage would likely not occur; likewise, if the safety systems failed, but no transient occurred, the plant would also not be in danger. It is only when both transients occur and safety systems fail that the core is in risk of damage.

The tree was kept very generic, consisting of many *undeveloped events*, (rhombuses in Figure 4.2) such as "SB LOCA somewhere." They were left undeveloped at this stage of proof of principle to focus on the structure and communication to RAPSS-STA, rather than dwell on specifics in the fault tree. Probabilities for undeveloped events were fixed at arbitrary values for the same reason.

RAPSS-STA used the module, LiteFTA (see section 2.2.3), to determine the cut sets and probabilities from the fault tree. It used this information to determine which transient to simulate.

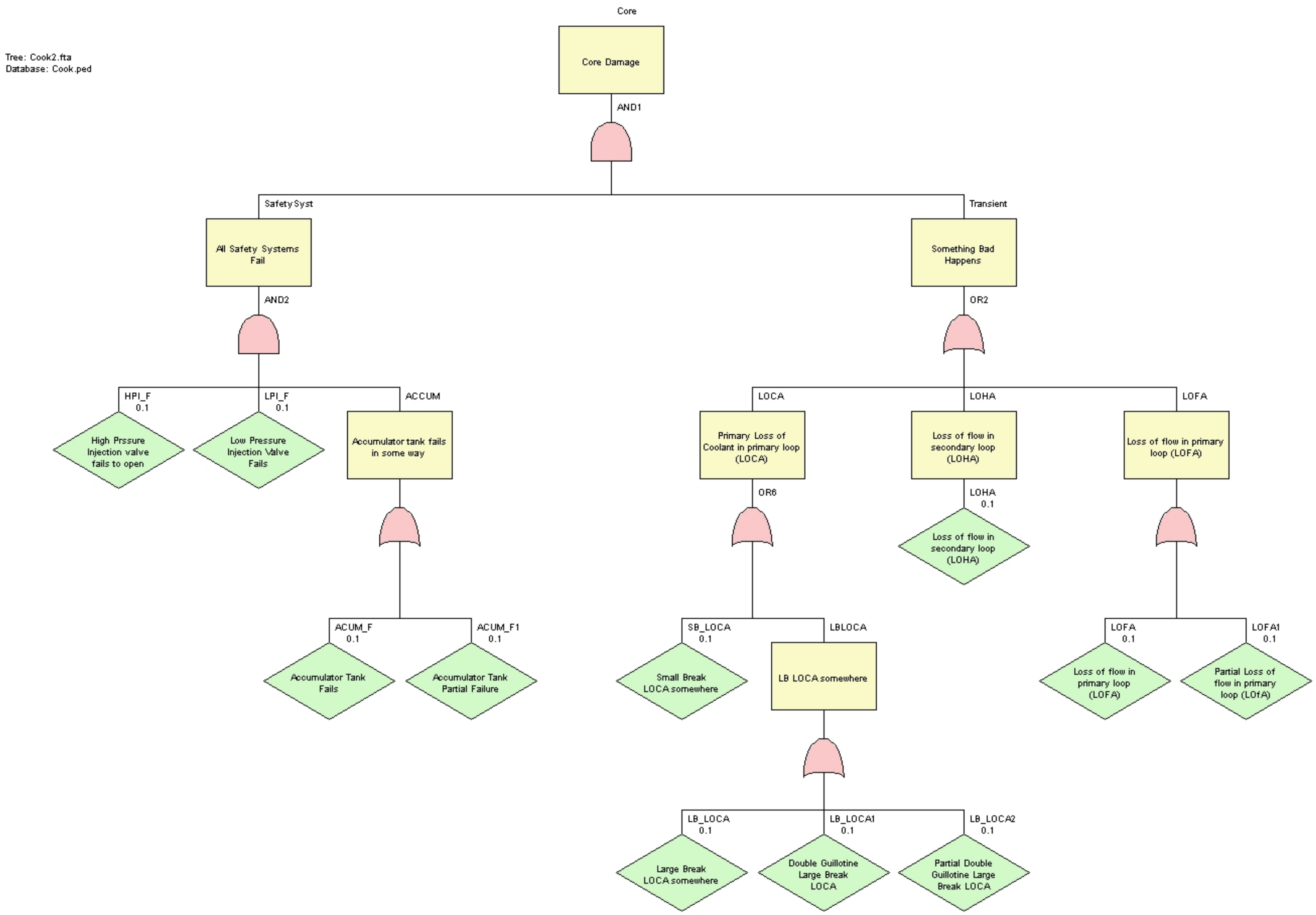


Figure 4.2 A generic fault tree built for the first-generation RAPSS-STA architecture

4.2 The MASLWR Facility

The Multi-Application Small Light Water Reactor (MASLWR) facility at Oregon State University was used as the system for RAPSS-STA calibration and experimentation. The MASLWR facility is an electrically-heated, scale model of a small modulator integral pressurized light water reactor, which relies on natural circulation during both steady-state and transient operation (See Figure 4.3). It is scaled at 1:3 length, 1:254.7 volume, and 1:1 time scale. It is also designed for full pressure (11.4 MPa) and full temperature (590 K) operation (Galvin & Bowser, 2010). MASLWR's safety systems are designed to operate passively, requiring no emergency cooling pumps or offsite power. The steam generators are located in the upper region of the vessel outside of the hot leg chimney and consist of sets of vertical helical tubes. The feedwater is fully vaporized inside the tubes resulting in superheated steam before entrance into the turbine generator.

The inherent safety structures in the MASLWR facility make it much more challenging to "break." A delicate balance was struck between simulating transients that lead to interesting results, but were not so severe that they caused the model to become unstable. Loss of coolant accidents (LOCAs) were significantly less likely because there are so few penetrations into the MASLWR reactor pressure vessel. However, if a significant hole between the reactor pressure vessel (RPV) and the pressurized containment developed, and the pressurized containment was also leaking, it would probably lead to a LOCA. Although even it that case, it may not lead to core melt because the pressurized containment itself is sitting in a tank of water which lives inside a secondary containment (Figure 4.3).

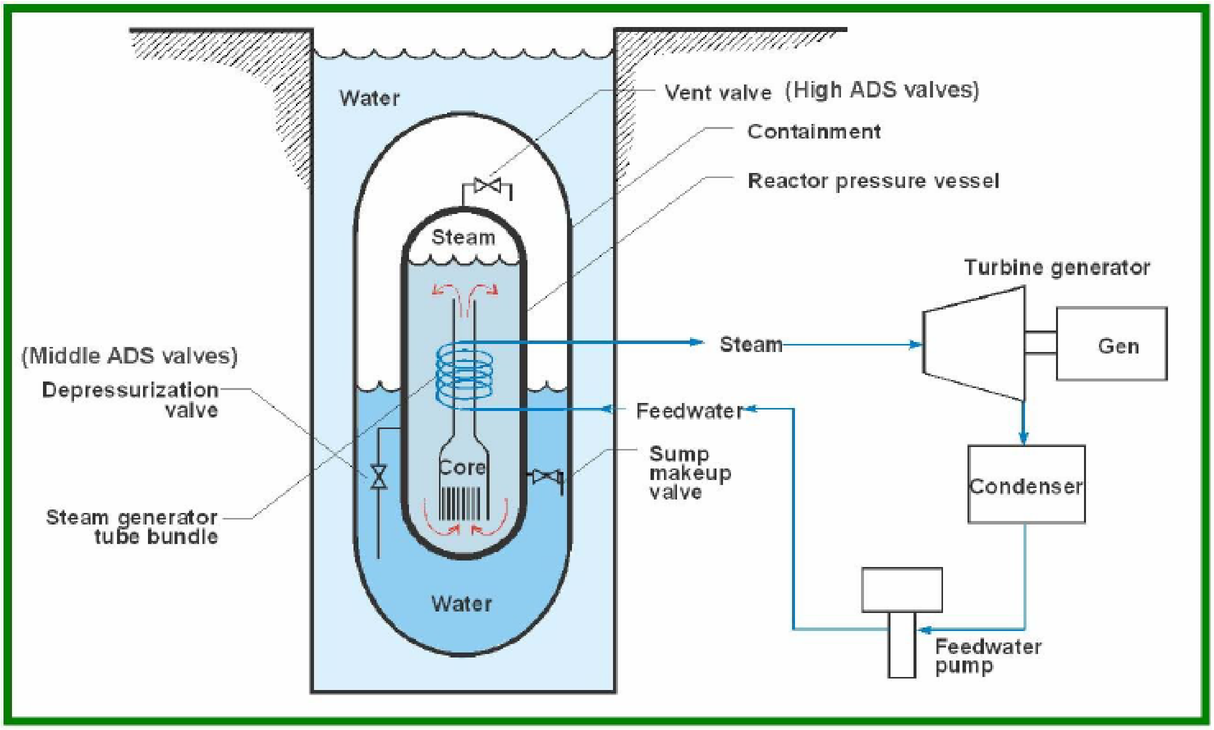


Figure 4.3 Conceptual design of the MASLWR test facility (Galvin & Bowser, 2010).

4.2.1 The MASLWR Real Time Simulator

The real time simulator was a simple program written to simulate the data output of the MASLWR facility. Because the modeling software (RELAP5) used in RAPSS-STA was not fast enough for real-time integration with the MASLWR facility, it was decided that writing software to behave as if the facility was running was the next best route. The software fetched measurements of temperature, pressure, and flow rates from the OSU MASLWR test facility output from the July 2011 IAEA experiments (Mai & Luo, 2011) as its data source to generate realistic initial conditions for the MASLWR R5 model. The output took the form of a tabbed separated values document.

From the start of the program, RAPSS-STA keeps track of the time elapsed to know where to sample from the MASLWR data. For example, if the “seed” run took 30 seconds, to begin the next cycle, RAPSS-STA samples from 30 seconds into the MASLWR experiment to set the current conditions for the RELAP5 model. Due to the rather complicated nature of timing a highly parallelized program, OpenMP's (see section 2.9.4) built in function, omp_get_wtime(), was used to produce accurate values of time elapsed during RAPSS-STA runs.

4.2.2 The MASLWR RELAP5 Model

A RELAP5 model of the MASLWR facility was used for simulation purposes. This model was originally built by Oregon State University facility and students: Dr. Brian Woods, Jordan Bowser, and recently for RAPSS-STA purposes by Thomas Riley.

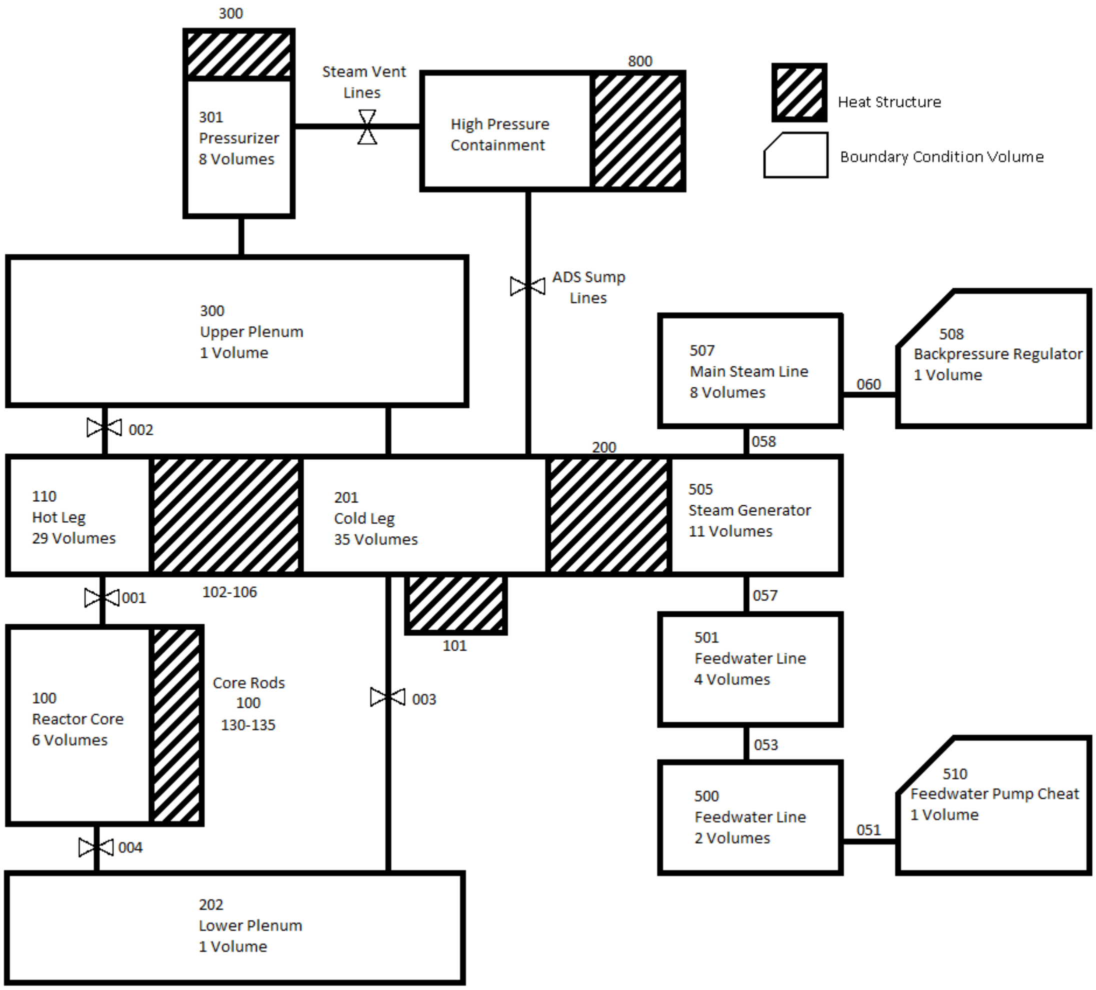


Figure 4.4 Schematic of the MASLWR RELAP5 model

To effectively simulate a variety of transients, a series of "Breaker Valves" were added to the RELAP5 model of the MASLWR facility, similar to the Cook model. These valves could either cut off flow between specific components to simulate flow blockages, or introduce flow between components that are not normally connected, to simulate leaks. An example of a flow blockage valve was the junction connecting the core components and the lower plenum. To simulate a partial flow blockage, this valve, which is open during normal operations, could be partially or completely closed. An example of a normally closed breaker valve is the component that is connecting the chimney hot leg to the downcomer cold leg. When closed, the two have no thermal hydraulic connectivity beyond the already present flow of heat across the temperature gradient in the steel; when opened, water can flow between the two freely, altering the thermal profile of the core coolant.

This specially modified MASLWR R5 model is capable of simulating ten flavors of transients. Each transient was given a code (e.g., VV_O for vent valve open) that is used in the fault tree (see Figure 4.5))

- Vent valve open (VV_O);
- Vent valve closed (VV_C);
- Hot channel chimney hole (HCC_F);
- Reactor pressure vessel (RPV) leak (RPV_F);
- Primary containment leak (CONT_F);
- Flow blockage (FLOW_B);
- Partial flow blockage (FLOW_B1);
- Sump water makeup failure (SWMup_F);
- Sump Open (SUMP_O); and
- Secondary loop failure (SECOND_F).

4.2.3 The MASLWR Model Fault Tree

The simplified fault tree illustrated in Figure 4.5 outlines the ten transients that were capable of being simulated in the MASLWR R5 model. The tree was kept simple to focus on application the RAPSS concept. Future work will incorporate more developed fault trees.

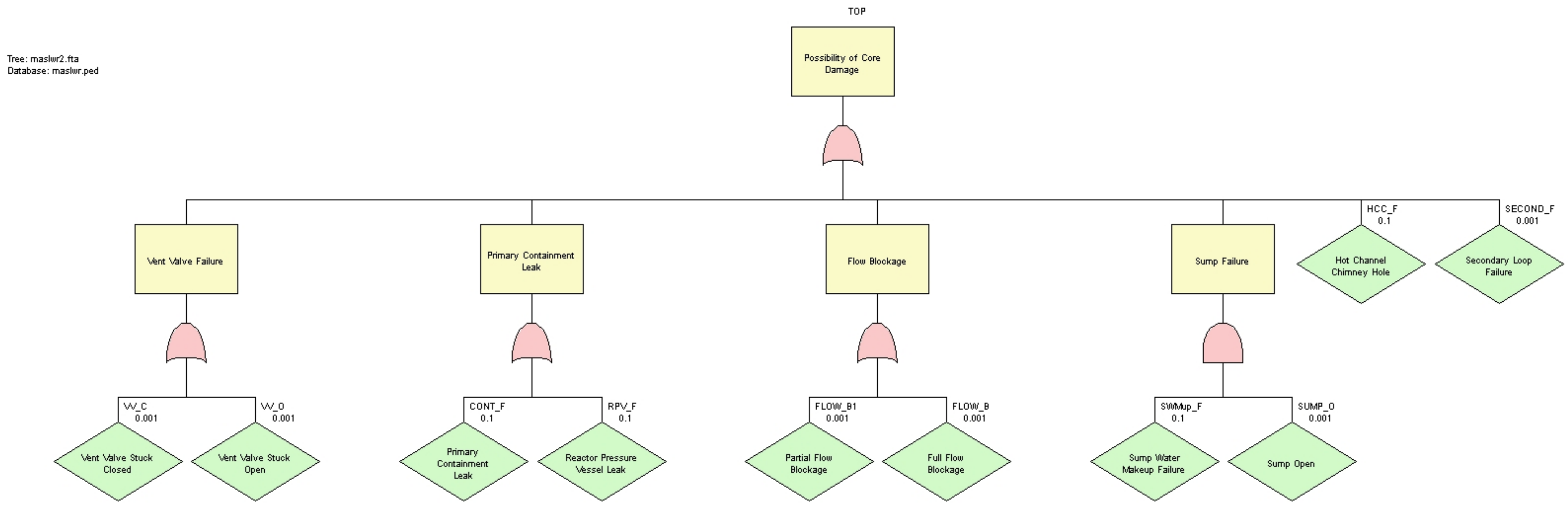


Figure 4.5 A generic fault tree built for the MASLWR facility for the first-generation RAPSS-STA architecture

5 RAPSS-STA Structure

The following are broad descriptions of the functions of each file used by RAPSS-STA. Source code is contained in Appendix A and detailed descriptions are contained in Appendix B. Readers with some familiarity of computer science are highly encouraged to explore the appendices for a more in depth understanding of the nuts and bolts of RAPSS.

5.1 RAPSmain.cpp

RAPSmain.cpp (see Appendices A.1 and B.1) was the file that "runs" RAPSS-STA. It mainly consisted of a user interface, and a function for reading user defined variables from the input file, RAPSinputFile(). These variables were then passed onto a single function, CycleR5(). The real meat of the program resided in CycleR5(), which was defined in CycleR5.h (see Appendices A.2 and B.2), one level below RAPSmain.cpp.

5.1.1 RAPSS Input file

The RAPSS input file consisted of three sections: R5 parameters; PCA, and MSA parameters; and RAPSS parameters. An example input file was included in Appendix A.12, and detailed explanations of each parameter are contained in Appendix B.12. Lines that begun with the comment character "*" were not read. Lines that begun with a three-digit number were read, allowing the user to pass information to RAPSS without recompiling.

5.2 CycleR5.h

CycleR5.h (see Appendices A.2 and B.2) was the first level below RAPSmain.cpp and consisted of a single function, CycleR5(),which was composed of a large while-statement and was the primary control mechanism for RAPSS-STA. Among many other things, this section allowed the user to specify how many cycles she wished to run before stopping. When fully implemented, RAPSS-STA would be continually running in the background, but for the purposes of this project, the user specifies how many cycles to run.

5.3 BloodAndGuts.h

As suggested by the title of this header file, this file contained the "blood and guts" of RAPSS-STA. It was basically a collection of various functions used in other parts of the program. This file can be considered the third layer below RAPSmain.cpp. Readers are highly encouraged to read Appendices A.3 and B.3 for more information.

5.4 OrganizeR5Output.h

OrganizeR5Output.h was written to search a RELAP5 output for state variable time series information and write to a .csv file. It entailed some rather complex organizational structures and is included in Appendices A.4 and B.4.

.

6 Data Processing

Each run of RELAP5 yields enormous amounts of data. Combine that with many, many continuously cycling simultaneous parallel runs, and the amount of data quickly becomes overwhelming. Further analysis and manipulation was necessary for display to the senior operators. RAPSS performs several tasks to boil down the monstrous amount of data into simple charts that a senior operator under high cognitive load can read and understand.

OrganizeR5Output.h (see Appendices A.4 and B.4) is a crucial module of RAPSS-STA that transformed the very long-winded R5 output file into a concise, .csv file. Since it was difficult to tell one run from another if no thresholds were tripped, these scenarios were passed for clustering using the mean shift algorithm (see Section 2.5.3). Due to the computationally intensive nature of MSA, principle component analysis (see Section 2.5.1) was first performed to reduce its dimensionality. This sped up the mean shift algorithm immensely (for benchmarking results see Table 6.8, Section 6.3).

6.1 Output from a single RAPSS-STA cycle

For a given RAPSS-STA cycle an R5 output (.p) file for each thread was genearted. Unfortunately, these outputs were rather challenging to digest, as they were 10,000 lines or more, and often contained state variable time series information parsed into many *sections*, which were further parsed into many *sets*. In this context, a *set* refers to a grouping of state variables, always beginning with time, up to 10 (including time) columns across, and up to 50 time steps (rows) long. A *section* refers to a grouping of sets, which represent the entirety of state variables for the same 50 time steps. Table 6.1 displays a single section from an R5 output, composed of two sets.

Table 6.1 Example R5 output, showing one set, composed of two sections.

```
0---Restart no.    1274 written, block no.     3, at time=     45.0272    ---
1 time        p            p            tempf        tempf        voidg        voidg        velgj        velfj        httemp
  (sec)        100010000    101010000    100010000    101010000    100010000    101010000    209000000    209000000    5000001  1
               (Pa)         (Pa)         (K)          (K)                                    (m/sec)      (m/sec)      (K)

   0.00000      1.54100E+07  1.54100E+07   577.60       577.60       0.0000       0.0000       0.0000       0.0000       581.83
   1.02723      1.53191E+07  1.53196E+07   538.69       537.01       0.0000       0.0000       3.0466       3.0466       578.86
   2.02723      1.52711E+07  1.52714E+07   534.90       534.18       0.0000       0.0000       1.5288       1.5288       572.98
        ...     ...     ...     ...     ...
   47.0272      1.31546E+07  1.31546E+07   587.22       590.39       0.0000       6.06593E-03  0.44576      0.44576      593.24
   48.0272      1.31074E+07  1.31073E+07   587.35       590.17       0.0000       6.30375E-03  0.46531      0.46531      593.74
   49.0272      1.30601E+07  1.30601E+07   587.48       589.97       0.0000       6.58475E-03  0.48352      0.48352      594.23
1 time        httemp
  (sec)        5010001  1
               (K)

   0.00000      726.05
   1.02723      723.32
   2.02723      719.22
        ...     ...     ...
   47.0272      750.24
   48.0272      750.00
   49.0272      749.76
1  RELAP5/3.3gl          Reactor Loss Of Coolant Analysis Program
```

Sections are often separated by thousands of lines of R5 output. OrganizeR5Output() (see Appendix A.4 for source code) was written to search a RELAP5 output for state variable time series information and write to a .csv file. A detailed description of the organizational algorithms is covered in Appendix B.4.

6.2 Organizational structure of PCA and MSA

An illustrative example will help by breaking down the organizational steps and using a small number of time steps, threads, and state variables. In this hypothetical RAPSS-STA simulation, R5 was run on three threads, with four time steps, and four state variables. The raw data would look similar to Table 6.2:

Table 6.2 Raw R5 example data for a 3x4x4 R5 run

Time (Scenario 1)	**Pressure**	**Temperature**	**Liq/Vap**	**Velocity**
1Sc1	P1	T1	L1	V1
2Sc1	P2	T2	L2	V2
3Sc1	P3	T3	L3	V3
4Sc1	P4	T4	L4	V4
Time (Scenario 2)	**Pressure**	**Temperature**	**Liq/Vap**	**Velocity**
1Sc2	P1	T1	L1	V1
2Sc2	P2	T2	L2	V2
3Sc2	P3	T3	L3	V3
4Sc2	P4	T4	L4	V4
Time (Scenario 2)	**Pressure**	**Temperature**	**Liq/Vap**	**Velocity**
1Sc3	P1	T1	L1	V1
2Sc3	P2	T2	L2	V2
3Sc3	P3	T3	L3	V3
4Sc3	P4	T4	L4	V4

If PCA was performed on each scenario individually, one would obtain an eigenvector matrix for each scenario. The inverse eigenvector matrix is used to transform the data back into physically meaningful numbers. If there was one eigenvector matrix per scenario, the inverse eigenvector matrices would be useless after clustering the scenarios because the clustered scenarios would not be the same as the unclustered. For this reason, the data was reorganized to obtain the same eigenvector matrix for every scenario. The example data would look similar to Table 6.3.

Table 6.3 Example data reorganized for PCA

Time	Pressure	Temperature	Liq/Vap	Velocity
1Sc1	P1	T1	L1	V1
1Sc2	P1	T1	L1	V1
1Sc3	P1	T1	L1	V1
2Sc1	P2	T2	L2	V2
2Sc2	P2	T2	L2	V2
2Sc3	P2	T2	L2	V2
3Sc1	P3	T3	L3	V3
3Sc2	P3	T3	L3	V3
3Sc3	P3	T3	L3	V3
4Sc1	P4	T4	L4	V4
4Sc2	P4	T4	L4	V4
4Sc3	P4	T4	L4	V4

The purpose of PCA was to reduce the number of state variables by looking for linear correlations among the data. At this point, the data would be broken into linear approximation intervals (see Section 6.4.1 for details), but for the sake of simplicity, this step was omitted in this example. After PCA, without any trimming, the user is left with one principal component per state variable, organized by the amount of variance each represents, resembling Table 6.4,

Table 6.4 Principal components for example problem data

Time	PC1	PC2	PC3	PC4
1Sc1	PC1(s1t1)	PC2(s1t1)	PC3(s1t1)	PC4(s1t1)
1Sc2	PC1(s2t1)	PC2(s2t1)	PC3(s2t1)	PC4(s2t1)
1Sc3	PC1(s3t1)	PC2(s3t1)	PC3(s3t1)	PC4(s3t1)
2Sc1	PC1(s1t2)	PC2(s1t2)	PC3(s1t2)	PC4(s1t2)
2Sc2	PC1(s2t2)	PC2(s2t2)	PC3(s2t2)	PC4(s2t2)
2Sc3	PC1(s3t2)	PC2(s3t2)	PC3(s3t2)	PC4(s3t2)
3Sc1	PC1(s1t3)	PC2(s1t3)	PC3(s1t3)	PC4(s1t3)
3Sc2	PC1(s2t3)	PC2(s2t3)	PC3(s2t3)	PC4(s2t3)
3Sc3	PC1(s3t3)	PC2(s3t3)	PC3(s3t3)	PC4(s3t3)
4Sc1	PC1(s1t4)	PC2(s1t4)	PC3(s1t4)	PC4(s1t4)
4Sc2	PC1(s2t4)	PC2(s2t4)	PC3(s2t4)	PC4(s2t4)
4Sc3	PC1(s3t4)	PC2(s3t4)	PC3(s3t4)	PC4(s3t4)
% Variance	**70%**	**87%**	**95%**	**100%**

The amount of variance to trim is the user's judgment call. For the purpose of this demonstration, one principal component will be trimmed, and 95% of the variability will be retained, as shown in Table 6.5.

Table 6.5 Three principal components for the example problem data

Time	PC1	PC2	PC3
1Sc1	PC1(s1t1)	PC2(s1t1)	PC3(s1t1)
1Sc2	PC1(s2t1)	PC2(s2t1)	PC3(s2t1)
1Sc3	PC1(s3t1)	PC2(s3t1)	PC3(s3t1)
2Sc1	PC1(s1t2)	PC2(s1t2)	PC3(s1t2)
2Sc2	PC1(s2t2)	PC2(s2t2)	PC3(s2t2)
2Sc3	PC1(s3t2)	PC2(s3t2)	PC3(s3t2)
3Sc1	PC1(s1t3)	PC2(s1t3)	PC3(s1t3)
3Sc2	PC1(s2t3)	PC2(s2t3)	PC3(s2t3)
3Sc3	PC1(s3t3)	PC2(s3t3)	PC3(s3t3)
4Sc1	PC1(s1t4)	PC2(s1t4)	PC3(s1t4)
4Sc2	PC1(s2t4)	PC2(s2t4)	PC3(s2t4)
4Sc3	PC1(s3t4)	PC2(s3t4)	PC3(s3t4)
% Variance	**70%**	**87%**	**95%**

Before MSA can be performed, the data must to be regrouped according to Equation (6.1) (Diego Mandelli, 2011). Each scenario, $\vec{x}_i = (i = 1,\ldots,\mathrm{I})$, is represented by *M* state variables, x_{im} (m= 1,…*M*) plus time t (ranging from 0 to T) as the state variable:

$$\vec{x}_i = \left[x_{i1}(t_1),\ldots,x_{iM}(t_1),\ldots x_{i1}(t_K),\ldots,x_{iM}(t_K) \right] \tag{6.1}$$

Where $x_{im}(t_k)$ corresponds to the value of the variable x_m (e.g., temperature, pressure, etc. at a computational node) sampled at time t_k (e.g., $t_1 = 0$ and $t_k = T$) for scenario *i*.

The reorganized example data would look similar to Table 6.8

Table 6.6 Example data reorganized for MSA

Scenario 1	Scenario 2	Scenario 3
PC1(s1t1)	PC1(s2t1)	PC3(s3t1)
PC1(s1t2)	PC1(s2t2)	PC1(s3t2)
PC1(s1t3)	PC1(s2t3)	PC1(s3t3)
PC1(s1t4)	PC1(s2t4)	PC1(s3t4)
PC2(s1t1)	PC2(s2t1)	PC2(s3t1)
PC2(s1t2)	PC2(s2t2)	PC2(s3t2)
PC2(s1t3)	PC2(s2t3)	PC2(s3t3)
PC2(s1t4)	PC2(s2t4)	PC2(s3t4)
PC3(s1t1)	PC3(s2t1)	PC3(s3t1)
PC3(s1t2)	PC3(s2t2)	PC3(s3t2)
PC3(s1t3)	PC3(s2t3)	PC3(s3t3)
PC3(s1t4)	PC3(s2t4)	PC3(s3t4)

And finally, after clustering, similar scenarios are clustered together yielding data looking similar to Table 6.7.

Table 6.7 Clustered example data

Cluster 1	Cluster 2
PC1(s1t1)	PC1(s2t1) & PC3(s3t1)
PC1(s1t2)	PC1(s2t2) & PC1(s3t2)
PC1(s1t3)	PC1(s2t3) & PC1(s3t3)
PC1(s1t4)	PC1(s2t4) & PC1(s3t4)
PC2(s1t1)	PC2(s2t1) & PC2(s3t1)
PC2(s1t2)	PC2(s2t2) & PC2(s3t2)
PC2(s1t3)	PC2(s2t3) & PC2(s3t3)
PC2(s1t4)	PC2(s2t4) & PC2(s3t4)
PC3(s1t1)	PC3(s2t1) & PC3(s3t1)
PC3(s1t2)	PC3(s2t2) & PC3(s3t2)
PC3(s1t3)	PC3(s2t3) & PC3(s3t3)
PC3(s1t4)	PC3(s2t4) & PC3(s3t4)

In this case, the dimensionality was reduced from 48 to 24, yielding a 50% reduction.

6.3 PCA and MCA Sample "Toy" Problem

A second illustrative example using real RELAP5 data are presented to show how the process scales. This example also provides visualization of the data via plots to form a better understanding of how the data are behaving under manipulation. In this example, PCA and MCA were performed on a 24 threads, running a 24 state variable (SV), and 61 time steps.

The first step was to normalize the data, and perform PCA, as shown in Figure 6.1.

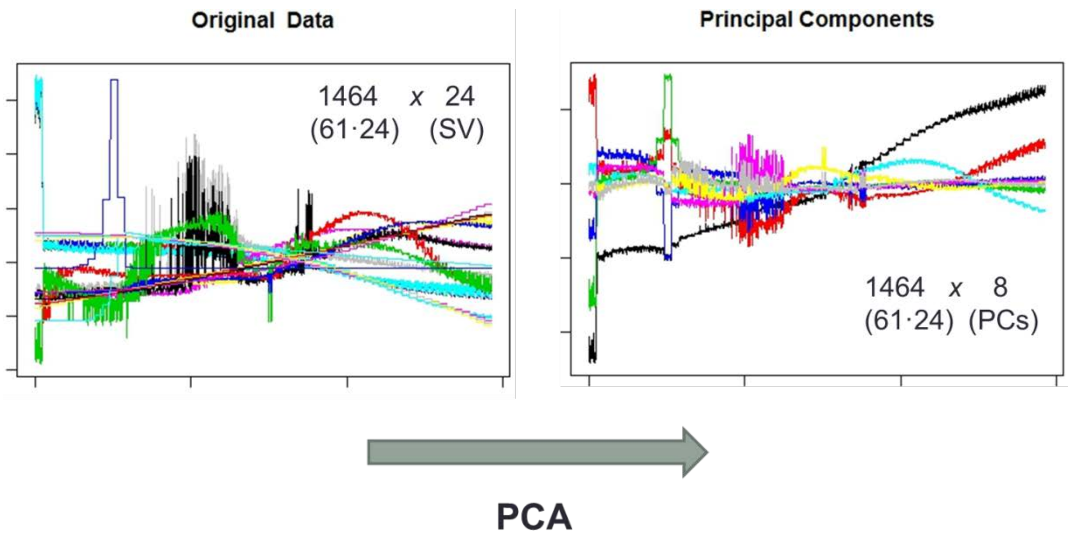


Figure 6.1 An illustration of sample data after performing PCA

Notice that the PC data are representative combinations of state variables that attempt to incorporate all the variability. The variability threshold was set to 95% for this experiment and reduced the state variables from 24 to 8. This showed heavy correlation among state variables and drastic dimensionality reduction.

The mean shift algorithm plots were much more difficult to discern to the naked eye, shown in Figure 6.2.

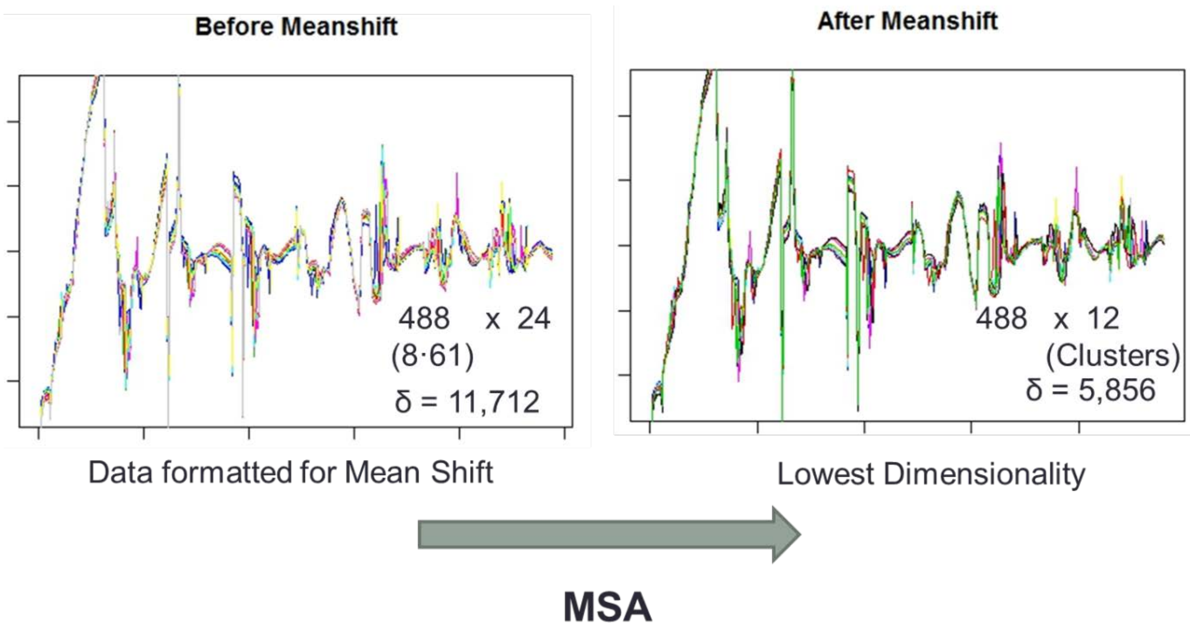


Figure 6.2 Data formatted for the Mean Shift Algorithm before and after clustering

In this case, the 24 scenarios were reduced to 12 representative scenarios by using the mean shift algorithm. Once the data was clustered, it was rearranged back into principal components, shown in Figure 6.3

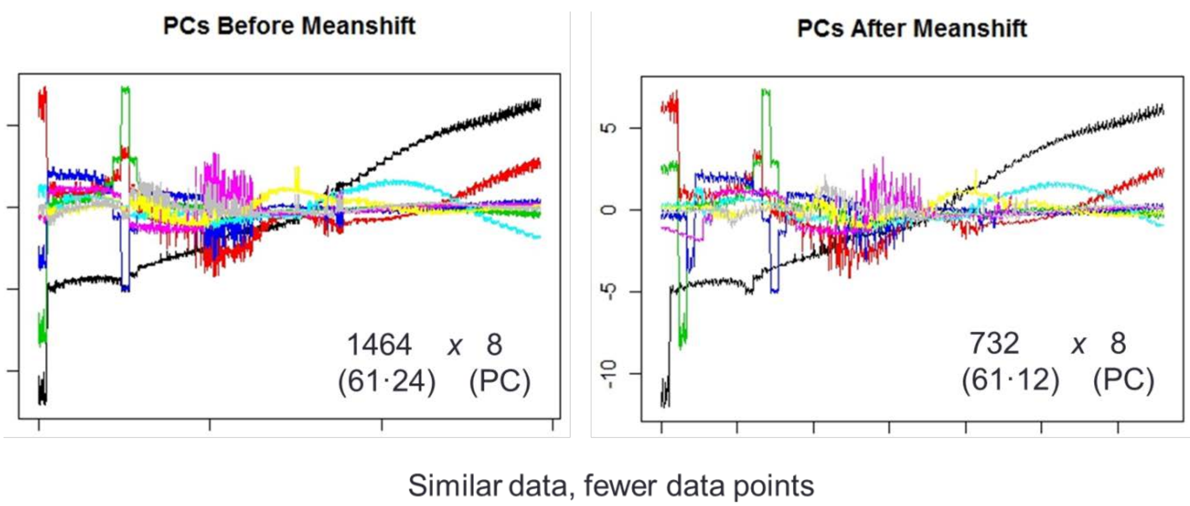


Figure 6.3 Principal components before and after clustering

Notice that the principal components before and after clustering look almost identical. The original data can also be retrieved, and is shown in Figure 6.4.

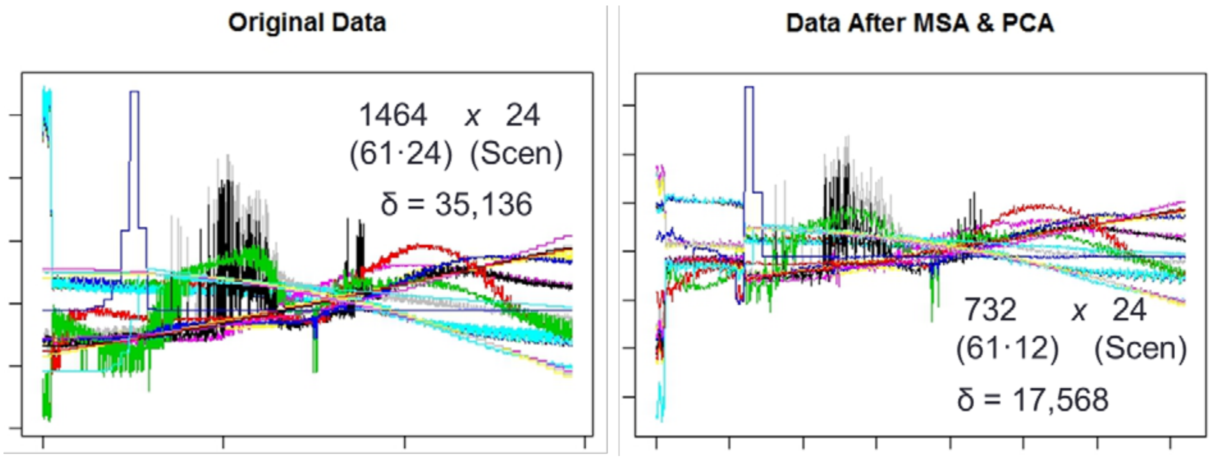


Figure 6.4 Original data with dimensionality 35,136, and processed data with dimensionality 17,568

Notice how closely the processed data resembles the original data, but with a dimensionality reduction of around 50%.

To test the speed of the algorithms, a simple benchmark was performed on the sample data; speeds were averaged over ten runs, and the results are displayed in Table 6.8.

Table 6.8 Benchmarking PCA and MSA with 24x24x61 sample data

	MSA w/o PCA (s)	Organization w/o PCA (s)	MSA w/ PCA (s)	Organization w/ PCA (s)
Average (10 runs):	0.077	0.825	0.048	0.70
Totals (MSA+R):	0.902		0.750	
% reduction (Only MSA)	**37.5%**			
% reduction (Only PCA)	**14.9%**			
% reduction (MSA & PCA)	**16.9%**			

This table illustrates how much time each process took. Performing MSA without PCA and the organizational steps took 0.077 seconds. Just the organization by itself took 0.825 seconds. Surprisingly, performing PCA and organizing actually took less time (0.70 seconds, a decrease of 14.9 percent) than organizing the data without performing PCA. This is due to the fact that after PCA (dimensionality reduction) there are less data, so organization isn't as computationally intensive. The combination of organization, PCA, and MSA took 0.75 seconds, down from 0.902 seconds of MSA and organization without PCA. That saved 16.9 percent by first doing PCA, then MSA, versus just organizing the data, doing MSA, then reorganizing it. This is the reason PCA is a critical component of RAPSS – it sped up MSA by at least 17 percent. It is further expected to significantly increase this percentage with larger dimensionality.

Before making any conclusions about the utility of PCA, a final check was performed. Cluster membership should be the same in MSA for the raw data and the principal components, as illustrated in Table 6.9 or else PCA is not accurately representing the data.

Table 6.9 Sample problem cluster membership with and without PCA

Cluster #	Membership w/PCA BW=5	Membership w/o PCA BW=5
1	[1,20]	[1,20]
2	[2,3,5,12,16,18,21]	[2,5,12,16,18,21]
3	4	4
4	[6,8,9]	[6,8,9]
5	7	7
6	[10,22]	[10,22]
7	11	11
8	13	13
9	[14,19,23]	[14,19,23]
10	15	15
11	17	17
12	24	24
13		3

Cluster membership, for this example, was almost identical. One can conclude that scenario three was probably on the edge of cluster two, and got bumped into its own cluster after PCA. One can observe, from this example that PCA not only significantly reduced the amount of computational time, but also accurately preserved the data in such a way that cluster membership was nearly identical.

6.4 Organizing, linear approximation intervals, and PCA in R

The first script executed by R, initPCA.r loaded the libraries and initial parameters into the R memory (see Appendices A.5 and B.5). This was separated from the rest because it was slow, and it is not necessary to reinstall all libraries with every cycle of RAPSS. Because PCA relies on the assumption of linear correlation between state variables, and state variables are not always linearly correlated, the data was split into linear approximation intervals before performing PCA.

6.4.1 Determining Linear Approximation Intervals

While the idea to use linear approximation intervals was described by Mandelli (2011); the automation of the process was the original work of the author.

Principal component analysis is only capable of determining linear correlations among state variables. Because thermal fluids simulation codes are characteristically non-linear in nature (especially during transients), if PCA were performed over the entire length of simulation, the linear correlation requirement would not be satisfied and large errors would be introduced. However, if the time history was broken into small enough chunks that the data looked linear over that interval when compared with the next, PCA could still be utilized. These intervals do not have to be the same size. For instance, during steady state conditions, the intervals should be large to save processing time; during transients, the intervals should be small to maintain the linearity assumption. To determine how linear a data set is, the *norm* of the change in the covariance matrix is determined. A detailed explanation follows.

The covariance of matrix A, cov(A), shows how one variable is correlated to another. The variances appear along the diagonal and the covariances appear in the off-diagonal elements, shown in Equation (6.2). After normalization, the diagonals take on values of 1, as normalized data has a variance of 1.

$$\text{cov(A)} = \begin{bmatrix} \sum x_1^2 / N & \sum x_1 x_2 / N & \ldots & \sum x_1 x_n / N \\ \sum x_2 x_1 / N & \sum x_2^2 / N & \ldots & \sum x_2 x_n / N \\ \ldots & \ldots & \ldots & \ldots \\ \sum x_n x_1 / N & \sum x_2 x_2 / N & \ldots & \sum x_n^2 / N \end{bmatrix} \tag{6.2}$$

For perfectly correlated data, the derivative of the covariance matrix is zero:

$$\frac{d}{dt}\left(\text{cov}(A)\right) = \vec{0} \tag{6.3}$$

We can use the limit definition of the derivative to reduce Equation (6.3) further:

$$\lim_{\Delta t \to 0}\left[\text{cov}\left(\frac{A(t+\Delta t) - A(t)}{\Delta t}\right)\right] = \vec{0} \tag{6.4}$$

Or presented differently, if the time-series data are broken into two, arbitrarily small, sequential blocks, A and B, (formally A(t+Δt) and A(t) in Equation (6.4)) and the data are perfectly correlated, then:

$$\frac{\text{cov(A)} - \text{cov(B)}}{\Delta t} = \vec{0} \tag{6.5}$$

The data, however, are rarely perfectly correlated. To determine how correlated the data actually are, a measurement of the norm is used. In Euclidian space, the norm is often thought of as the intuitive notion of the length of a vector, namely:

$$\|x\| = \sqrt{x_1^2 + \ldots + x_n^2} \tag{6.6}$$

But can be more generally thought of as the square root of the inner product of the vector and itself:

$$\|x\| = \sqrt{\vec{x}*\vec{x}} \tag{6.7}$$

Where $\vec{x}*$ denotes the conjugate transpose of $\vec{x}$. We can use the norm to determine the "length" of the difference in covariance matrices. If the length is small, the matrices can be assumed to be correlated enough to perform principal component analysis. In summary, if the norm of the difference in covariance matrices divided by the change in time is less than a small, user defined threshold, ε, (Equation (6.8)) then the data are linearly correlated enough for principal component analysis.

$$\left\| \frac{\text{cov(A)} - \text{cov(B)}}{\Delta t} \right\| < \varepsilon \tag{6.8}$$

6.5 Principal Component Analysis (PCA)

After the linear approximation intervals are found, PCA is performed (see section 2.5.1) on each interval. The same number of principal components are used for each interval, or else it would be impossible to compare later with MSA. The number of principal components was fond by comparing the percent variation for the entire interval (not the linear approximation interval) to the user defined threshold (usually 95%). The number of components that add up to the desired level of variance representation were used for each interval. This relied on the conservative assumption that if the intervals were more linearly correlated than the entire data set, then the same number of principal components would represent more variability for the linear approximation intervals than for the entire data set.

The final step was to organize the data in a way that can be read by the mean shift algorithm (see Section 6.2, Equation (6.1)). Since PCA was meant to reduce the number of state variables, MSA was used to reduce (cluster) the scenarios. Instead of organizing the data by state variable, and incorporating the scenarios into the state variables, the data was organized by scenario, and the principal components (formally state variables) were grouped similar to Equation (6.1). This was written to a .csv file as “PC,” followed by the restart number that it was analyzing from RAPSS-STA, to be read by the mean shift algorithm.

6.6 The Mean Shift Algorithm in C++

Rather than spend time “reinventing the wheel”, RAPSS used a version of the mean shift algorithm written by Mandelli (2011), however, the code was adapted in several places to serve RAPSS. While cluster.h remained relatively untouched, several key changes were made to MeanShift.h (formally MeanShift.cpp in Mandelli (2011)). Please see Appendices A.10, A.11, B.10, and B.11 for details.

6.7 unMSAPCA.R

The purpose of unMSAPCA.R was to take the clustered data, organized for MSA, reorganize it for PCA, perform operations to convert from principal components to state variables, and finally to unnormalize the data. Please see Appendices A.7 and B.7 for further details.

7 RAPSS-STA User Interface and Display

In order to determine how “risky” a certain scenario is, RAPSS-STA reads user defined criteria for thresholds of concern for certain components. These “red” and “yellow” threshold values are used to determine that a R5 run ended by trip (causing a red threshold indicator), or is to be continued on to the next cycle (causing a yellow threshold indicator). When a run ends by exceeding a red threshold, RAPSS-STA alerts the STA, and generates plots of the behavior of the component before exceeding the threshold to give the STA an idea of what a particular component might look like before exceeding a safety threshold.

Unlike the other modules, the user interface for RAPSS-STA was written in html. This turned out to be an ideal language for the interactive nature required by RAPSS-STA. The user is first presented with the scenarios of immediate concern, highlighted by a flashing “alert” animated .gif. Figure 7.1 illustrates an example user interface of RAPSS-STA.

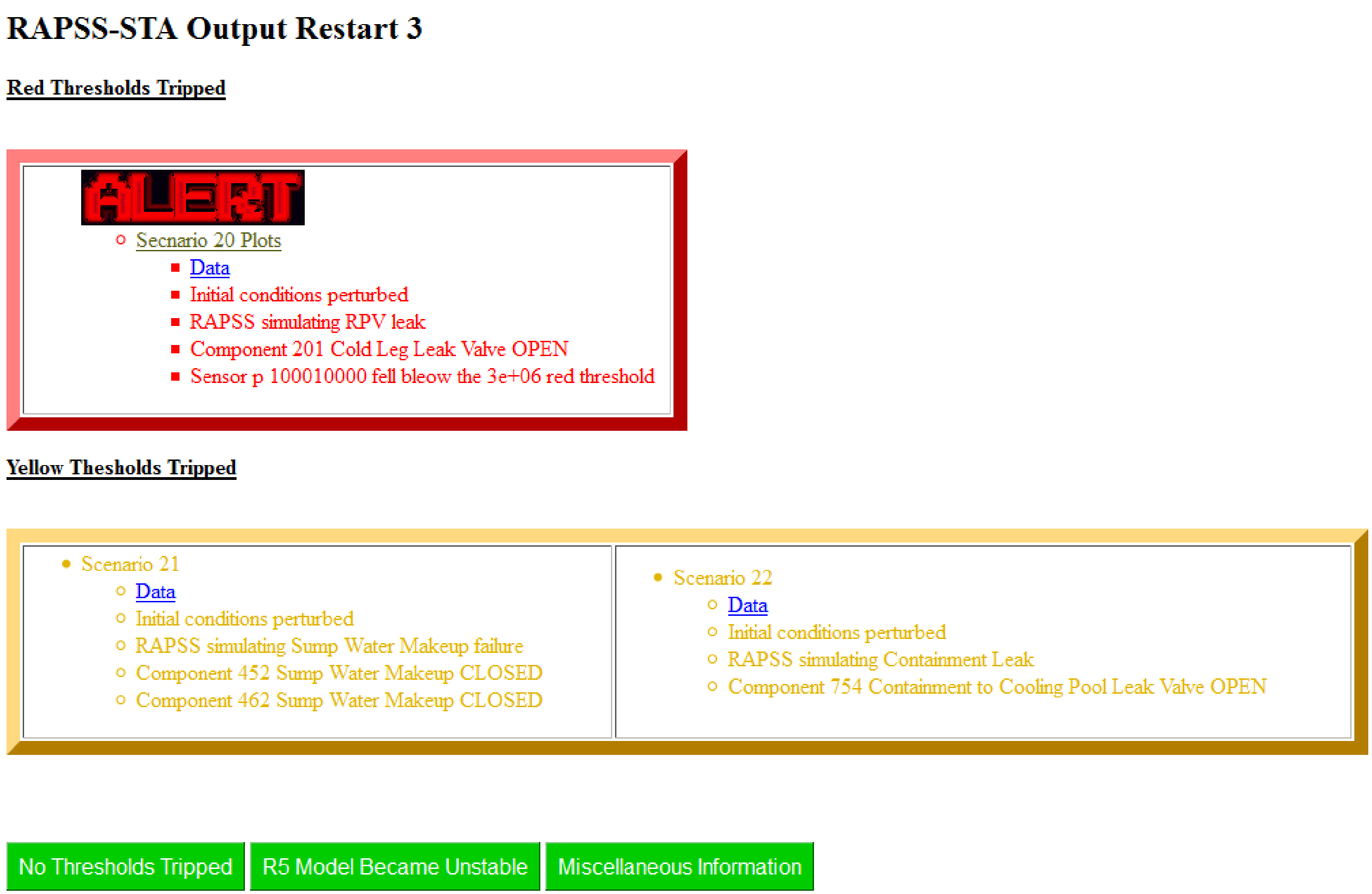


Figure 7.1 An example user interface for RAPSS-STA

Each of the links provides the user with more information about the scenario of interest. By clicking on the plot links, it takes the user to graphs of the sensor, or sensors that cause the run to exceed a “red” threshold. Clicking on the “Data” link takes the user to the time-series data produced from that run of RELAP5. Any scenarios that had sensors exceed “yellow” thresholds, but not “red” thresholds, meaning there might be a problem later but not immediately, are organized in the Yellow Thresholds Tripped section. This output was similar in nature to the boxes in the Red Thresholds Tripped section.

By clicking on the “Scenario 20 plots” link, the user is taken to a plot similar to Figure 7.2:

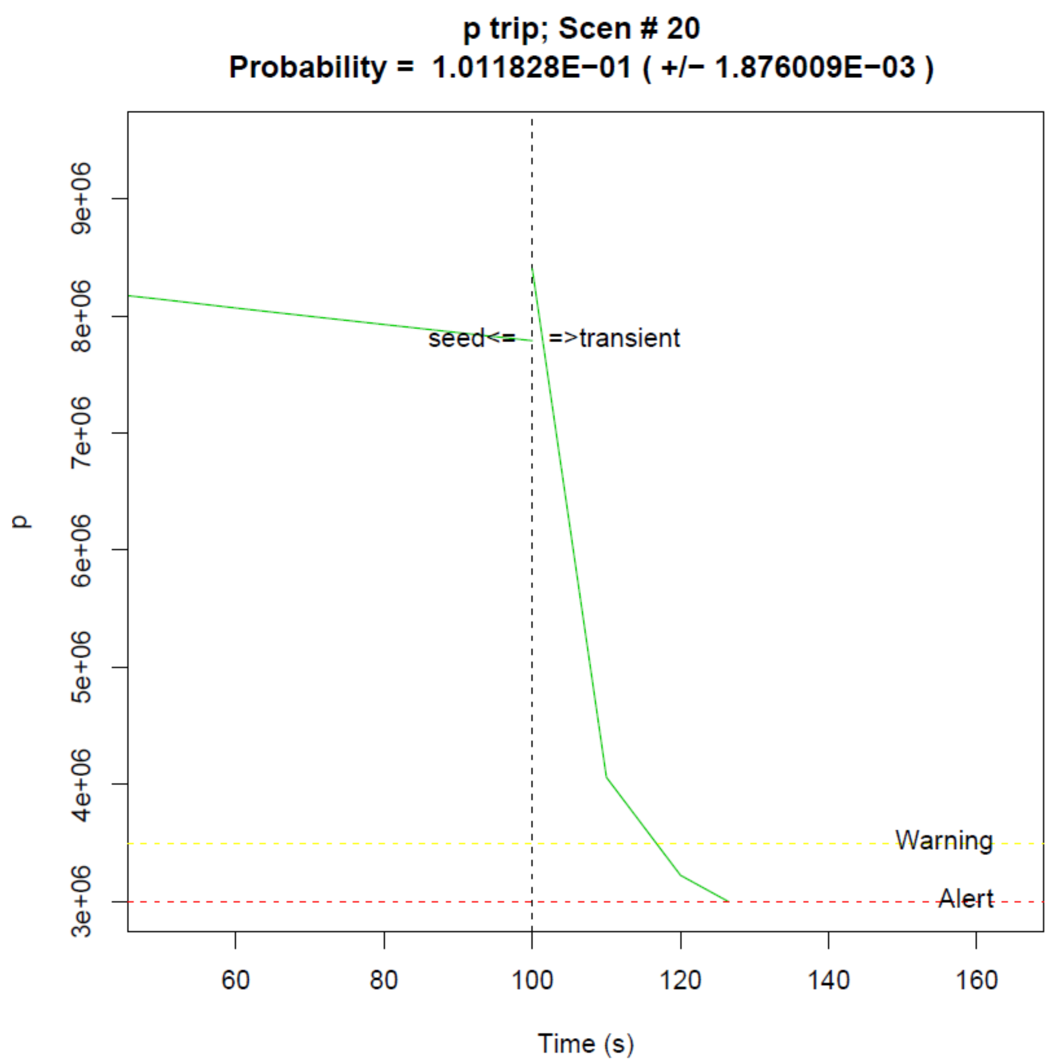


Figure 7.2 an example of a RAPSS plot of a parameter of interest falling below a user defined threshold

The green boxes below the yellow and red thresholds on the main user interface in Figure 7.1 contains extra information about the scenarios that did not trip any thresholds. Clicking on the “No Thresholds Tripped” box takes the user to a screen similar to Figure 7.3.

RAPSS-STA Output Restart 2

No Thesholds Tripped

- Secnario 0
  - Data
  - Initial conditions perturbed
- Secnario 1
  - Data
  - Initial conditions perturbed
- Secnario 2
  - Data
  - Initial conditions perturbed
- Secnario 3
  - Data
  - Initial conditions perturbed
- Secnario 4
  - Data
  - Initial conditions perturbed
- Secnario 5
  - Data
  - Initial conditions perturbed
- Secnario 6
  - Data
  - Initial conditions perturbed

Figure 7.3 Example output of the "No Thresholds Tripped" data from RAPSS-STA

The green next box, named, "R5 Model Became Unstable" in Figure 7.1, is for any R5 simulations that ended by errors. This screen resembles Figure 7.4.

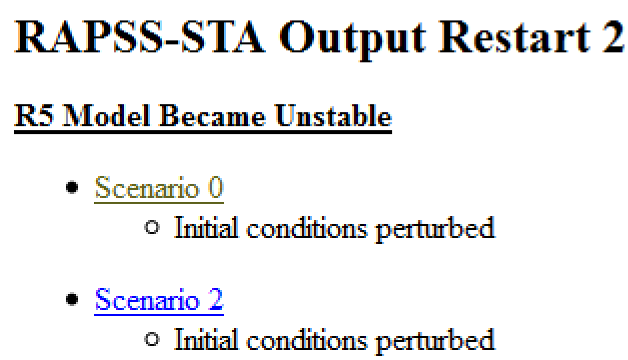


Figure 7.4 Example output for "R5 Model Became Unstable" Box for the RAPSS-STA display

The final green box, in Figure 7.1, "Miscellaneous Information," contained information regarding the scenario/cluster membership. The output resembled Figure 7.5.

RAPSS-STA Output Restart 2

Cluster Information

- Cluster 1 Plot
  - Cluster Data
  - Cluster Members
    - Scenario 0
    - Scenario 1
    - Scenario 2
    - Scenario 3
    - Scenario 4
    - Scenario 5
    - Scenario 6
    - Scenario 7
    - Scenario 8
    - Scenario 9
    - Scenario 10
    - Scenario 11
    - Scenario 12
    - Scenario 14
    - Scenario 15
    - Scenario 16
    - Scenario 17
    - Scenario 18
    - Scenario 19
    - Scenario 20
    - Scenario 21
    - Scenario 22

Figure 7.5 Example output for "Miscellaneous Information" Box for the RAPSS-STA display

By clicking on the "Cluster 1 Plot" link shown in Figure 7.5, the user is taken to a view of the normalized state variables contained in the given cluster. While this plot (Figure 7.6) is not intended for direct use by the senior reactor operators, it can be useful for debugging purposes, and may shed light on the behavior (e.g, increasing/decreasing) of the state variables when other methods fail, or if the STA is interested in reviewing the possible outcomes of a specific scenario considered by RAPSS.

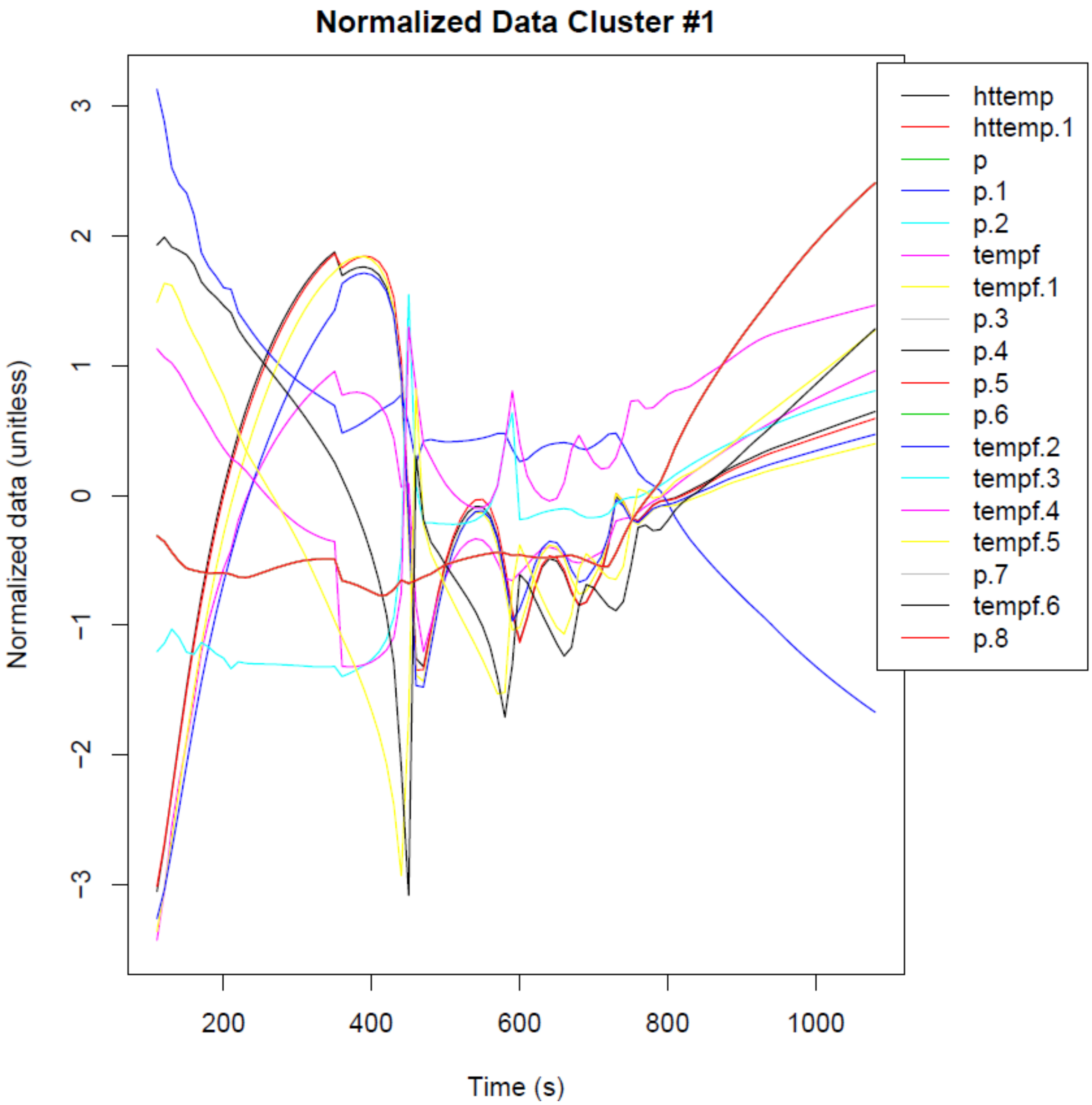


Figure 7.6 Normalized RAPSS data clusters for state variables of interest

8 RAPSS-STA Results

The Real Time Simulator (see Section 4.2.1) was used to sample a former MASLWR facility output and made the data available to RAPSS-STA in a similar fashion to the facility output as if it were running. The following experiments were performed using the Standard Problem 3 MASLWR experiment.

8.1 The MASLWR Standard Problem 3 Experiment

The MASLWR experiment used for RAPSS-STA benchmarking was the Standard Problem (SP) 3 test, originally performed in July 2011, as a way to characterize the steady-state natural circulation in the primary side of the facility during various core power configurations. In this experiment, the power inputs of the core heaters were increased from 10 percent of full power to 80 percent of full power in 10 percent increments over a roughly 6000 second run time. Each time the power was increased, flow rate and temperatures were monitored to determine whether flow stabilization was achieved. According to the test procedures, if the core subcooled margin degraded below 20 degrees F for each power interval, the operator was to take action by decreasing the core heater setpoint until steady state was achieved (Mai & Luo, 2011).

While the operator action for each interval was not ideal for benchmarking purposes, the SP-3 experiment was the simplest experiment available to the RAPSS team due to the proprietary nature of most of the facility experiments. A RELAP5 model of the facility (see Section 4.2.2) was used as the modeling software, and the SP-3 experiment was plugged into the Real Time Simulator, acting as the system. The RAPSS-STA data and the MASLWR model were then compared to determine how well the simulation represented the facility.

8.2 Comparing SP-3 Experiment and the R5 Model

Determining the difference between two state variables (e.g., core temperature between the RELAP5 run and the facility) required several steps. First, both variables needed to be run to the same point in time. This required trimming one of the time series to match with the other. Next, the time steps needed to be identical. The MASLWR data used time steps of one second. Since using this small of time step caused RELAP5 to run painfully slow, larger time steps for the RELAP model were used. However, this required trimming the MASLWR data to match with similar time steps. A simple R script was written for this purpose. The script compared the modulus (remainder) of the time steps. For example, if R5 was set to output every 2.5 seconds, the series would look like 2.5, 5, 7.5, 10… etc. Since the MASLWR data was composed of the natural numbers (i.e., 1, 2, 3...), everything that was not a natural number from the R5 data needed to be trimmed. After that, any number in the MASLWR data that did not have a comparable time step in the R5 data was trimmed, leaving 5, 10, 15, 20, etc... for both data sets.

After the two vectors ($\vec{A}$, the MASLWR data, and $\vec{B}$, the R5 data) were rendered comparable, the maximum error was measured by calculating the individual error for each time step, and taking the L-infinity norm (maximum) of the resulting vector, described analytically by Equation (8.1):

$$\text{MaxError} = \left\| \frac{\left| \vec{A} - \vec{B} \right|}{\vec{A}} \right\| x100\% \tag{8.1}$$

This yielded a single measurement of error, used for benchmarking between the R5 and MASLWR experiments.

Before RAPSS-STA was implemented, a simple comparison of core temperature and pressure was performed between the R5 and MASLWR data over the full length of the MASLWR experiment run time, 6000 seconds. This served as a benchmark for how accurately later runs of RAPSS-STA were capable of reproducing the data.

As can be observed by Figure 8.1, the core temperature between the R5 model and the facility do not align very closely, yielding a maximum error from Equation (8.1) of 21.2%.

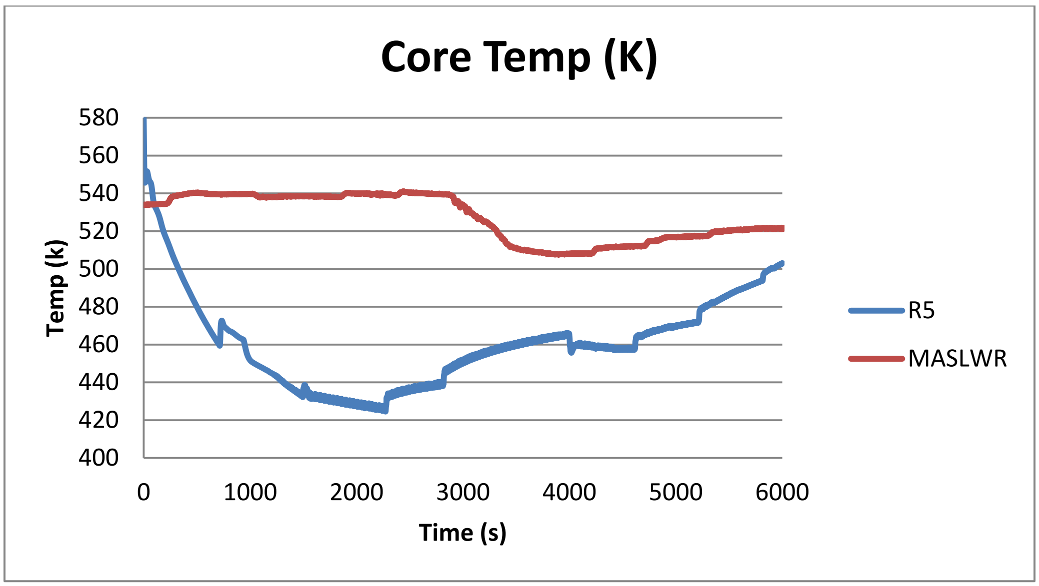


Figure 8.1 Core temperature shown for the MASLWR facility and the R5 model

Core pressure was even worse, varying by roughly an order of magnitude, yielding a maximum error of 96.6% as seen in Figure 8.2:

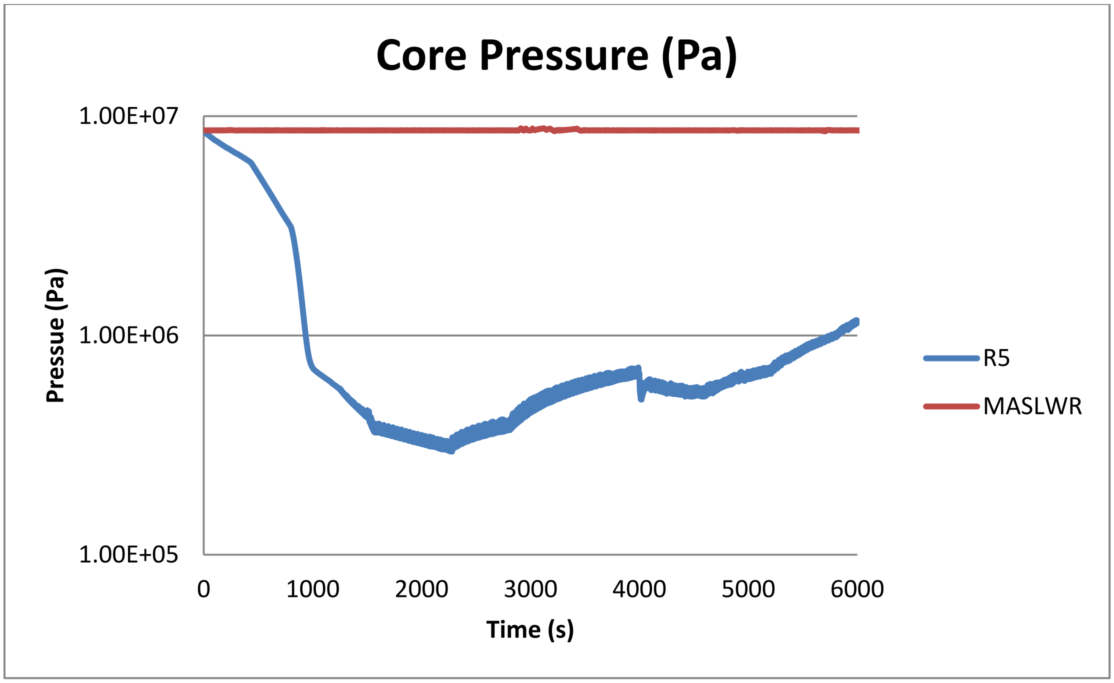


Figure 8.2 Core pressure shown for the MASLWR facility and the R5 model

While these errors were certainly not ideal, they serve as an important reminder of how crucial it is that the model accurately reflect the facility. In the SP-3 experiment, there was an operator tweaking parameters to maintain fairly constant core pressure and temperature. In order to accurately represent the experiment, an attempt was made to simulate the operator actions.

8.3 Simulating SP-3 Experiment Operator Actions

There was an attempt to model the exact operator actions performed in the SP-3 experiment. However, this required constant adjusting of the main steam mass flow rate (FVM-602M in the MASLWR facility) to maintain constant pressure and temperature. To simulate this, linear regressions (Figure 8.3) of the main steam mass flow rate were fit to the MASLWR output:

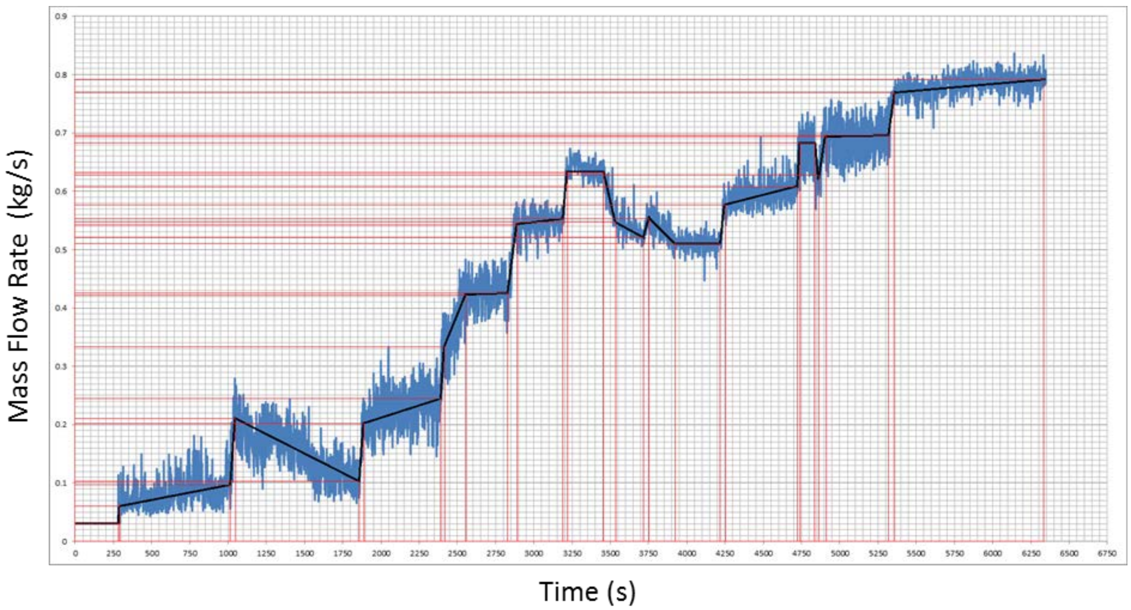


Figure 8.3 Linear regressions of flow velocity were determined from the MASLWR data to match the operator actions. Figure compliments of Thomas Riley.

After these adjustments were made for a 3600 second run, the core temperature and pressure aligned much closer with the facility, yielding maximum errors for temperature and pressure of 2.37 percent and 17.4 percent, respectfully, displayed in Figure 8.4 and Figure 8.5:

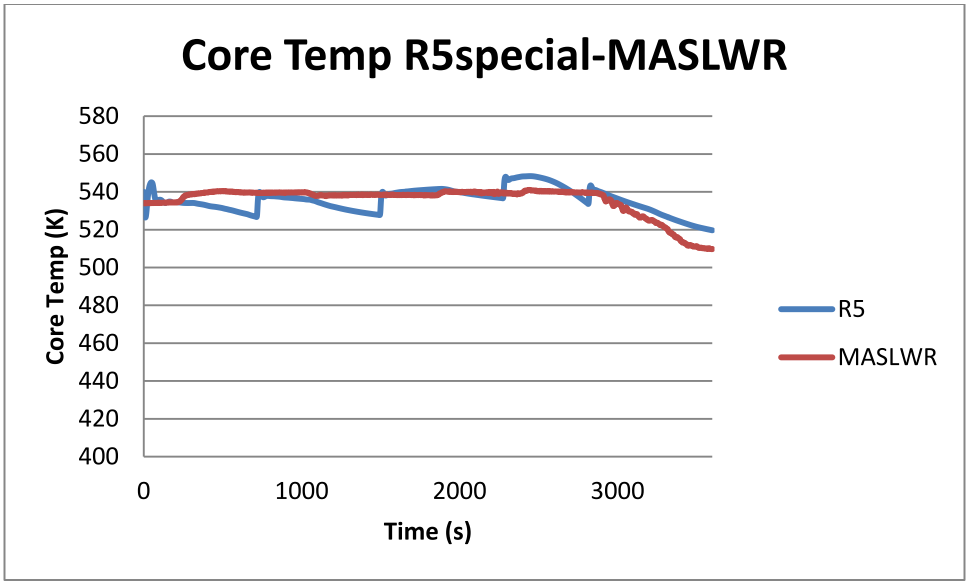


Figure 8.4 A plot of core temperature from a special R5 run designed to reflect small operator actions

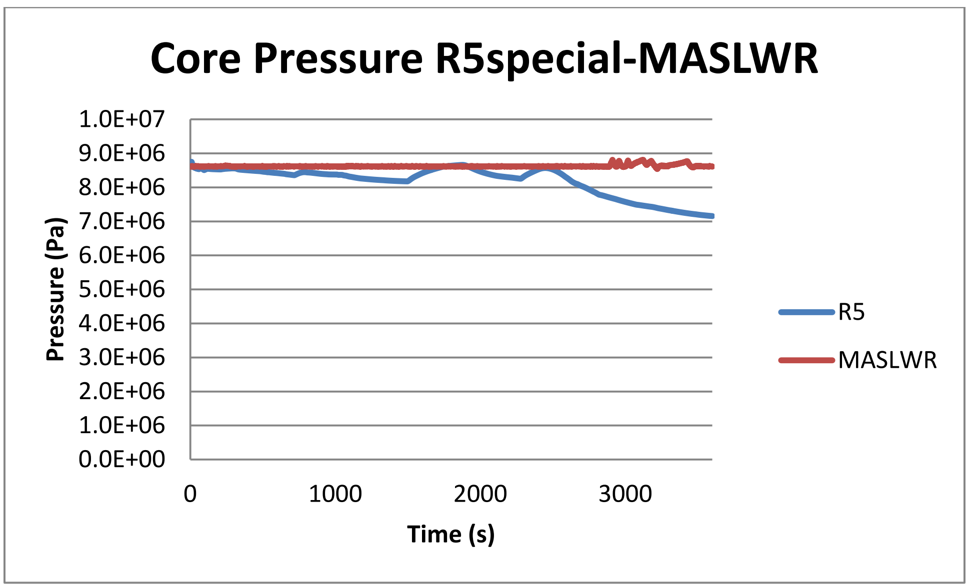


Figure 8.5 A plot of core pressure from a special R5 run designed to reflect small operator actions

While this was indeed encouraging for a single run, the model became unstable with restart runs, the primary control mechanism for RAPSS-STA. During a MASLWR experiment, if an operator adjusted a parameter, there was no way to communicate that change to the RELAP5 model using the current RAPSS architecture. RAPSS samples the current conditions and projects them ahead. Without accounting for the change in model parameters, restarting the model from the facility's "current conditions" caused the model to "jump" from its previous state, and usually resulted in RELAP5 ending by errors. For this reason, the constant flow velocity, but changing temperature and pressure model was chosen for RAPSS-STA demonstrations. This model was used with the acknowledgement that the core temperature and pressure will be most likely show greater errors than the aforementioned special case of the R5 model.

8.4 RAPSS-STA and the SP-3 Experiment

An experiment was performed to explore how closely RAPSS-STA representative scenarios simulated the MASLWR facility. The experiment duration in this case was 1000 seconds.

For the R5 model without modeling operator actions, the core temperature decreased at a fairly consistent rate, while the MASLWR facility was modified by the operator during the run to maintain a steady temperature, yielding a maximum error from Equation (8.1) of 16.3 percent, displayed in Figure 8.6:

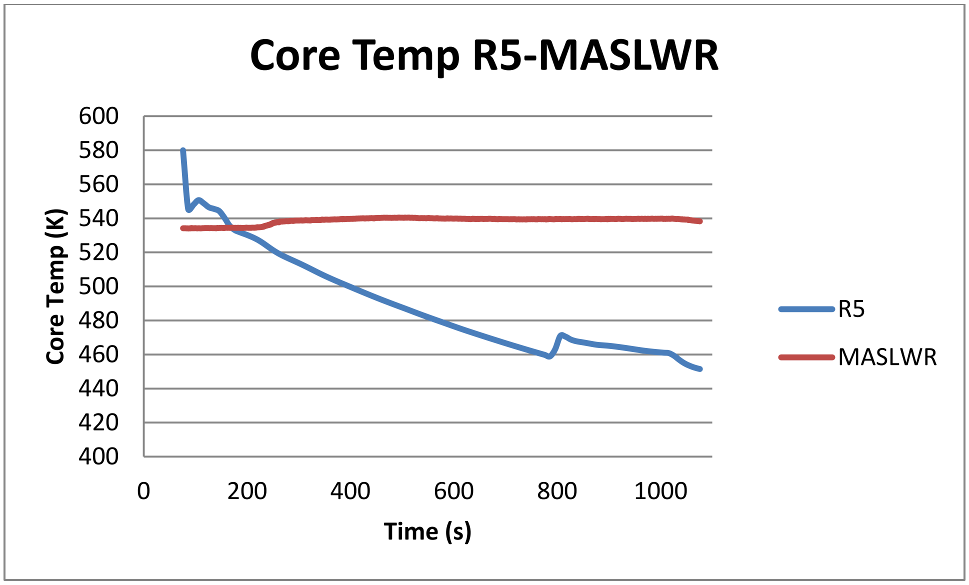


Figure 8.6 Core temperature from a normal R5 run plotted with core temperature from the MASLWR facility

Similar behavior was observed in the core pressure. However, the core pressure displayed an even stronger departure from the MASLWR data with a maximum error from Equation (8.1) of 91.7 percent, illustrated in Figure 8.7:

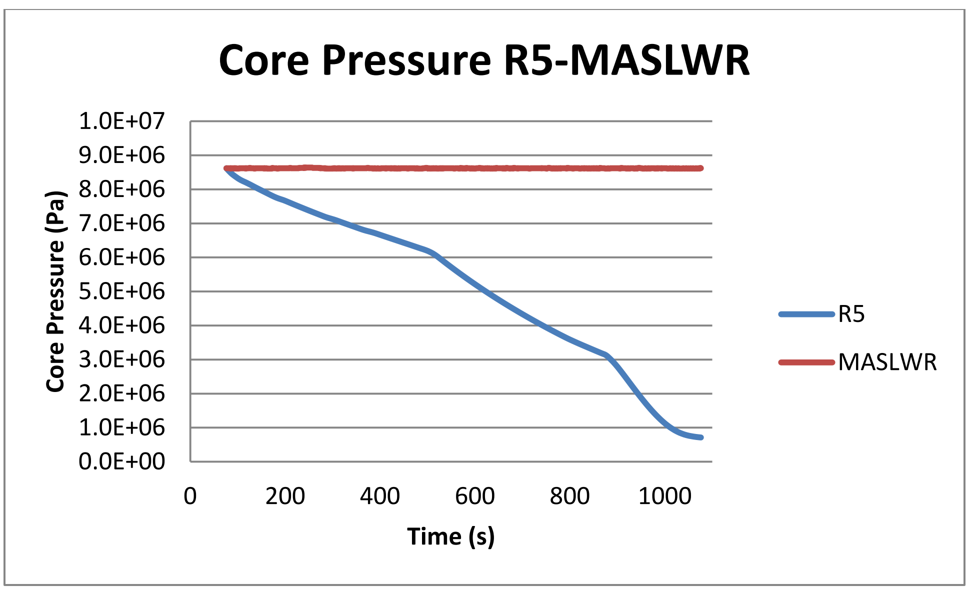


Figure 8.7 Core pressure from a normal R5 run plotted with core temperature from the MASLWR facility

When RAPSS-STA modeled the facility, it output a representative scenario based on many different starting conditions and configurations. Figure 8.7 and Figure 8.8 display the core temperature and pressure from a RAPSS-STA cluster, potted with the temperature and pressure from the MASLWR facility. For this simulation, the maximum error for temperature and pressure were 12.4 percent 7.92 percent respectfully.

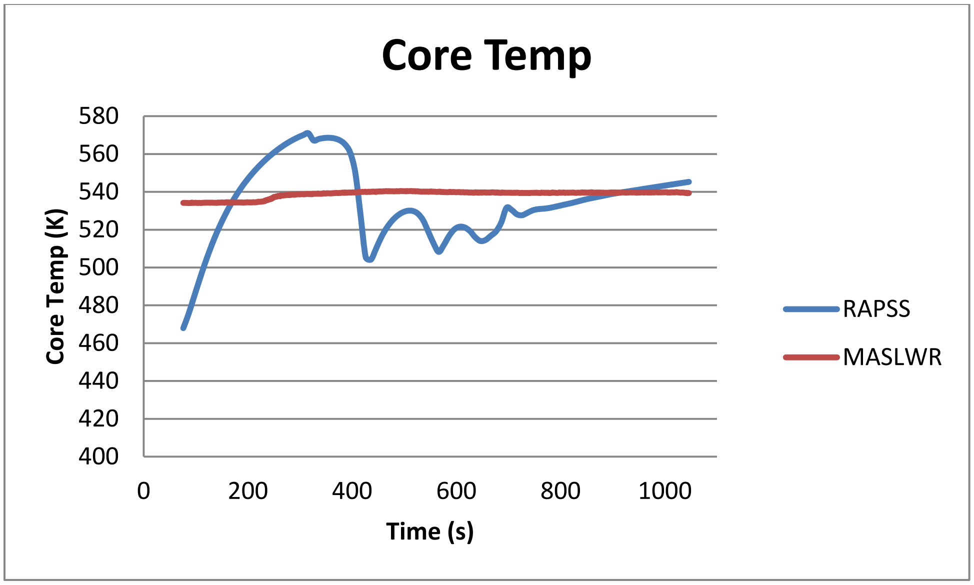


Figure 8.8 Core temperature from a representative RAPSS cluster plotted with core temperature from the MASLWR facility

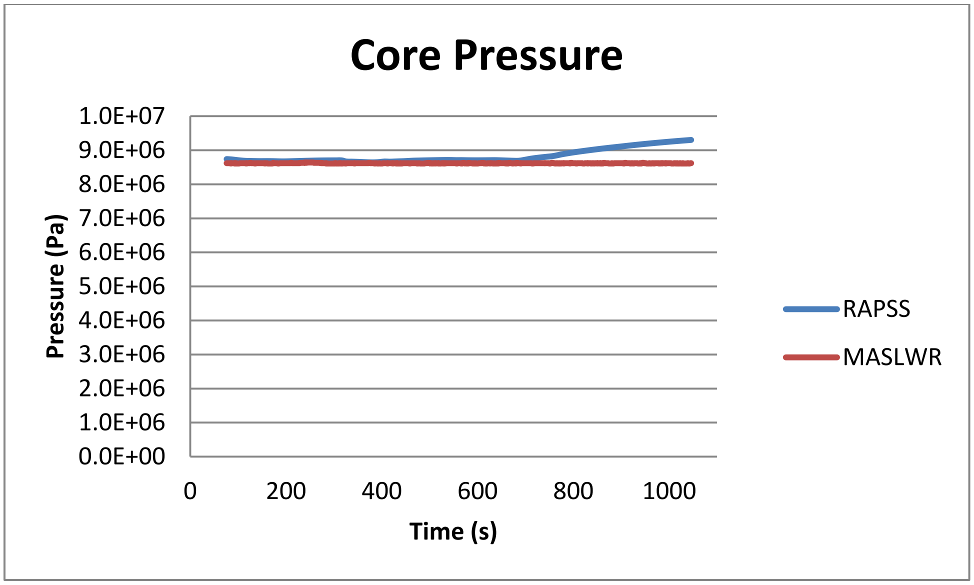


Figure 8.9 Core pressure from a representative RAPSS cluster plotted with core pressure from the MASLWR facility

8.5 Results Summary

A normal RELAP5 run wandered quite far from the conditions of the facility due to the unaccounted operator actions. For a 6000 second experiment, this led to a maximum error for core temperature and pressure of 21.2 and 96.6 percent, respectively. When operator actions were modeled explicitly for 3600 seconds, the error dropped to 2.37 percent and 17.4 percent, respectfully, for the duration of the experiment. While these errors were encouraging, it was not possible to integrate this type of “constant tweaking” model into the RAPSS-STA architecture.

Instead, the R5 model that did not include operator actions was used for RAPSS-STA benchmarking. For a 1000 second experiment on this R5 model, the core temperature and pressure maximum errors were 16.3 and 91.7 percent respectfully. When the same model was used, but in RAPSS-STA, the representative scenarios yielded maximum errors for core temperature and pressure of 12.4 and 7.92 percent, respectfully.

A significant improvement. It is expected that using a model that accurately reflects operator actions would reduce the error even further.

From a big-picture perspective, this demonstration of RAPSS-STA has exposed two important prerequisites for future real-time decision support. First, the model must accurately reproduce the system. RAPSS heavily relies on the assumption that the modeling software can accurately simulate system conditions. If this is not the case, no matter what fancy numeric tricks are performed, RAPSS will never be able to make accurate predictions, especially when operator actions can change the plant status. The RELAP5 model used for MASLWR was adequate for the purpose of this demonstration, but greater model fidelity is paramount for complete RAPSS-STA implementation with the MASLWR facility. If a better model were to become available (through NuScale, for example), it is expected that RAPSS-STA would be able to more accurately predict future system conditions. However, as the current model stands, it does a poor job of modeling the system conditions, and is not satisfactory for real-time decision support.

The second prerequisite is the ability to accurately follow and represent operator actions. When an operator acts, he changes the physical properties of the system. This invalidates all predictions up until that point because the previous predictions were based on a model and a system that does not exist anymore. If an operator constantly "tweaked" the system, RAPSS would need to wait until the operator did not act for a predetermined amount of time to provide any meaningful analysis, system assessment, and plant status predictions.

9 RAPSS-EOC

An emergency operations center (EOC) is the primary command and control facility responsible for carrying out emergency management in the event of a large-scale disaster (like a catastrophic failure and radioactive contamination release at a nuclear power station, for instance). Emergency operations centers are responsible for understanding the “big picture,” of the disaster and are primarily responsible for gathering and analyzing data, and making decisions to protect life and property.

To illustrate an extension of RAPSS methodology, RAPSS was applied to emergency operations centers (RAPSS-EOC). Instead of the system being a nuclear power plant and the simulation software being RELAP5, in RAPSS-EOC the system was atmospheric conditions, and the modeling software was a plume program written by the author specifically for this purpose.

The output of RAPSS-STA was color-coded plot of the current plume concentrations, followed by plots of clustered scenarios of future plume behavior based on wind rose data. The underlying data structure was a grid of log-scaled concentrations, generated from the plume program. The plot also displayed a hypothetical city at a user-defined location. The display alerted the user when a “red” threshold was reached by concentrations at the city location exceeding a user defined threshold. The display also alerted the user when a “yellow” threshold was reached, but not a red. This signified that the plume was close to the city, and concentrations might be rising, but the city was not in danger yet.

9.1 RAPSS-EOC Structure

RAPSS-EOC contained six C++ files and two R scripts. Of the six C++ files, two of them were identical to those used in RAPSS-STA, cluster.h and MeanShift.h, originally written by Diego Mandelli. FunctionsEOC.h (see Appendices C.3 and D.3) was similar in nature to bloodandguts.h in RAPS-STA. CyclePlume.h (see Appendices C.2 and D.2) was similar in functionality to CycleR5.h in RAPS-STA.

PlumeProgram.h (see Section 9.2, Appendices C.4 and D.4) was the original work of the author and was analogous to RELAP5 in RAPSS-STA. The plume program was used as a substitute for a modeling software such as GENII or RASCAL. GENII and RASCAL are both compiled for Windows and operated through a Windows GUI. This was not appropriate for RAPSS integration, as RAPSS was designed to run in a high performance UNIX environment.

Pmain.cpp (see Appendices C.1 and D.1) was the main control mechanism for the program, and was relatively short and simple. GridOrganizer.R (see Section 9.3, Appendices C.5 and D.5) was a fairly simple R script that organizes the grid output by the plume program to prepare it for mean shift analysis. The final two R scripts were created at run time, initR.r, and unpdateRWindex.r (see Appendices C.7, C.8, D.7, and D.8). As the names suggest, these function identically to the R scripts that share the same name in RAPSS-STA.

9.2 The Plume Program

The main driver behind the plume program was the integrated puff model (see Section 2.6.1). The output of this model was organized as a grid of squares, each assigned a concentration for a given time. X and Y were lateral and horizontal distance from the source along the direction the wind was blowing. A negative X value denoted an area behind the plume, and was assigned a value of 0.0001 instead of zero for ease of log-scaling. A positive Y value denoted an area to the left of the wind center line, and negative to the right. For this proof of principle, Z, the vertical distance, was set to zero to measure ground concretion, but could easily be changed for later applications.

Historical atmospheric data was used from the Remote Automated Weather Station (RAWS) USA Climate Archive[4] for both current state estimation and future condition prediction of the system. The RAWS Climate Archive is a network of weather stations run by the U.S. Forest Service and Bureau of Land Management mainly to observe potential wildfire conditions. The data chosen for this demonstration was the Juniper Dunes (near Pasco, Washington) data set. This area was chosen for its simple terrain and proximity to Columbia Generation Station commercial nuclear power plant, as well as a handful of other research facilities that could potentially cause the need for an emergency operation center in the event of a release.

9.2.1 Estimating the Current State of the System

Before predictions about the system could be made, the current state of the system was estimated as a starting point. In weather prediction, estimating the current state of a system as complex as atmospheric conditions, is equally as critical as the model predicting ahead, as small variations in initial conditions could cascade into huge changes later for chaotic systems. For this proof of principle, however, only wind direction and wind speed were sampled. The RAWS data produced hourly observations of these data for nearly every day since 1987. After selecting the location, the user could click on the

---

http://www.raws.dri.edu/index.html[4]

"Daily Summary" link on the left side of the screen, followed by selecting the day and English or metric units. This produced a text file that could be read by RAPSS-EOC.

An estimate of the duration the plume has been active is provided by the user when writing a RAPSS-EOC input file. RAPSS-EOC reads the wind speed and direction from the RAWS data file, and reproduces the plume using the integrated puff model. This was expressed as a grid, the resolution of which can be changed by the user. This current state was then added to any predictions by simply adding equivalent sections in the grid.

9.2.2 Predicting the Future State of the System

Similar to reading a fault tree for predicting transient probabilities in RAPSS-STA, RAPSS-EOC read wind rose data from RAWS to predict wind speed and direction. After selecting the location, the user can click on the "Wind Rose Graph and Tables" link on the left side of the screen, followed by selecting the month, English or metric units, and the format of the output tables (RAPSS-EOC requires downloadable ASCII instead of html). This produces a text file that can be read by RAPSS-EOC. The wind rose diagram that the downloadable data generated looked similar to Figure 9.1:

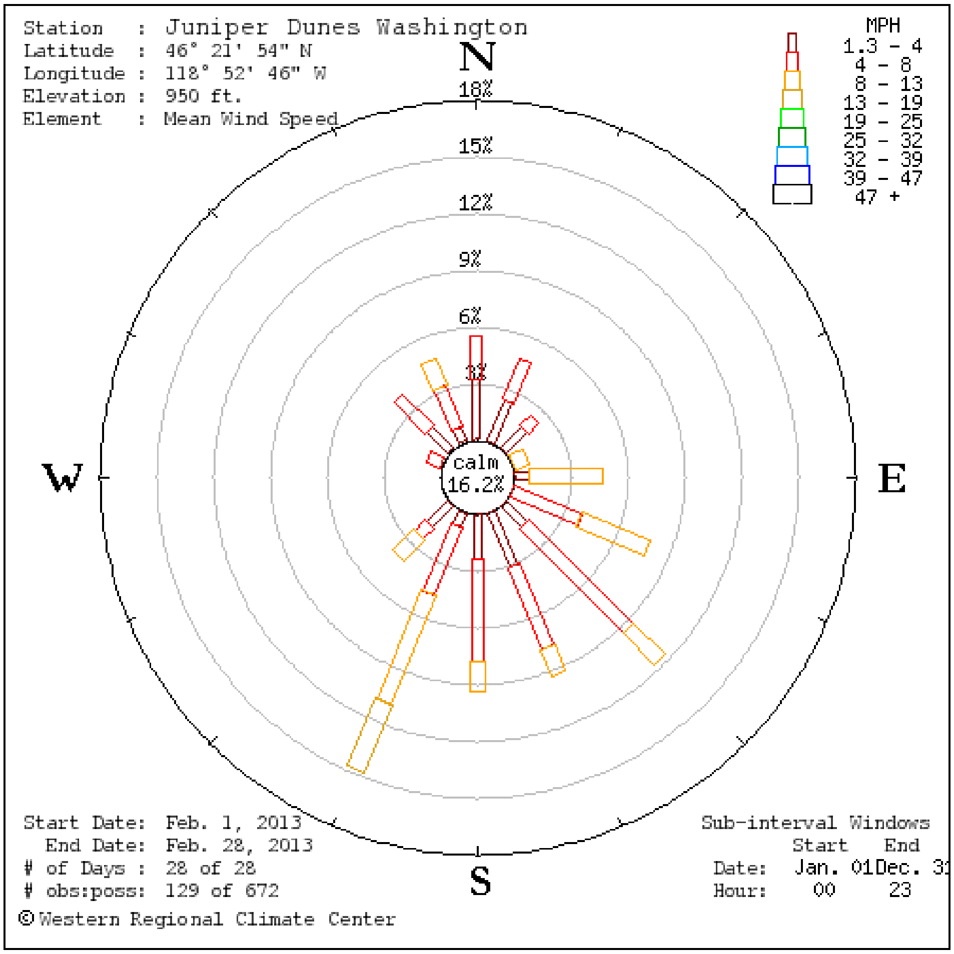


Figure 9.1 Example wind rose for Juniper Dunes from http://www.raws.dri.edu/cgi-bin/rawMAIN.pl?waWJUN (US Forest Service and Bureau of Land Management, 2012)

The longer arms of the wind rose expressed greater probability of the wind coming from that direction, and the thickness of the arms expressed the probability of a given speed.

9.3 Data Processing

In the case of simply looking at concentration per area, it did not make sense to use PCA. Principal component analysis looks for correlations among state variables, since there was only one state variable of interest, ground level concentration, no reduction was possible or necessary.

However, since there were many different scenarios that sample various wind speeds and directions, a version of the mean shift algorithm was utilized to reduce the scenario output size by determining representative scenarios.

An illustrative example follows to describe the process of converting a grid of concentration measurements into a format that can be used as an input for the mean shift algorithm. In this example, the user starts with an n *x* n matrix of concentration measurements across the X and Y directions, shown in Table 9.1.

Table 9.1 Example 3 x 3 grid used to demonstrate the organizational structure of the MSA for RAPSS-EOC

Scenario 1			**Scenario 2**			**Scenario 3**		
Y1X1S1	Y1X2S1	Y1X3S1	Y1X1S2	Y1X2S2	Y1X3S2	Y1X1S2	Y1X2S2	Y1X3S2
Y2X1S1	Y2X2S1	Y2X3S1	Y2X1S2	Y2X2S2	Y2X3S2	Y2X1S2	Y2X2S2	Y2X3S2
Y2X1S1	Y2X2S1	Y2X3S1	Y2X1S2	Y2X2S2	Y2X3S2	Y2X1S2	Y2X2S2	Y2X3S2

Similarly to how the data was organized in Equation (6.1), the X and Y values were stacked on top of each other so one scenario was fully represented by one column, as shown in Table 9.2.

Table 9.2 Example organized data ready for clustering by the mean shift algorithm

Scenario 1	Scenario 2	Scenario 3
Y1X1S1	Y1X1S2	Y1X1S3
Y2X1S1	Y2X1S2	Y2X1S3
Y2X1S1	Y2X1S2	Y2X1S3
Y1X2S1	Y1X2S2	Y1X2S3
Y2X2S1	Y2X2S2	Y2X2S3
Y2X2S1	Y2X2S2	Y2X2S3
Y1X3S1	Y1X3S2	Y1X3S3
Y2X3S1	Y2X3S2	Y2X3S3
Y2X3S1	Y2X3S2	Y2X3S3

The mean shift algorithm was used to eventually generate representative clusters of similar scenarios similar to Table 9.3.

Table 9.3 Example clustered data output from the mean shift algorithm

Cluster 1	Cluster 2
Y1X1C1	Y1X1C2
Y2X1C1	Y2X1C2
Y2X1C1	Y2X1C2
Y1X2C1	Y1X2C2
Y2X2C1	Y2X2C2
Y2X2C1	Y2X2C2
Y1X3C1	Y1X3C2
Y2X3C1	Y2X3C2
Y2X3C1	Y2X3C2

The data could then be reorganized to look similar to Table 9.1, but for a reduced number of representative scenarios.

9.4 The RAPSS-EOC User Interface and Display

After loading RAPSS-EOC, the user is asked to enter the name of the input file, followed by how many cycles he or she wishes to run. RAPSS-EOC reproduces the current state from the wind data specified in the input file, and makes predictions across several parallel computational nodes to predict the future state of the system. While this is happening, a timer is running in the background to know when to update the estimate of the current state. A user specified real-time-speed-up-factor was used to generate more frequent updates of the state estimation than one hour in real time.

R was used as the primary graphics engine. The data was log-scaled with each order of magnitude assigned a shade of color starting with white to represent concentrations of 0.0001, up to dark red, which was used to represent concentrations of the maximum release rate used for this demonstration, $1x10^{30}$ Bq/s. A simple polar grid with the usual 16 directions (N, NNW, etc…) was overlain on the rectangular grid of concentration measurements yielding an output similar to Figure 9.3.

After the data from a desired number of cycles was obtained, the user accessed the predictions through an interface, written in html, similar to the RAPSS-STA interface, displayed in Figure 9.2:

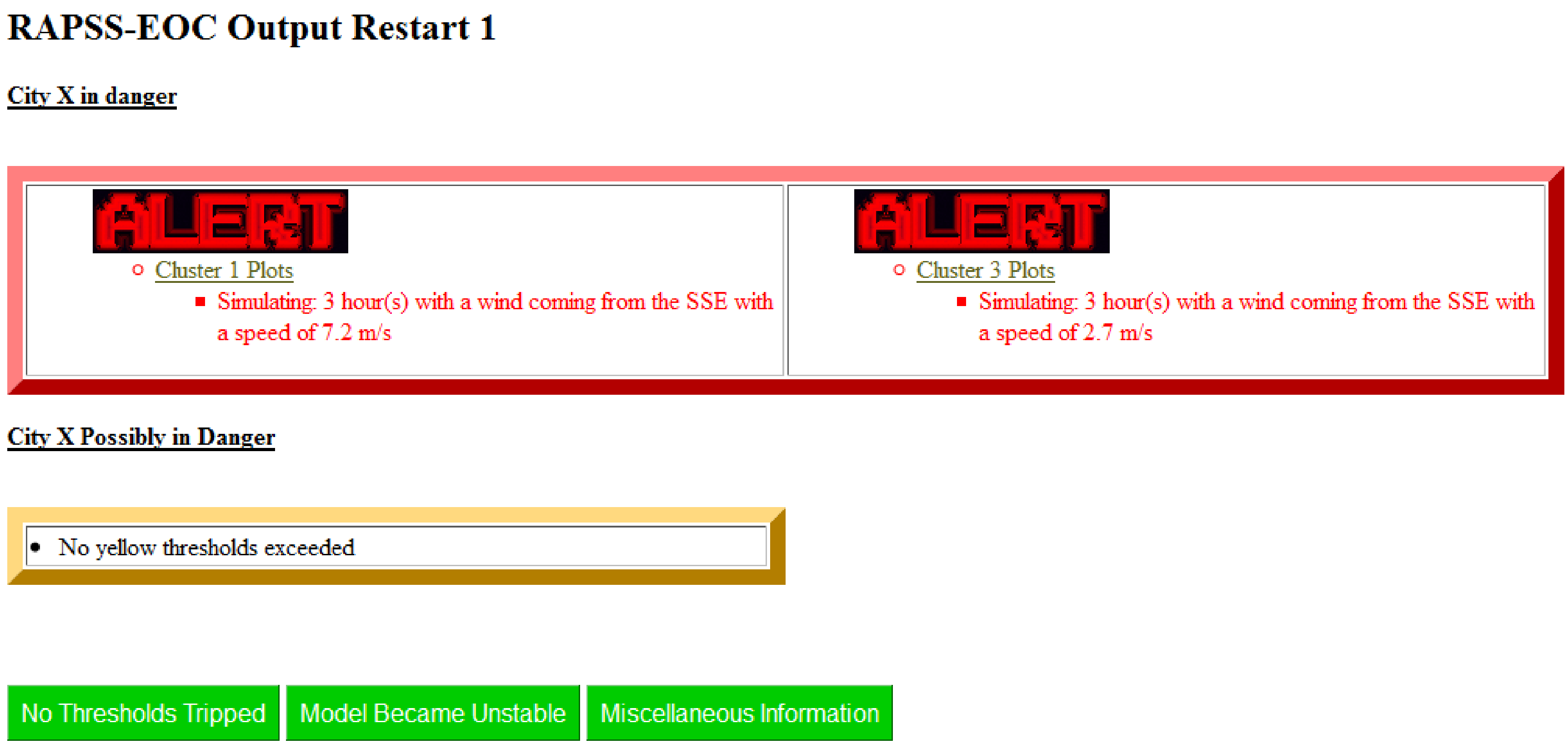


Figure 9.2 An example RAPSS-EOC user interface.

The risk of a given scenario was based on the proximity to a user defined hypothetical city. When the user clicks on one of the plots links in the red boxes of Figure 9.2, she is taken to a 2-paged PDF where the first page is an estimate of the current state of the plume, with the city marked as a black dot, similar to Figure 9.3:

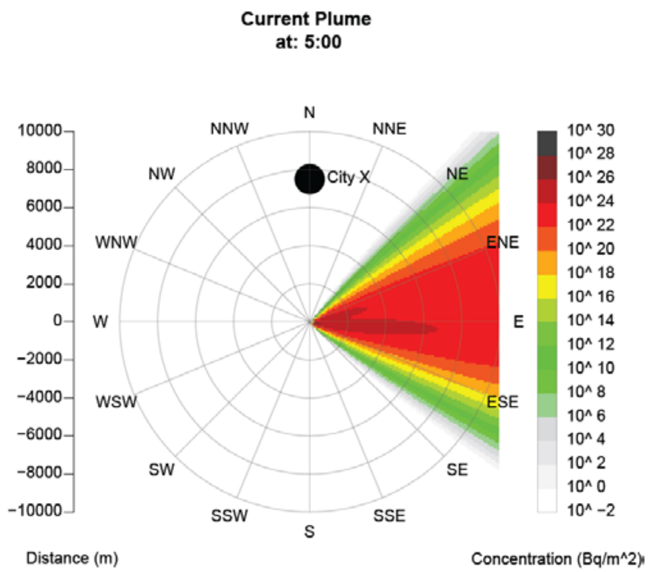


Figure 9.3 An example of an estimate of the current state of a plume from RAPSS-EOC

The second page shows an estimate of the plume that either crossed or came close to the city in question similar to Figure 9.4:

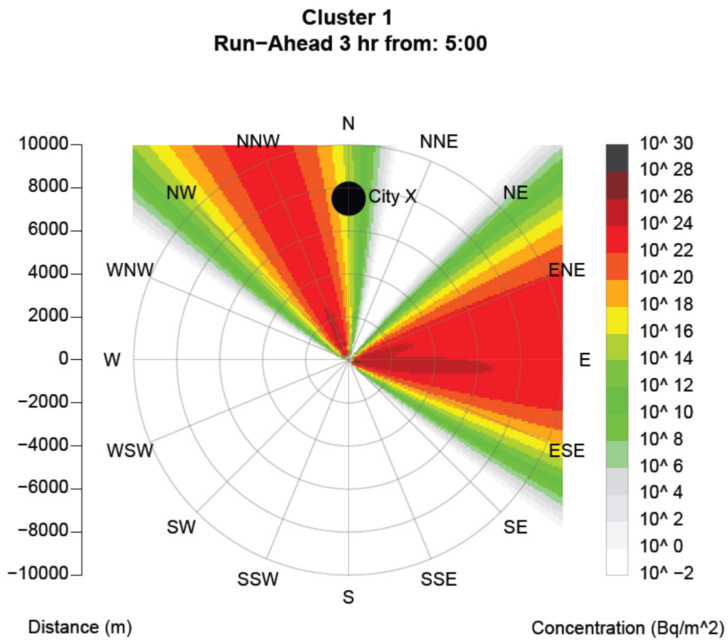
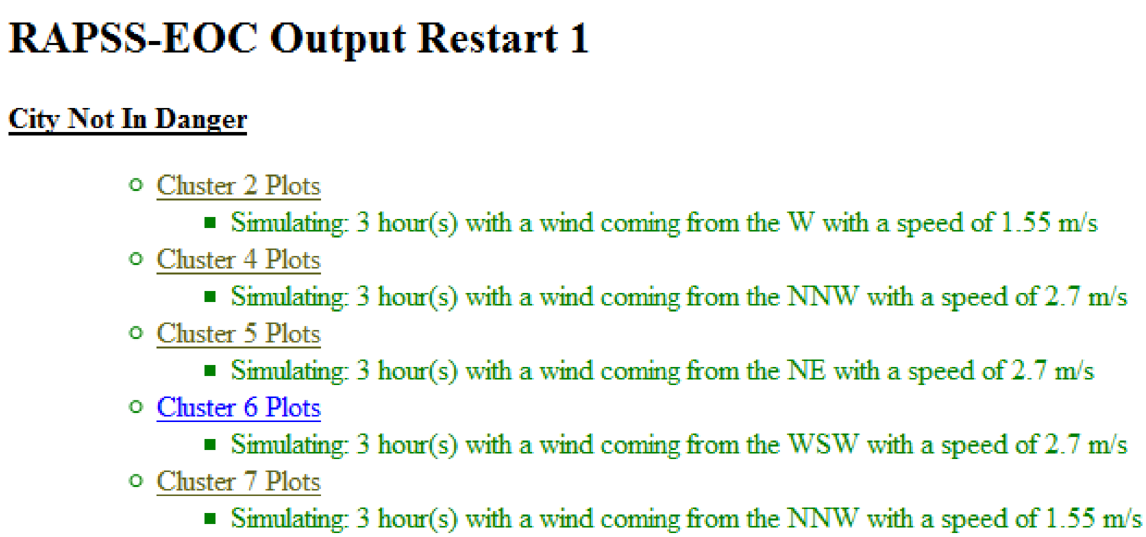


Figure 9.4 An example of an estimate of the future state of a plume from RAPSS-EOC

The green "No Thresholds Tripped" box of Figure 9.2, takes the user to a screen that details the clusters that did not endanger the city, similar to Figure 9.5:


RAPSS-EOC Output Restart 1

City Not In Danger

- Cluster 2 Plots
  - Simulating: 3 hour(s) with a wind coming from the W with a speed of 1.55 m/s
- Cluster 4 Plots
  - Simulating: 3 hour(s) with a wind coming from the NNW with a speed of 2.7 m/s
- Cluster 5 Plots
  - Simulating: 3 hour(s) with a wind coming from the NE with a speed of 2.7 m/s
- Cluster 6 Plots
  - Simulating: 3 hour(s) with a wind coming from the WSW with a speed of 2.7 m/s
- Cluster 7 Plots
  - Simulating: 3 hour(s) with a wind coming from the NNW with a speed of 1.55 m/s

Figure 9.5 An example of an estimate of No Thresholds Tripped screen from RAPSS-EOC

The green “Instabilities” box of Figure 9.2 in RAPSS-EOC simply takes the user to a screen that displays no instabilities in the model, similar to Figure 9.6. RAPSS-EOC used a simple plume program that was not prone to the same instabilities as RELAP5.

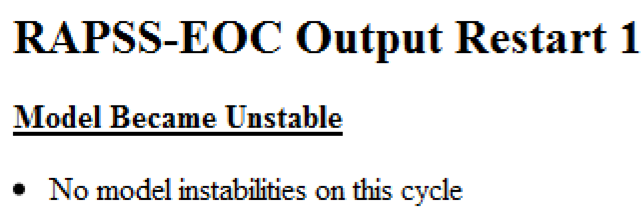


Figure 9.6 An example the instabilities screen from RAPSS-EOC

The green “Miscellaneous Information” box of Figure 9.2 took the user to a screen that details which scenarios were contained in each cluster, similar to Figure 9.7:

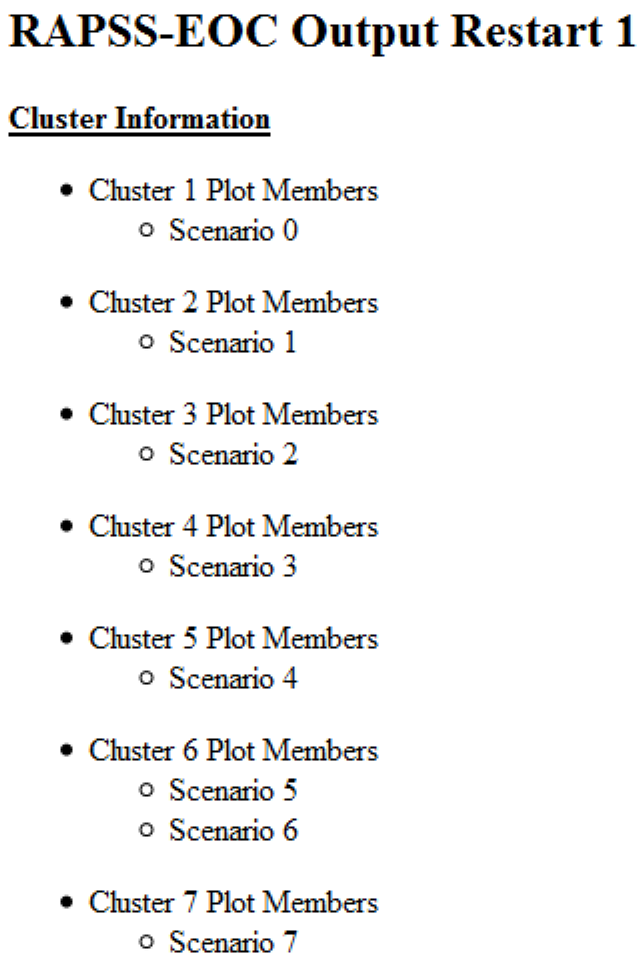


Figure 9.7 An example the miscellaneous information screen from RAPSS-EOC

This user interface allowed the user to quickly page through important displays, and provided more in depth information accessed by a simple click of the mouse.

10 Discussion and Conclusion

The beauty of RAPSS is that it is not simply dependent on a particular modeling code or system. It is generalizable across many fields, from manned space missions, to air traffic controllers. Anything that can be modeled and sampled can be integrated into the RAPSS methodology. This dissertation has shown that it is possible to apply the RAPSS methodology to two significantly different situations, RAPSS-STA and RAPSS-EOC.

In a field that is dominated by risk-adverse attitudes, a methodology to revolutionize risk assessment performance in nuclear power plants (i.e., RAPSS-STA) would certainly be beneficial, if not critical to the success of the industry. We cannot continue forever to rely on legacy computer codes and traditional methods of risk assessment. The worst commercial nuclear power disaster in the United Sates (Three Mile Island) could have been prevented by a better understanding of system conditions by the operators, and the future impacts of their decisions. While there have been great improvements in some areas post Three Mile Island, shift technical advisors and unit supervisors still use *ad hoc* decision making in situations that have the potential to take lives, or cause billions of dollars worth of damage. This needs to change. RAPSS-STA directly addresses this issue by offering STAs and unit supervisors real-time model-driven decision support.

As was seen during the Fukushima accident, adequate assessment and management of a disaster is crucial to minimize the damage to lives and property. RAPSS-EOC does illustrate the promise of improving ecological modeling in emergency operations centers using high performance computing. For now, however, RAPSS-EOC was used mainly to illustrate the flexibility of the RAPSS architecture across a new system. RAPSS-EOC also serves another important purpose by hinting at the possibility of generalizing RAPSS for other systems.

The long term goal of RAPSS development is real-time, model-drive decision support for operators of nearly any type of complex network. While this is a monumental task, and certainly not realistic given the time constraints of a single dissertation, this research did lay the foundation and proved the principles necessary for further development of the RAPSS architecture. This research has opened the door for a flood of possible future applications by not only developing the principles, but proving them in two distinctly different situations: nuclear power plant control rooms, and emergency operations centers.

10.1 Limitations

While RAPSS-STA and RAPSS-EOC have both shown great potential throughout the course of this research, they are still are susceptible to limitations. RAPSS-STA has underscored how important it is that the model of the environment accurately represent the system. RAPSS will never produce accurate predictions if the model it uses does not adequately reflect the operational fundamentals of the system. For the preceding experiments, it was seen that the model did not represent the system well, and RAPSS-STA struggled to make meaningful predictions as a result. It is interesting, though, to observe how much better predictions RAPSS-STA made than a normal R5 model, especially core pressure measurement (91.7 percent error without RAPSS to only 7.92 error with RAPSS).

In cases where the user can influence the system (i.e., in nuclear power plants, but not in environmental modeling), the changes the user makes to the model are not currently communicated to the model. This is especially challenging for situations where the operator makes several changes over a short time-span. For the proof of principle, this lack of communication was acceptable to illustrate the architecture, but future work requires this communication.

RAPSS-EOC also shows limitations in key areas. Firstly, no radiological release event data was used to calibrate RAPSS-EOC predictions. Environmental monitoring is significantly harder to simulate in a laboratory environment, so past radiological dispersion events would need to be used. The most recent and obvious candidate would be data from the Fukishima accident.

RAPSS-EOC also does not use an industry-standard plume modeling code. The plume program written to demonstrate the proof of principle was purposely simplified to focus attention on the application of the methodology, rather than get caught up in duplicating the intricacies and functionality of industry standard codes that have decades of advantage in development time.

10.2 Future Work

While this proof of principle has exposed the limitations of the RAPSS methodology in two applications, it is precisely this type of research that moves the concept forward. Any new methodology begins with a limited scope, and as researchers spend more time exposing more new limitations, paths are made to overcome those limitations. In order to progress RAPSS into a usable architecture for real-time model-driven support for operators of complex networks, several tasks, outlined next, are required.

10.2.1 Future Work RAPSS-STA

The next task for RAPSS-STA is to break free of the limitations of RELAP5. It is absolutely essential that RAPSS-STA gain the ability to translate operator actions into changes in the model. The way that makes the most sense at this time is to use control room simulation software. This has the ability to not only speed up RAPSS-STA, but also allows for real time operator actions to be reflected in the model.

The other main missing piece of RAPSS-STA is an ability to “look back” at previous facility data and compare it current transient predictions to identify if the facility might be in a transient without the operator’s knowledge. Once a transient begins, that information must be communicated to the model, or else RAPSS-STA would predict that the model would restabilize when in fact, it would travel farther down the transient path.

Looking further into the future, the next steps are to optimize RAPSS-STA to run much faster than real time. This can be done by either lowering the resolution in the modeling code, or, perhaps using a new, state of the art, much faster thermal fluids code such as R7.

Another route would be to use Gaussian process model emulators instead of running instances of large thermal fluids models in real time. When using emulators, one would run many transients under many different conditions before the software is installed. Once the software is installed and running, it would use extrapolations and interpolations from the preloaded libraries to determine any scenarios that it has not seen before, and output a display of uncertainty associated with the analysis. This holds the prospect of a very light and fast program without the enormous resources required to continually run multiple instances of a thermal fluids code. More information about Gaussian process modeling can be found at: http://www.mucm.ac.uk/.

Once truly faster-than-real-time and operator action translation methodologies have been adopted, RAPSS-STA can be integrated with the MASLWR facility at Oregon State University for real-time decision support. After RAPSS-STA has been successfully integrated with MASLWR, it could be adapted to run with the APEX facility at Oregon State University to increase the complexity of the model. After APEX integration, the next goal is to integrate RAPSS-STA into a regulated control room, namely the Oregon State University TRIGA reactor. While the TRIGA reactor is fairly simple, and will most likely yield fairly uninteresting results, it will serve as another credential to move to the next phase: integration into a simulated control room of a real plant. Similar steps will be taken to integrate RAPSS-STA into the simulated control room, as were taken to integrate it into the MASLWR, APEX, and TRIGA facilities. The final step, and the end-goal of the project, is RAPSS-STA control room integration for STA and unit supervisor decision support.

10.2.2 Future Work RAPSS-EOC

RAPSS-EOC is a much younger program than RAPSS-STA, and still has plenty of potential left to be realized. In the near term, RAPSS-EOC should be connected to a standardized radiological plume modeling program such as GENII or RASCAL. This is problematic because neither of these programs were designed to be used in a high-performance and UNIX environment. Recent efforts have focused on using an emulator-style program, Wine, for UNIX, which has the potential to run programs compiled for

DOS/Windows in a UNIX environment. While the RAPSS team has had some success with this, parts of the program written in Visual Basic would not function due to missing libraries. This became the challenge of using Wine. When a program is compiled for Windows, it depends on many .dll and .ocx files that are buried deep in the Windows system. To run one of these programs requires accounting for every single dependency, and creating the appropriate paths and directories (e.g., C:/Windows/system32/..) for the programs to function correctly.

What appears most feasible at this point is accessing a version of GENII compiled for UNIX. While there are rumors of a UNIX version of GENII floating around in Europe, for RAPSS-EOC applications, GENII will most likely have to be rebuilt in a UNIX environment with help from the original development team.

10.2.3 Generalizing RAPSS

One of the most ambitious future goals of the RAPSS team is to generalize the architecture, so other researchers, with a minimal amount of overhead can apply the RAPSS methodology to a system of their choosing. The secret is to rewrite RAPSS to contain connectors for inputs, outputs, and the modeling code (Figure 10.1).

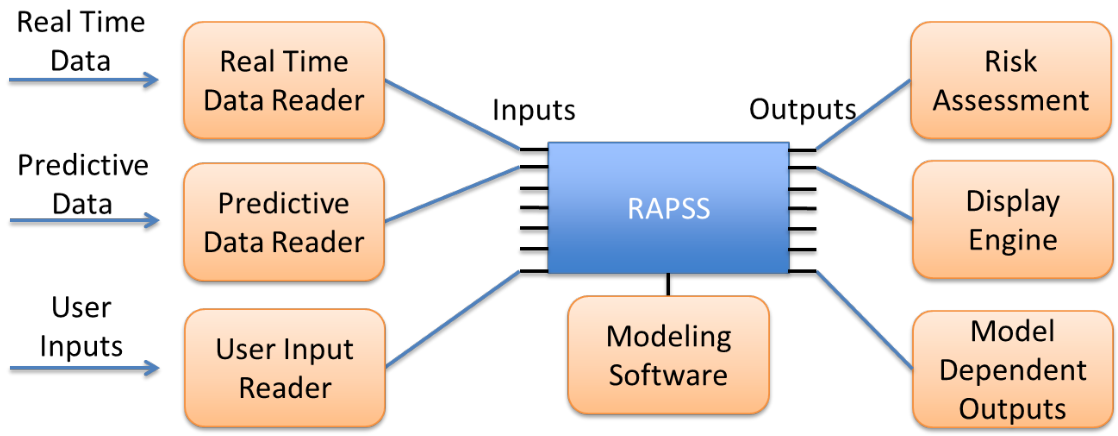


Figure 10.1 An illustration of a future RAPSS configuration that allows a researcher to apply RAPSS to other situations without significant rewriting of the code

There still would require some system-specific modules to be written, such as a real time data reader. This is the algorithm that fetches the data from the system. In the case of RAPSS-EOC, this was the wind data from remote automated weather stations; for RAPSS-STA, this was the algorithm that read the data in real time from the thermal hydraulic test facility. The predictive data reader is how one obtains probabilities for the future scenarios. In RAPSS-EOC, this was the wind rose data, and for RAPSS-STA, this was the fault tree processor, LiteFTA. For each system, the user input would need to change as well, depending on the needs of the modeling code, among many other factors.

On the output side, the process of risk assessment is different for every system. For example, pressure exceeding a threshold would be risky in a nuclear power plant, but would be useless in an emergency operation center. The display is also an important factor that the user of generalized RAPSS would be responsible for. This would change based on the needs of the user and scenario. For example, it is appropriate to display a polar grid of radioactive contamination for RAPSS-EOC, but would be useless for a shift technical advisor monitoring core temperature.

Once RAPSS has been generalized, there are limitless possibilities of applications. Recent discussions have yielded jet engine failure prediction and power grid modeling as promising systems for the next generation of RAPSS.

For the jet engine system, it is understood that the jet engine industry collects real-time data on their fleet of turbine engines. While they monitor this data, there is little ability beyond traditional failure/time probabilities to “run-ahead” and make future predictions about the state of the engine. Doing so would not only increase the safety of the engines, by requiring service before malfunctions happen, but also save money, by allowing perfectly functioning engines, to continue running past their scheduled maintenance if it is deemed necessary.

Power grid modeling is very similar to the operation of a nuclear power plant in many key ways. Both are highly complex systems with human operators. The decisions the senior operators make in both cases are largely based on experience with limited real-time decision support. With the added challenges from intermittent energy sources, there is a significant need for robust decision support and failure prediction software for power grid operators.

Other applications include fossil fuel power plants, gas pipeline systems, telecommunication systems, aviation networks, subway/train networks, manned space flight, financial markets, social diffusion event prediction, or virtually anything that can be sampled and modeled faster-than-real-time.

Bibliography

Aldrich, D. C., Sprung, J. L., Alpert, D. J., Diegert, K., Ostmeyer, R. M., Ritchie, L. T., & Strip, D. R. (1982). Technical Guidance for Siting Criteria Development, NUREG/CR-2239, SAND81-1539. Sandia National Laboratories.

Allison, C. M., & Hohorst, J. K. (2008). Role of RELAP/SCDAPSIM in Nuclear Safety. Presented at the International Topical Meeting of Safety of Nuclear Installations, Dubrovnik, Croatia: TOPSAFE.

Andreasen, M. M. (2008). Non-linear DSGE Models, The Central Difference Kalman Filter, and The Mean Shifted Particle Filter. Center for Research in Econometric Analysis of Time Series- 33.

Apostolakis, G. E. (2004). How Useful is Quantitative Risk Assessment? *Risk Analysis*, *24*(3), 515–520.

Ariely, D., & Zakay, D. (2001). A timely account of the role of duration in decision making. *Acta Psychologica (in English)*, *108*, 187–207.

Atomic Energy and Alternative Energies Commission (CEA) Website. (2011). CATHARE : Advanced Safety Code for Pressurized Water Reactors (PWR). Retrieved October 7, 2011, from http://www-cathare.cea.fr/scripts/home/publigen/content/templates/show.asp?L=EN&P=134

Ayyub, B. M. (2003). *Risk Analysis in Engineering and Economics*. Bpca Ratpm, Florida: CRC Press LLC.

Bayes, T. (1763). An Essay towards solving a Problem in the Doctrine of Chances. *Philosophical Transactions of the Royal Society of London*, *53*, 370–418.

Bengtsson, T., Snyder, C., & Nychka, D. (2002). A nonlinear filter that extends to high dimensional systems. National Center for Atmospheric Research.

Bengtsson, T., Snyder, C., & Nychka, D. (2003). Toward a nonlinear ensemble filter for high-dimensional systems. *Journal of Geophysical Research*, *108*(D24).

Borg, I., & Groenen, P. (2005). Modern Multidimensional Scaling: theory and applications (2nd ed., pp. 207–212). New York, NY: Springer-Verlag.

Burgers, G., Jan van Leeuwen, P., & Evensen, G. (1998). Analysis scheme in the ensemble Kalman filter. *Monthly Weather Review*, *126*(6), 1719–1724.

Burkardt, J., & Cliff, G. (2009, September 9). *MATLAB Parallel Computing. FDI Fall Short Course: Introduction to Parallel MATLAB*. Virginia Tech.

Cember, H., & Johnson, T. E. (2009). *Introduction to Health Physics* (4th ed.). New York, NY: The McGraw-Hill Companies.

Chang, Y. H., & Mosleh, A. (2007). Cognitive modeling and dynamic probabilistic simulation of operating crew response to complex system accidents - Part 2: IDAC performance influencing factors model. *Reliability Engineering & System Safety*, *92*, 1014–1040.

Cheng, H. S., & Rohatgi, U. S. (1996). RAMONA-4B Code for BWR Systems and Analysis BNL-NUREG-63265. Brookhaven National Laboratory.

Cheng, Y. (1995). Mean Shift, Mode Seeking, and Clustering. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *17*(8), 790–799.

Christensen-Szalanski, J. J., Beck, D. E., Christensen-Szalanski, M., & Koepsell, T. D. (1983). Effects of Expertise and Experience on Risk Judgments. *Journal of Applied Psychology*, *68*(2), 278–284.

Clemen, R. (1997). *Making Hard Decisions: An Introduction to Decision Analysis* (2nd ed.). Duxbury.

CNN. (2003, November 26). “Master” and “slave” computer labels unacceptable, officials say. *CNN Technology*. Los Angeles, California.

Cojazzi, G. (1996). The DYLAM approach for the dynamic reliability analysis of systems. *Reliability and Safety Analysis of Dynamic Process Systems*, *52*(3), 279–296.

Coyne, K. (2009). *A Predictive Model of Nuclear Power Plant Crew Decision-Making and Performance in a Dynamic Simulation Environment* (PhD Dissertation). University of Maryland.

Coyne, K., & Mosleh, A. (2009). *Dynamic PRA Approach for the Prediction of Operator Errors*. Presented at the Center for Risk and Reliability, University of Maryland, College Park, MD.

Dawes, R. M. (1998). *Behavioral decision making and judgment* (4th ed.). Boston, MA: McGraw-Hill.

Dougherty, M. R. P., Gettys, C. F., & Thomas, R. P. (1997). The Role of Mental Simulation in Judgments of Likelihood. *Organizational Behavior and Human Decision Processes*, *70*(2), 135–148.

Drottz-Sjoberg, B.-M., & Persson, L. (1993). Public Reaction to Radiation: Fear, Anxiety, or Phobia? *Health Physics*, *64*(3), 223–231.

Du, J., Mullen, S. L., & Sanders, F. (1997). Short-Range Ensemble Forecasting of Quantitative Precipitation. *American Meteorological Society*, 2427–2459.

Dwivedy, K. K., Bhargava, D., & Hook, T. G. (2007). Significance Determination Process for Plant Condition Assessment. In *Structural Mechanics in Reactor Technology (SMiRT) 19*. Toronto, Canada.

Edland, A., & Svenson, O. (1993). *Judgment and decision making under time pressure: Studies and findings*. New York, NY: Plenum Press.

Edwards, J. A., Snyder, F. J., Allen, P. M., Makinson, K. A., & Hamby, D. M. (2012). Decision Making for Risk Management: A Comparison of Graphical Methods for Presenting Quantitative Uncertainty. *Risk Analysis*.

Edwards, W. (1962). Dynamic decision theory and probabilistic information processing. *Human Factors*, *4*(14).

Evensen, G. (1994). Sequential data assimilation with a nonlinear quasi-geostrophic model using Mote Carlo Methods to forecast error statistics. *Journal of Geophysical Research*, *99*(C5), 143–162.

Evensen, G. (2003). The Ensemble Kalman Filter: theoretical formulation and practical implementation. *Ocean Dynamics*, *53*, 343–367.

Fleming, K. N., Unwin, S. D., Kelly, D., Lowry, P. P., Toloczko, M. B., Layton, R. F., … Heasler, P. G. (2010). Treatment of Passive Component Reliability in Risk-Informed Safety Margin Characterization: FY2010 Report. Idaho National Lab.

Fudenberg, D., & Levine, D. (2009, October 14). *Self Control, Risk Aversion, and the Allais Paradox*. Retrieved from http://www.dklevine.com/econ506/allais-slides.pdf

Galvin, M. R., & Bowser, J. C. (2010). *OSU MASLWR Test Facility Modification Description Report IAEA Contract Number USA-13386*. Oregon State University.

Garrick, J. B. (2006). Warren K. Sinclair Keynote Address: Contemporary Issues in Risk-Informed decision Making on the Disposition of Radioactive Waste. *Health Physics Journal*, *91*(5), 430–439.

Gerhardt, H., Biele, G., Uhlig, H., & Heekeren, H. (2011). Cognitive Load Increases Risk Aversion. In *Deutsche Forschungsgemeinschaft*. Presented at the Research Center 649 "Economic Risk."

Gertman, D., Blackman, H., Marble, J., Byers, J., & Smith, C. (2005). The SPAR-H Human Reliability Analysis Method, NUREG/CR-6883. Idaho National Lab.

Ghile, Y. B., & Schulze, R. (2009). Evaluation of Three Numerical Weather Prediction Models for Short and Medium Range Agrohydrological Applications. *Water Resource Management*, *24*, 1005–1028.

Green, A. E. S., Singhal, R. P., & Venkateswar, R. (1980). Analytic Extensions of the Gaussian Plume Model. *Journal of the Air Pollution Control Association*, *30*(7), 773–776.

Hakobyan, A. (2006). *Severe Accident Analysis Using Dynamic Accident Progression Event Trees* (PhD Dissertation). The Ohio State University.

Hakobyan, A., Aldemir, T., Denning, R., Dunagan, S., Kunsman, D., Rutt, B., & Catalyurek, U. (2008). Dynamic generation of accident progression event trees.

Hess, S. M., Dinh, N., Gaertner, J. P., & Szilard, R. (2009). Risk-Informed Safety Margin Characterization. Presented at the Proceedings of the 17th International Conference on Nuclear Engineering, Brussels, Belgium: The Idaho National Lab.

Hofer, E., Kloos, M., Krzykacz-Hausmann, J., Peschke, J., & Sonnenkalb, M. (2004). Dynamic Event Trees for Probabilistic Safety Analysis. Gesellschaft fur Anlagen und Reacktorsicherheit (GRS) (in English).

Horng, T.-C. (2004). *MIDAS: Minor Incident Decision Analysis Software* (Masters Thesis). Massachusetts Institute of Technology, Cambridge Massachusetts.

Hsueh, K.-S., & Mosleh, A. (1996). The development and application of the accident dynamic simulator for dynamic probabilistic risk assessment of nuclear power plants. *Reliability Engineering and System Safety*, *52*, 297–314.

Huntley, J., & Miller, E. (2009). Using DSGE Models. Congressional Budget Office.

Ibrekk, H., & Morgan, M. G. (1987). Graphical communication of uncertain quantities to nontechnical people. *Risk Analysis*, *7*(4), 519–529.

Idaho National Lab. (2003). RELAP5-3D Code Manual, Vol 1-3. NUREG/CR-5535/Rev1.

Innovative Systems Software (ISS) Website. (2011). SCDAP Development and Training Program (SDTP). Retrieved October 7, 2011, from http://www.sdtp.org/software.html#top

INPO. (2004). Principles for a Strong Nuclear Safety Culture. Institute of Nuclear Operations.

Johnson, B. B., & Slovic, P. (1995). Presenting uncertainty in health risk assessment: Initial studies of its effects on risk perception and trust. *Risk Analysis*, *15*, 485–494.

Kalman, R. E. (1960). A New Approach to Linear Filtering and Prediction Problems. *Journal of Basic Engineering*, *82*(D), 35–45.

Kaplan, S., & Garrick, J. (1981). On the Quantitative Definition of Risk. *Risk Analysis*, *1*(1), 11–28.

Keeney, R., & Railla, H. (1993). *Decisions with Multiple Objectives*. New York, NY: Cambridge University Press.

Keller, T. S., & Reese, S. R. (2009). Going from HEU to LEU: Conversion of the Oregon State TRIGA Reactor. In *RERTR 2009*. Presented at the 31st International Meeting on Reduced Enrichment for Research and Test Reactors, Beijing, China.

Kirschenbaum, S. S., & Arruda, J. E. (1994). Effects of graphic and verbal probability information on command decision making. *Human Factors: The Journal of the Human Factors and Ergonomics Society*, *36*(3), 406–418.

Kleinmuntz, B. (1990). Why we still use our heads instead of formulas: toward an integrative approach. *Psychological Bulletin*, (107), 296–310.

Kloos, M., & Peschke, j. (2008). Consideration of human actions in combination with the probabilistic dynamics method Monte Carlo dynamic event tree. *Proceedings of the IMechE: Risk and Reliability*, *222*, 303–313.

Knaus, J. (2011). Package “snowfall”: Easier cluster computing (based on snow) (User Manual). Comprehensive R Archive Network (CRAN).

Knaus, J., & Porzelius, C. (2009). Tutorial: Parallel computing using R package snowfall.

Knaus, J., Porzelius, C., Binder, H., & Schwarzer, G. (2009). Easier Parallel Computing in R with Snowfall and sfCluster. *The R Journal: Contributed Research Articles*, *1*(1), 54–59.

Lewis, H. W., Budnitz, R. J., Kouts, H. J. C., Loewenstein, W. B., Rowe, W. D., Von Hippel, F., & Zachariasen, F. (1978). *Risk Assessment Review Group Report to the U.S. Nuclear Regulatory Commission* (Ad Hoc Risk Assessment Review Group).

Li, B., Li, M., Chen, K., & Smidts, C. (2006). Integrating Software into PRA: A Software-Related Failure Mode Taxonomy. *Risk Analysis*, *26*(4), 997–1012.

Li, B., Li, M., & Smidts, C. (2005). Integrating Software into PRA: A Test-Based Approach. *Risk Analysis*, *25*(4), 1061–1077.

Lichtenstein, S., Slovic, P., Fischhoff, B., Laymen, M., & Combs, B. (1978). Judged Frequency of Lethal Events. *Journal of Experimental Social Psychology: Human Learning and Memory*, (6), 551–578.

Loewenstein, G. F., Weber, E. U., Hsee, C. K., & Welch, N. (2001). Risk as Feelings. *Psychological Bulletin*, *127*(2), 267–286.

Lorenz, E. N. (1963). Deterministic Nonperiodic Flow. *Journal of Atmospheric Sciences*, *20*, 130–141.

Mackenzie, D. (2003). Ensemble Kalman Filters Bring Weather Models Up to Date. Society for Industrial and Applied Mathematics (SIAM) News.

Mai, A. T., & Luo, H. (2011). *OSU MASLWR Test Facility Quick Look Report* (No. OSU-MASLWR-QLR-SP3). Corvallis, OR: Oregon State University.

Mandel, J. (2006). Efficient Implementation of the Ensemble Kalman Filter. Center for Computational Mathematics Reports. No. 231.

Mandelli, D, Yilmaz, A., & Aldemir, T. (2011). *Clustering on Manifolds: An Application to Scenario Analysis*. Presented at the 2011 ANS Winter Meeting and Nuclear Technology Expo, “The Status of Global Nuclear Deployment”, Washington, D.C.

Mandelli, Diego. (2011). *Scenario Clustering and Dynamic Probabilistic Risk Assessment* (PhD Dissertation). The Ohio State University.

Martin, J. E. (2006). *Physics for Radiation Protection* (2nd ed.). Weinheim, Germany: Wiley-VCH.

McGuire, S. A., Ramsdell, J. V. J., & Athey, G. F. (2007). *RASCAL 3.0.5: Description of Models and Methods* (No. NUREG-1887). Washington, DC: U.S. Nuclear Regulatory Commission (USNRC).

Medvedev, G. (1989). Chernobyl Notebook (in English). *Novy Mir*, *6*, 3–108.

Mercurio, D., Podofillini, L., Zio, L., & Dang, V. (2009). Identification and classification of dynamic event tree scenarios via possibilistic clustering: application to a steam generator tube rupture event. *Accident Analysis and Prevention*, *41*, 1180–1191.

Mesina, G., Hykes, J., & Guillen, D. (2007). Streamlining of the RELAP5-3D Code INL/CON-07-12089. Presented at the The 12th International Topical Meeting on

Nuclear Reactor Thermal Hydraulics (NURETH-12), Sheraton Station Square, Pittsburgh, Pennsylvania, U.S.A.

Modarres, M., Kaminskiy, M., & Krivtsov, V. (2010). *Reliability Engineering and Risk Analysis: A Practical Guide* (2nd ed.). Boca Raton, Florida: CRC Press.

Modro, M. S., Fisher, J., Kavan, W., Babka, P., Reyes, J., Groome, J., & Wilson, G. (2002). Generation-IV Multi-Application Small Light Water Reactor (MASLWR) NEEL/CON-02-00017. In *Proceedings of ICONE 10*. Presented at the Tenth International Conference on Nuclear Energy, Arlington, Virginia, USA: Idaho National Engineering and Environmental Laboratory (INEEL).

Moler, C. (2007). Cleve's Corner - Parallel Matlab: Multiple Processors and Multiple Cores. *The MathWorks News*.

Murray, C. (2007). *Overview of TRACE V5.0*. Presented at the Regulatory Information Conference.

Napier, B. A. (2012). *GENII Version 2 Users' Guide* (No. PNNL-14583). Richland, WA.

Nourgaliev, R. R., Bui, A. V., Ougouag, A. M., Cogliati, J. J., Gleicher, G., Phillips, J. H., … Dinh, N. T. (2011). *Summary Report on NGSAC (Next-Generation Safety Analysis Code) Development and Testing* (No. 412.09). Idaho Falls, ID: Idaho National Lab.

NRC Website. (2011a). NRC: Computer Codes. Retrieved October 5, 2011, from http://www.nrc.gov/about-nrc/regulatory/research/comp-codes.html

NRC Website. (2011b). NRC: Probabilistic Risk Assessment (PRA). Retrieved October 6, 2011, from http://www.nrc.gov/about-nrc/regulatory/risk-informed/pra.html

NUCE. (2002). PSA Glossary. PSAM 6: International Conference on Probabilistic Safety Assessment and Management.

Office of Technology Assessment. (1984). Nuclear Power in an Age of Uncertainty. Chapter 8: Public Attitudes Toward Nuclear Power.

OpenFTA Website. (2012). OpenFTA.com. Retrieved August 8, 2012, from http://www.openfta.com/

OpenMP Website. (2011). OpenMP.org. Retrieved October 3, 2011, from http://openmp.org/

Osei, E. K., Amoh, G. E. A., & Schandorf, C. (1997). Risk Ranking by Perception. *Health Physics*, *72*(2), 195–203.

Pagani, L., Smith, C. L., & Apostolakis, G. E. (2004). Making Decision for incident management in nuclear power plants using probabilistic safety assessment. *Risk Decision and Policy*, *9*(4), 271–295.

Pozo, J. T., Pappenberger, F., Salamon, P., Bogner, K., Burek, P., & De Roo, A. (2010). The state of the art of flood forecasting – Hydrological Ensemble Prediction Systems. Presented at the European Meeting of Statisticians Annual Meeting, Zurich, Switzerland.

Raiffa, H., & Schlaifer, R. (1968). *Applied Statistical Decision Theory*. Cambridge, Massachusetts: MIT Press.

Ramsey, F., & Schafer, D. (2002). *The Statistical Sleuth* (2nd ed.). Pacific Grove, CA: Duxbury.

Ribeiro, M. I. (2004). Kalman and Extended Kalman Filters: Concept, Derivation and Properties. Instituto Superior Tecnico, Lisboa Portugal.

Roebber, P. J., Schultz, D. M., Colle, B., & Stensrud, D. J. (2004). Toward Improved Prediction: High-Resolution and Ensemble Modeling Systems in Operations. *American Meteorological Society*, *19*, 936–949.

Sandia National Lab. (1998). Code Manual for MACCS2: Volume 1, User's Guide NUREG/CR-6613. U.S. Nuclear Regulatory Commission (USNRC).

Sandia National Lab. (2000). MELCOR Computer Code Manual: Primer and User's Guide- NUREG/CR-6119. U.S. Nuclear Regulatory Commission (USNRC).

Schapira, M. M., Nattinger, A. B., & McHorney, C. A. (2001). Frequency or probability? A qualitative study of risk communication formats used in health care. *Medical Decision Making*, *21*, 459–467.

Schmidberger, M., Morgan, M., Eddelbuettel, D., Yu, H., Tierney, L., & Mansmann, U. (2009). State of the Art in Parallel Computing with R. *Journal of Statistical Software*, *31*(1), 1–27.

Schultz, E. E., & Johnson, G. L. (1988). User Interface Design in Safety Parameter Display Systems: Directions for Enhancement. Presented at the Fourth IEEE Conference on Human Factors and Power Plants, Monterey, California: Lawrence Livermore National Laboratory.

Schwarz, D. R., & Howell, W. C. (1985). Optional stopping performance under graphic and numeric CRT formatting. *Human factor*, *27*(4), 433–44.

Siu, N. (1994). Risk assessment for dynamic systems: An overview. *Reliability Engineering and System Safety*, *43*(1), 43–73.

Slovic, P. (1995). The Construction of Preference. *American Psychologist*, *50*(5), 364–371.

Smith, C., Knudsen, J., Kvarfordt, K., & Wood, T. (2008). Key attributes of the SAPHIRE risk and reliability analysis software for risk-informed probabilistic applications. *Reliability Engineering and System Safety*, *93*, 1151–1164.

Smith, C. L. (2002). *Risk-Informed Incident Management for Nuclear Power Plants* (PhD Dissertation). Massachusetts Institute of Technology.

Snir, M., Otto, S., Huss-Lederman, S., Walker, D., & Dongarra, J. (1996). *MPI: The Complete Reference*. Cambridge Massachusetts: The MIT Press.

Snyder, C., & Zhang, F. (2003). Assimilation of Simulated Doppler Radar Observations with an Ensemble Kalman Filter. *American Meteorological Society*, 1663–1677.

Soffer, L., Burson, S. B., Ferrell, C. M., Lee, R. Y., & Ridgely, J. N. (1995). Accident Source Terms for Light-Water Nuclear Power Plants, NUREG-1465. U.S. Nuclear Regulatory Commission (USNRC).

Spore, J. W., Weaver, W. L., Shumway, R. W., Giles, M. M., Phillips, R. E., Mohr, C. M., … Fischer, S. R. (1981). TRAC-BD1 - Transient Reactor Analysis Code for Boiling-Water Systems. Idaho National Engineering Laboratory (INEL).

Stensrud, D. J., Bao, J.-W., & Warner, T. (2000). Using Initial Condition and Model Physics Perturbations in Short-Range Ensemble Simulations of Mesoscale Convective Systems. *American Meteorological Society*, 2077–2107.

Stone, E. R., Yates, J. F., & Parker, A. M. (1997). Effects of numerical and graphical displays on professed risk-taking behavior. *Journal of Experimental Psychology: Applied*, *3*, 243–256.

Strid, I., & Walentin, K. (2008). Block Kalman filtering for large-scale DSGE models. Sveriges Riksbank Working Paper Series 224.

Stutzke, M., & Smidts, C. (2001). A Stochastic Model of Human Error During Software Development. *IEEE Transactions on Reliability*, *50*(2), 184–193.

Sur, S., Koop, M., & Panda, D. (2006). High-Performance and Scalable MPI over InfiniBand with Reduced Memory Usage: An In-Depth Performance Analysis. In *SC ’06 Proceedings of the 2006 ACM/IEEE conference on Supercomputing*.

Swain, A. D., & Guttman, H. E. (1983). Handbook of human reliability analysis with emphasis on nuclear power plant applications. NUREG/CR-1278, Washington D.C.

The R Project Webpage. (2011). The R Project for Statistical Computing. Retrieved October 3, 2011, from http://www.r-project.org/

Tierney, L., Rossini, A. J., & Li, N. (2003, March 13). *Simple Parallel Statistical Computing in R*. University of Iowa.

Tierney, L., Rossini, A. J., Li, N., & Sevcikova, H. (2011). Package “snow”: Simple Network of Workstations (user manual). Comprehensive R Archive Network (CRAN).

Travers, W. D. (2000). Risk-Informing Special Treatment Requirements SECY-00-0194. Policy Issue Information Memo to the USNRC.

Tumer, I. Y., & Smidts, C. (2011). Integrated Design-Stage Failure Analysis of Software-Driven Hardware Systems. *IEEE Transactions on Computers*, *60*(8), 1072–1084.

Tversky, A., & Kahneman, D. (1974). Judgment under Uncertainty: Heuristics and Biases. *Science*, *185*(4157), 1124–1131.

US Atomic Energy Commission (USAEC). (1957). Theoretical Possibilities and Consequences of Major Accidents in Large Nuclear Power Plants (WASH-740). United States Energy Research and Development Administration.

US Forest Service and Bureau of Land Management. (2012). RAWS USA Climate Archive State Selection Map. Retrieved February 19, 2013, from http://www.raws.dri.edu/index.html

US Nuclear Regulatory Commission (USNRC). (1975). Reactor Safety Study - An Assessment of Accident Risks in U.S. Commercial Nuclear Power Plants, WASH-1400, (NUREG-75/014).

US Nuclear Regulatory Commission (USNRC). (1983). Discussion Paper on Safety Goals for Nuclear Power Plants, NUREG-880.

US Nuclear Regulatory Commission (USNRC). (1986). Safety Goals for the Operations of Nuclear Power Plants; Policy Statement; Republication (51FR30028).

US Nuclear Regulatory Commission (USNRC). (1990). Severe Accident Risks: An Assessment for Five U.S. Nuclear Power Plants, NUREG-1150.

US Nuclear Regulatory Commission (USNRC). (2002a). NRC Regulations (10CFR) Part 100 -- Reactor Site Criteria, 10CFR100.11.

US Nuclear Regulatory Commission (USNRC). (2002b). Regulatory Guide 1.174 - An approach for Using Probabilistic Risk Assessment in Risk-Informed Decisions on Plant-Specific Changes to the Licensing Basis.

US Nuclear Regulatory Commission (USNRC). (2003a). NRC Regulations (10CFR) Part 50 -- Domestic Licensing of Production and Utilization Facilities. Sec. 50.109 Backfitting.

US Nuclear Regulatory Commission (USNRC). (2003b). Domestic Licensing of Production and Utilization Facilities 10CFR50. U.S. Government Printing Office.

US Nuclear Regulatory Commission (USNRC). (2009). NRC Regulations (10CFR) Part 52 -- Licenses, Certifications, and Approvals for Nuclear Power Plants.

US Nuclear Regulatory Commission (USNRC). (2011a). State-of-the-Art Reactor Consequence Analyses (SOARCA).

US Nuclear Regulatory Commission (USNRC). (2011b). Risk-Informed Regulation. *NRC: Glossary*. Retrieved from http://www.nrc.gov/reading-rm/basic-ref/glossary/risk-informed-regulation.html

US Nuclear Regulatory Commission (USNRC). (2011c). Resolution of Generic Safety Issues: Appendix G. Generic Issues Program Current and Historical Procedures (NUREG-0933, Main Report with Supplements 1–33). Retrieved from http://www.nrc.gov/reading-rm/doc-collections/nuregs/staff/sr0933/appendices/appg.html#sec-1

USNRC. (2004). Regulatory Analysis Guidelines of the U.S. Nuclear Regulatory Commission NUREG/BR-0058.

Van Dijk, E., & Zeelenberg, M. (2003). The discounting of ambiguous information in economic decision making. *Journal of Behavioral Decision Making*, *16*(5), 341–352.

Von Neumann, J., & Morgenstern, O. (1944). *Theory of Games and Economic Behavior*. Princeton University Press.

Weary, G., Vaughn, L., Stewart, B., & Edwards, J. A. (2006). Adjusting for the correspondence bias: Effects of causal uncertainty, cognitive busyness, and causal strength of situational information. *Journal of Experimental Social Psychology*, (42), 87–94.

Weil, R., & Apostolakis, G. E. (2001). A methodology for the prioritization of operating experience in nuclear power plants. *Reliability Engineering & System Safety*, *74*(1), 23–42.

Welch, G., & Bishop, G. (1997). SCAAT: Incremental Tracking with Incomplete Information. Association for Computing Machinery (ACM).

Wickens, C. D., Gempler, K., & Morphew, M. E. (2000). Workload and reliability of predictor displays in aircraft traffic avoidance. *Transportation Human Factors Journal*, *2*(2), 99–126.

Wilson, J. R. (1993). SHEAN (Simplified Human Error Analysis code) and automated THERP. Presented at the GOCO database meeting, Augusta, GA: Westinghouse Idaho Nuclear Company.

Youngblood, R. W. (2011, August 8). Post RAPS Presentation Discussion at Idaho National Lab.

Youngblood, R. W., Mousseau, V. A., Kelly, D. L., & Dinh, T.-N. (2010a). Risk-Informed Safety Margin Characterization (RISMC): Integrated Treatment of Aleatory and Epistemic Uncertainty in Safety Analysis. Presented at the 8th international Topical Meeting on Nuclear Thermal-Hydraulics, Operation and Safety (NUTHOS-8), Shanghai, China.

Youngblood, R. W., Mousseau, V. A., Kelly, D. L., & Dinh, T.-N. (2010b, October 10). *Risk-Informed Safety Margin Characterization (RISMC): Integrated Treatment of Aleatory and Epistemic Uncertainty in Safety Analysis*. Presented at the 8th international Topical Meeting on Nuclear Thermal-Hydraulics, Operation and Safety (NUTHOS-8), Shanghai, China.

Youngblood, R. W., Nourgaliev, R. R., Kelly, D. L., Smith, C. L., & Dinh, T.-N. (2011). Heartbeat Model for Component Failure Time in Simulation of Plant Behavior. In *ANSA PSA 2011*. Presented at the International Topical Meeting on Probabilistic Safety Assessment and Analysis, Wilmington, NC: American Nuclear Society.

Zhou, H., Gomez-Hernandez, J. J., Hendricks Franssen, H.-J., & Li, L. (2011). An Approach to Handling Non-Gaussianity of Parameters and State Variables in Ensemble Kalman Filtering. Advances in Water Resources.

Zhu, D., Chang, Y. H., & Mosleh, A. (2008). *The Use of Distributed Computing for Dynamic PRA: The ADS Approach*. Presented at the International Conference on Probability Safety Assessment and Management (PSAM 9), Hong Kong, China.

Zio, E., & Maio, F. D. (2009). Processing dynamic scenarios from a reliability analysis of a nuclear power plant digital instrumentation and control system. *Annals of Nuclear Energy*, *36*, 1386–1399.

Appendices

A. Appendix A: RAPSS-STA Source Code

This Appendix contains 12 sections. Appendices A.1-A.4 are C++ main and header files written by Kevin Makinson. Appendices A.5-A.9 are R scripts written by Kevin Makinson. Appendices A.10 and A.11 contain the C++ main and header files for the mean shift algorithm, originally written by Diego Mandelli, and modified by Kevin Makinson. Appendix A.12 is a sample input file for RAPSS-STA. Readers are encouraged to read through Appendix B, while referencing Appendix A. Appendix B contains valuable explanations of the nuts and bolts of RAPSS-STA, Specifically Appendix B.6.1contains an explanation of the Automated Linear Approximation Interval Sequencer, which was crucial to the success of principal components analysis.

A.1. RAPSmain.cpp Source Code

```

// 3/6/12
// Oregon State University
// Written by Kevin Makinson
// This is the main control structure for RAPSS

#include <iostream>
#include <string>
#include <fstream>
#include "BloodAndGuts.h"
#include "CycleR5.h"
#include <stdlib.h> //for system calls in UNIX
#include <stdio.h> //for removing shell files.

//R5 parameters 100 cards
string R5Input;
string R5Output;
string R5H2oData;
double EndTime;
string MinTimeStep;
double MinTimeStepTemp;
double MaxTimeStep;
int CtlMode;
int MinEdit;
int MajEdit;
int RstFreq;
//R, PCA, MSA parameters 200 cards
double PCAthreshold;    //for PCA - how much variance do you want to capture?
double BW;              //for MSA - How big do you want your clusters?
string libloc; //for R library files to download into
string Rrepos;
//RAPS parameters 300 cards
string R5ExePath;
string InDir;
bool dataOut;
```

```
int requestTh=1;
double TStep;//troubleshooting
double T1Step;
void RAPSinputFile(string RAPSinput); //function that reads input file, defined below
string FTAdir;
string FTAfileName;
string realTimeSimData;
int numOfCutSets;
vector <string> stateVarTripNames;
vector <string> stateVarCodes;
vector <string> stateVarEquiv;
vector <double> yellowTripThresh;
vector <double> redTripThresh;
vector <double> FTApars;
vector<size_t> positions;
size_t pos;
double temp;

int main () {
    const char *InitShOutput = "InitRun.sh";
    const char *RstShOutput = "rst.sh";
    string ProbType = "restart";
    string ProbOpt = "transnt";
    string R5RstData = "rst.r";
    string answer = "y";
    string RAPSinput;
    string dos2unixInput;

    system(ChangeFont(2));
    cout << "Welcome to RAPSS" << endl << "Written by Kevin Makinson" << endl
    << "Last compiled on " << __DATE__ << " at " << __TIME__ << endl
    << "Begin run? (y/n)" << endl;
    cin >> answer;

    if ((answer == "n") || (answer == "N") || (answer == "no") || (answer=="No")) {
        cout << "Thank you for running RAPSS" << endl;
        system(ResetFont());
        return 0;
```

```
}
while ((answer != "n") && (answer != "y") && (answer != "Y") && (answer != "N") &&
    (answer != "yes") && (answer != "no") && (answer != "Yes") && (answer != "No")) {
            cout << "You did not enter a \"y\" or an \"n\"!" << endl;
            cout << "Begin run? (y/n)" << endl;
            cin >> answer;
        }

cout << "Please type the name of RAPSS input file (e.g., input.raps): ";
cin >> RAPSinput;
cout << endl;

ifstream fin(RAPSinput.c_str());
while (!fin) { //added the break statement
    cout << "File does not exist!" << endl
        << "Please carefully type the name of RAPS input file, or type \"exit\": ";
    cin >> RAPSinput;
    cout << endl;
    ifstream fin(RAPSinput.c_str());
    if (RAPSinput=="exit") {
        cout << "Thank you for running RAPS" << endl;
        system(ResetFont());
        fin.close();
        return 0;
    }
    else if (fin.good()) {break;} //added because the "while" statement doesn't work
}
fin.close();

RAPSinputFile(RAPSinput); //reads input file
string OutDir = (InDir + "/RAPS_data"); //Assigning Output Directory inside input directory
system(("rm -rf " + OutDir).c_str()); //this removes it if it already exists (to overwrite)
string CreateDataDir= ("mkdir -p " + OutDir);
system(CreateDataDir.c_str()); //creates a directory for data output
    CycleR5(answer, ProbType, ProbOpt, EndTime, MinTimeStep, MaxTimeStep, CtlMode, MinEdit,
        MajEdit, RstFreq, R5ExePath, R5RstData, R5H2oData, InitShOutput, RstShOutput, Rrepos,
        libloc, PCAthreshold, BW, dataOut, InDir, OutDir, R5Input, R5Output, requestTh,
        TStep, T1Step, FTAdir, FTAfileName, numOfCutSets, stateVarTripNames, stateVarCodes,
```

```
            stateVarEquiv, yellowTripThresh, redTripThresh, FTApars, realTimeSimData);

        cout << "Thank you for running RAPS" << endl;

    system(ResetFont());
    remove("ChangeFont.sh");
    remove("ResetFont.sh");
    remove("runFTA.sh");

    return 0;
}


void RAPSinputFile(string RAPSinput) {
    //variables for this program
    int cardNo;
    vector <string> inputVec;
    inputVec = LoadFile(RAPSinput);

    for (unsigned int i=0; i<(inputVec.size()); i++) {
        if (inputVec[i][0] != '*') {
            istringstream(string(inputVec[i].begin(), inputVec[i].begin()+3)) >> cardNo;
            switch (cardNo) {
                case 101: //R5 Parameters: Cards 100-199
                    R5Input = string((inputVec[i].begin()+4), inputVec[i].end());
                    break;
                case 102:
                    R5Output= string((inputVec[i].begin()+4), inputVec[i].end());
                    break;
                case 103:
                    R5H2oData= string((inputVec[i].begin()+4), inputVec[i].end());
                    break;
                case 104:
                    istringstream(string((inputVec[i].begin()+4), inputVec[i].end())) >> EndTime;
                    break;
                case 105:
                    istringstream(string((inputVec[i].begin()+4), inputVec[i].end())) >>
                        MinTimeStepTemp;
```

```
                    MinTimeStep=R5SciConv(MinTimeStepTemp);
                    break;
                case 106:
                    istringstream(string((inputVec[i].begin()+4), inputVec[i].end()))
                        >> MaxTimeStep;
                    break;
                case 107:
                    istringstream(string((inputVec[i].begin()+4), inputVec[i].end())) >> CtlMode;
                    break;
                case 108:
                    istringstream(string((inputVec[i].begin()+4), inputVec[i].end())) >> MinEdit;
                    break;
                case 109:
                    istringstream(string((inputVec[i].begin()+4), inputVec[i].end())) >> MajEdit;
                    break;
                case 110:
                    istringstream(string((inputVec[i].begin()+4), inputVec[i].end())) >> RstFreq;
                    break;
                case 201://R, PCA and MSA Parameters Cards 200-299
                    istringstream(string((inputVec[i].begin()+4), inputVec[i].end()))
                        >> PCAthreshold;
                    break;
                case 202:
                    istringstream(string((inputVec[i].begin()+4), inputVec[i].end())) >> BW;
                    break;
                case 203:
                    libloc=string((inputVec[i].begin()+4), inputVec[i].end());
                    break;
                case 204:
                    Rrepos=string((inputVec[i].begin()+4), inputVec[i].end());
                    break;
                case 301: //RAPS parameters 300 cards
                    R5ExePath=string((inputVec[i].begin()+4), inputVec[i].end());
                    break;
                case 302:
                    istringstream(string((inputVec[i].begin()+4), inputVec[i].end())) >> dataOut;
                    break;
                case 303:
```

```
            InDir = string((inputVec[i].begin()+4), inputVec[i].end());
            break;
        case 304:
            istringstream(string((inputVec[i].begin()+4), inputVec[i].end()))
                >> requestTh;
            break;
        case 305:
            istringstream(string((inputVec[i].begin()+4), inputVec[i].end())) >> TStep;
            break;
        case 306:
            istringstream(string((inputVec[i].begin()+4), inputVec[i].end())) >> T1Step;
            break;
        case 307:
            FTAdir = string((inputVec[i].begin()+4), inputVec[i].end());
            break;
        case 308:
            FTAfileName = string((inputVec[i].begin()+4), inputVec[i].end());
            break;
        case 309:
            istringstream(string((inputVec[i].begin()+4), inputVec[i].end()))
                >> numOfCutSets;
            break;
        case 310:
            positions.clear();
            positions.push_back(0);
            pos = string((inputVec[i].begin()+4), inputVec[i].end()).find(" ", 0);
            if (pos==string::npos) {
                stateVarTripNames.push_back(string((inputVec[i].begin()+4),
                    inputVec[i].end()));
            } else {
                while(pos !=string::npos) {
                    positions.push_back(pos);
                    pos = string((inputVec[i].begin()+4),
                        inputVec[i].end()).find(" ", pos+1);
                }
                for (int j=0; j<positions.size(); j++) {
                    if (j==0) {
                        stateVarTripNames.push_back(
```

```
                            string((inputVec[i].begin()+4+positions[j]),
                                (inputVec[i].begin()+4+positions[j+1])));
                    } else if (j==(positions.size()-1)) {
                        stateVarTripNames.push_back(
                            string((inputVec[i].begin()+5+positions[j]),
                                (inputVec[i].end())));
                    } else {
                        stateVarTripNames.push_back(
                            string((inputVec[i].begin()+5+positions[j]),
                                (inputVec[i].begin()+4+positions[j+1])));
                    }
                }
            }
            break;
        case 311:
            positions.clear();
            positions.push_back(0);
            pos = string((inputVec[i].begin()+4), inputVec[i].end()).find(" ", 0);
            if (pos==string::npos) {
                stateVarCodes.push_back(string((inputVec[i].begin()+4),
                inputVec[i].end()));
            } else {
                while(pos !=string::npos) {
                    positions.push_back(pos);
                    pos = string((inputVec[i].begin()+4),
                    inputVec[i].end()).find(" ", pos+1);
                }
                for (int j=0; j<positions.size(); j++) {
                    if (j==0) {
                        stateVarCodes.push_back(string((inputVec[i].begin()+
                            4+positions[j]),
                            (inputVec[i].begin()+4+positions[j+1])));
                    } else if (j==(positions.size()-1)) {
                        stateVarCodes.push_back(string((inputVec[i].begin()+
                            5+positions[j]),
                            (inputVec[i].end())));
                    } else {
                        stateVarCodes.push_back(string((inputVec[i].begin()+
```

```
                    5+positions[j]),
                    (inputVec[i].begin()+4+positions[j+1])));
            }
        }
    }
    break;
case 312:
    positions.clear();
    positions.push_back(0);
    pos = string((inputVec[i].begin()+4), inputVec[i].end()).find(" ", 0);
    if (pos==string::npos) {
        stateVarEquiv.push_back(string((inputVec[i].begin()+4),
            inputVec[i].end()));
    } else {
        while(pos !=string::npos) {
            positions.push_back(pos);
            pos = string((inputVec[i].begin()+4),
            inputVec[i].end()).find(" ", pos+1);
        }
        for (int j=0; j<positions.size(); j++) {
            if (j==0) {
                stateVarEquiv.push_back(string((inputVec[i].begin()+
                    4+positions[j]),
                    (inputVec[i].begin()+4+positions[j+1])));
            } else if (j==(positions.size()-1)) {
                stateVarEquiv.push_back(string((inputVec[i].begin()+
                    5+positions[j]),
                    (inputVec[i].end())));
            } else {
                stateVarEquiv.push_back(string((inputVec[i].begin()+
                    5+positions[j]),
                    (inputVec[i].begin()+4+positions[j+1])));
            }
        }
    }
    break;
case 313:
    positions.clear();
```

```
            positions.push_back(0);
            pos = string((inputVec[i].begin()+4),
            inputVec[i].end()).find(" ", 0);
            if (pos==string::npos) {
                istringstream(string((inputVec[i].begin()+4),
                    inputVec[i].end())) >> temp;
                yellowTripThresh.push_back(temp);
            } else {
                while(pos !=string::npos) {
                    positions.push_back(pos);
                    pos = string((inputVec[i].begin()+4),
                        inputVec[i].end()).find(" ", pos+1);
                }
                for (int j=0; j<positions.size(); j++) {
                    if (j==0) {
                        istringstream(string((inputVec[i].begin()+4+positions[j]),
                            (inputVec[i].begin()+4+positions[j+1]))) >> temp;
                        yellowTripThresh.push_back(temp);
                    } else if (j==(positions.size()-1)) {
                        istringstream(string((inputVec[i].begin()+5+positions[j]),
                            (inputVec[i].end()))) >> temp;
                        yellowTripThresh.push_back(temp);
                    } else {
                        istringstream(string((inputVec[i].begin()+5+positions[j]),
                            (inputVec[i].begin()+4+positions[j+1]))) >> temp;
                        yellowTripThresh.push_back(temp);
                    }
                }
            }
            break;
        case 314:
            positions.clear();
            positions.push_back(0);
            pos = string((inputVec[i].begin()+4), inputVec[i].end()).find(" ", 0);
            if (pos==string::npos) {
                istringstream(string((inputVec[i].begin()+4),
                    inputVec[i].end())) >> temp;
                redTripThresh.push_back(temp);
```

```
    } else {
        while(pos !=string::npos) {
            positions.push_back(pos);
            pos = string((inputVec[i].begin()+4),
                inputVec[i].end()).find(" ", pos+1);
        }
        for (int j=0; j<positions.size(); j++) {
            if (j==0) {
                istringstream(string((inputVec[i].begin()+4+positions[j]),
                    (inputVec[i].begin()+4+positions[j+1]))) >> temp;
                redTripThresh.push_back(temp);
            } else if (j==(positions.size()-1)) {
                istringstream(string((inputVec[i].begin()+5+positions[j]),
                    (inputVec[i].end()))) >> temp;
                redTripThresh.push_back(temp);
            } else {
                istringstream(string((inputVec[i].begin()+5+positions[j]),
                    (inputVec[i].begin()+4+positions[j+1]))) >> temp;
                redTripThresh.push_back(temp);
            }
        }
    }
    break;
case 315:
    positions.clear();
    positions.push_back(0);
    pos = string((inputVec[i].begin()+4), inputVec[i].end()).find(" ", 0);
    if (pos==string::npos) {
        istringstream(string((inputVec[i].begin()+4),
            inputVec[i].end())) >> temp;
        FTApars.push_back(temp);
    } else {
        while(pos !=string::npos) {
            positions.push_back(pos);
            pos = string((inputVec[i].begin()+4),
                inputVec[i].end()).find(" ", pos+1);
        }
        for (int j=0; j<positions.size(); j++) {
```

```
                            if (j==0) {
                                istringstream(string((inputVec[i].begin()+4+positions[j]),
                                    (inputVec[i].begin()+4+positions[j+1]))) >> temp;
                                FTApars.push_back(temp);
                            } else if (j==(positions.size()-1)) {
                                istringstream(string((inputVec[i].begin()+5+positions[j]),
                                    (inputVec[i].end()))) >> temp;
                                FTApars.push_back(temp);
                            } else {
                                istringstream(string((inputVec[i].begin()+5+positions[j]),
                                    (inputVec[i].begin()+4+positions[j+1]))) >> temp;
                                FTApars.push_back(temp);
                            }
                        }
                    }
                    break;
                case 316:
                    realTimeSimData = string((inputVec[i].begin()+4), inputVec[i].end());
                    break;
                default:
                    cout <<  "Card not read:" << endl;
                    cout << string(inputVec[i].begin(), inputVec[i].begin()+3) << endl;
            }
        }
    }
}
```

A.2. CycleR5.h Source Code

```
//Written By Kevin Makinson
//Oregon State University
//2/24/12
//This the structure that Cycles R5 and controls the parallel structure

#ifndef CycleR5_h
#define CycleR5_h
#include <sstream> //for appending strings
#include <stdlib.h> //for system calls in UNIX
#include <omp.h>
#include <time.h>
#include "OrganizeR5Output.h"
#include "MeanShift.h"

//using namespace std;

void CycleR5(string answer, string ProbType, string ProbOpt, double EndTime, string MinTimeStep,
        double MaxTimeStep, int CtlMode, int MinEdit, int MajEdit, int RstFreq, string R5ExePath,
        string R5RstData, string R5H2oData, const char *InitShOutput, const char *RstShOutput,
        string Rrepos, string libloc, double PCAthreshold, double BW, bool dataOut, string InDir,
        string OutDir, string R5Input, string R5Output, int requestTh, double TStep,
        double T1Step, string FTAdir, string FTAfileName, int numOfCutSets,
        vector <string> stateVarTripNames, vector <string> stateVarCodes,
        vector <string> stateVarEquiv, vector <double> yellowTripThresh,
        vector <double> redTripThresh, vector <double> FTApars, string realTimeSimData) {

        //local defs
        stringstream sstm;
        bool next=false;
        bool counted=false;
        int th_id, nthreads;        //thread identifier & # of threads
        int Windex=0;
        int timestep=1;
        int numOfCycles=0;
        int cycleCounter=1;
```

```
        double p1;          //threshold variables
        double httemp;      //threshold variables
        double voidg;       //threshold variables
        double velgj;       //threshold variables
        double PStep = 2;
        double firstRstNbr=0;
        double t1, t2;
        vector <double> keepGoingEndTime(requestTh, (EndTime+T1Step));
        vector <double> RstNbr;
        vector <int> EndByVec;              //Terminate by Trip, Timestep, or Errors?
        vector <vector <int> > EndBySumVec;  //summary of how the scenarios terminated
        vector <vector<string> > FormatData; //this is the data in string format
        vector <int> translator;            //translates scenario numbers to real scenario names
        vector <string> ThDir;          //name of thread directory
        vector <int> keepGoing;
        vector <int> prevKeepGoing(requestTh, 0);
        vector <string> stateVarCodes2=stateVarCodes; //for when R5 deletes the 7th character
        vector <int> stateVarNum(stateVarTripNames.size());
        vector <double> R5Values (stateVarTripNames.size());
        vector <vector <int> > clustMembers;
        vector <vector <string> > cutSetVec;
        vector <vector <string> > MCdataVec;
        vector <vector <string > > fullSysData;
        vector <vector <string > > realTimeData;
        vector <vector <string> > transientExplanation; //explains transient in words
        transientExplanation.resize(requestTh, vector<string> (0, " "));
        vector<string> singleTransientExplanation;

        string InitShFullPath;
        string RstShFullPath;
        string chmod = "chmod +x ";
        string MkThDirPath;             //make thread directory path
        string MkThODirPath;
        string MkThIDirPath;
        string CsvFile;
        string CsvFilePath;
        string PrevR5RstOutput;
        string R5RstInput;
```

```
        string R5RstOutput;
        string R5RstOutputPath;
        string prpFile;
        string mrpFile;
        string sysDataFileName = (InDir + "/" + realTimeSimData);
        string copyPath;
        string extraTripInfo;

        //deleting 7th character and adding "  " to account for R5's funkiness
        for (unsigned int i=0; i<stateVarCodes2.size(); i++) {
            stateVarCodes2[i].replace(7, 1, "  ");
        }

        //start by determining which transients to run from fault tree
        cout << "Processing fault tree information..." << endl;
        ftaFileFixer(FTAdir+ "/" +FTAfileName + "/" +FTAfileName+".fta");
        doFTA(FTApars, FTAfileName, FTAdir);
        prpFile=(FTAdir+ "/" +FTAfileName + "/" +FTAfileName+".prp");
        mrpFile=(FTAdir+ "/" +FTAfileName + "/" +FTAfileName+".mrp");
        cutSetVec= getCutSetData(prpFile);
        MCdataVec= getMCdata(mrpFile);
        if (numOfCutSets>cutSetVec.size()) {
            cerr << endl <<
                "Input error! Number of requested cutsets greater than size of cutsets"
                << endl << "setting cutsets to maximum value" << endl;
            numOfCutSets=cutSetVec.size();
        }

        vector <int> ThTransientTranslator(requestTh);
        for (int i=0; i<requestTh; i++) {
            ThTransientTranslator[i]=(i%numOfCutSets);
        }

        //---
        //Control structure for RAPS, gigantic while-loop
        while ((answer == "y") || (answer == "yes") || (answer == "Y") || (answer == "Yes")) {
            Windex++;
            sstm << "rst" << Windex << ".p";    //adding index to the string
```

```
        R5RstOutput = sstm.str();
        sstm.str("");                                   //clearing stringstream
        sstm << "rst" << Windex << ".i";    //adding index to the string
        R5RstInput = sstm.str();
        sstm.str("");
        sstm << "rst" << Windex-1 << ".p";  //adding previous index to the string
        PrevR5RstOutput = sstm.str();
        sstm.str("");
        sstm << "rst" << Windex << ".csv";
        CsvFile = sstm.str();
        sstm.str("");
        RstNbr.clear();  //resets RstNbr
        EndByVec.clear(); //resets EndByVec

        //User interface
        if (Windex == 1) {
            RstNbr.push_back(0);
        }

        else if (Windex==2) { //if it's the second time through (first restart)
            firstRstNbr=FindRstNbr(OutDir + "/" + R5Output);
            switch (R5EndBy(OutDir + "/" + R5Output)) {
            case 1:
                EndTime += T1Step;
                cout << "Transient ended by end of alloted time."  << endl;
                break;
            case 2:
                cout << "Transient ended by trip" << endl;
                break;
            case 3:
                cout << "Transient ended by reaching steady state."  << endl;
                break;
            case 0:
                cout << "Transient ended by errors!"  << endl;
            }
        }

        //Run RELAP with the given parameters
```

```
if (Windex == 1) {
    cout << endl <<
        "The RAPSS engine will first perform the initial run for a RELAP5 file."
        << endl;
    cout << "End time: " << fixed << setprecision(2) << EndTime << " s" << endl
    << endl;
    system("read -p \"Press the [Enter] key to continue...\"");
    cout << "initializing RAPSS;" << endl
        << "downloading and installing R libraries from the internet..." << endl;
    t1=omp_get_wtime(); //starts timer
    initR(Rrepos, libloc, PCAthreshold, dataOut, OutDir, stateVarTripNames,
        stateVarCodes, stateVarEquiv, yellowTripThresh, redTripThresh);
    system("R CMD BATCH --slave R_data/initPCA.r R_data/initPCA.Rout");
    cout << "Performing initial RELAP5 run..." << endl;
    InitShFullPath = WriteInitShFile(InitShOutput, InDir, R5ExePath, R5Input,
        R5Output, R5RstData, R5H2oData, OutDir);
    system((chmod + InitShFullPath).c_str());
    system(ChangeFont(4));
    system(InitShFullPath.c_str());
    system(ChangeFont(2));
    cout << "Reorganizing RELAP5 output into csv file..." << endl;
    CsvFilePath = (OutDir + "/" + "initout.csv");
    OrganizeR5Output((OutDir + "/" + R5Output), CsvFilePath, 0);
    cout << endl;
}
else {
    if (numOfCycles<=cycleCounter) {
    cout << "Next run index is " << Windex << ".  Continue run? (y/n)" << endl;
    cin >> answer;
    //if the program doesn't recognize, continue to tell the use to enter a y or n
    while ((answer != "n") && (answer != "y") && (answer != "Y") && (answer != "N")
        && (answer != "yes") && (answer != "no") && (answer != "Yes") &&
            (answer != "No")) {
        cout << "You did not enter a \"y\" or an \"n\"!" << endl;
        cout << "Next run index is " << Windex << " Continue run? (y/n)" << endl;
        cin >> answer;
    }
    if ((answer == "n") || (answer == "no") || (answer == "N") || (answer == "No")) {
```

```
    break;
}
cout << "How many cycles would you like to run? " << endl;
cin >> numOfCycles;
if (numOfCycles<1) {break;}
cycleCounter=0;
system("read -p \"Press the [Enter] key to continue...\"");
}
cycleCounter++;
cout << endl << "The RAPS engine will now perform the restart run" << endl;
cout << "End time for one cycle: " << fixed << setprecision(2)
    << EndTime << " s" << endl;
cout << "Input file: " << R5RstInput << endl;
cout << "Output file: " << R5RstOutput << endl;
vector <string> ThDir; //resets the vector

if (Windex==2) {
    fullSysData=loadSystemData(sysDataFileName);
    timestep=1; //writes a single timestep to "realTimeData.txt"
    realTimeData=realTimeSimulator(fullSysData, timestep, OutDir);
} else {
    realTimeData=realTimeSimulator(fullSysData, timestep, OutDir);
}


t2=omp_get_wtime(); //grabs time
timestep = int(t2-t1);
cout << "Time sampled: " << timestep << " s" << endl << endl;
//start parallel processing for restart runs.
if (requestTh>omp_get_max_threads()) { //need ()?
    requestTh=omp_get_max_threads();
    omp_set_num_threads(requestTh);
    cerr <<
        "Number of threads greater than maximum allowable by the system."
        << endl <<  "Setting number of threads to " <<
            omp_get_max_threads() << endl;
    system("read -p \"Press the [Enter] key to continue...\"");
}
```

```
else if (requestTh>1) {
    omp_set_dynamic(0); // turn off dynamic teams
    omp_set_num_threads(requestTh);
}
else {
    cerr << "Invalid thread number request.  Setting number of threads to 2."
        <<endl;
    omp_set_num_threads(2);
    system("read -p \"Press the [Enter] key to continue...\"");
}
cout << "Spawning threads, sampling MASLWR data, writing restart files..."
    << endl;
cout << "Prepare for RELAP..." << endl;
//Begin parallel processing section
system(ChangeFont(4));
#pragma omp parallel private(th_id, RstShFullPath, singleTransientExplanation)
    shared(nthreads, transientExplanation) //only on this line for print version
{
    th_id = omp_get_thread_num();
    #pragma omp critical  //restricts the execution of the associated statement
    {
        srand(time(NULL));
        ThDir.push_back(NameDir(th_id)); //puts in directory names
        if (Windex== 2) {
            MkThDirPath = ("mkdir " + OutDir + "/" + NameDir(th_id));
            MkThODirPath = ("mkdir " + OutDir + "/" + NameDir(th_id) +
                "/outputs");
            MkThIDirPath = ("mkdir " + OutDir + "/" + NameDir(th_id) +
                "/inputs");
            system(MkThDirPath.c_str());
            system(MkThODirPath.c_str());
            system(MkThIDirPath.c_str());
            singleTransientExplanation=RstIptGen(R5Output, R5RstInput,
                NameDir(th_id), ProbType, ProbOpt,
                firstRstNbr, EndTime, MinTimeStep, MaxTimeStep, CtlMode, MinEdit,
                MajEdit, RstFreq, th_id, OutDir, prevKeepGoing,
                MCdataVec[ThTransientTranslator[th_id]], stateVarTripNames,
                stateVarCodes, stateVarEquiv, yellowTripThresh, redTripThresh,
```

```
            requestTh, realTimeData);
    }  else if (prevKeepGoing[th_id]==1) { //continue's from spot left off
        keepGoingEndTime[th_id]+=TStep;
        singleTransientExplanation=RstIptGen(R5Output, R5RstInput,
            NameDir(th_id), ProbType, ProbOpt,
            RstNbr[th_id], keepGoingEndTime[th_id], MinTimeStep, MaxTimeStep,
            CtlMode, MinEdit, MajEdit, RstFreq, th_id, OutDir,
            prevKeepGoing, MCdataVec[ThTransientTranslator[th_id]],
            stateVarTripNames, stateVarCodes, stateVarEquiv,
            yellowTripThresh, redTripThresh, requestTh, realTimeData);
    }  else {
        singleTransientExplanation=RstIptGen(R5Output, R5RstInput,
            NameDir(th_id), ProbType, ProbOpt,
            firstRstNbr, EndTime, MinTimeStep, MaxTimeStep, CtlMode, MinEdit,
            MajEdit, RstFreq, th_id, OutDir, prevKeepGoing,
            MCdataVec[ThTransientTranslator[th_id]], stateVarTripNames,
            stateVarCodes, stateVarEquiv, yellowTripThresh, redTripThresh,
            requestTh, realTimeData);
    }
    RstShFullPath = WriteRstShFile(RstShOutput, InDir, R5ExePath,
        NameDir(th_id), R5RstData, R5RstInput, R5RstOutput,
        R5H2oData, Windex, OutDir);
}

#pragma omp barrier //wait to run until all the restart files are written
    system((chmod + RstShFullPath).c_str());
    system(RstShFullPath.c_str());//this actually runs it
#pragma omp single
{
    nthreads = omp_get_num_threads();
}

#pragma omp critical //maybe atomic?
{
    transientExplanation[th_id]=singleTransientExplanation;

}

```

```
} // end of parallel section!

system(ChangeFont(2));
cout << endl << "RELAP run on " << nthreads << " simultaneous threads" << endl;
EndByVec.clear(); //resets EndByVec
RstNbr.clear();  //resets RstNbr
for (int i=0; i<nthreads; i++) {
    EndByVec.push_back(R5EndBy(OutDir + "/" + NameDir(i) +
        "/outputs/" +R5RstOutput));
    RstNbr.push_back(FindRstNbr(OutDir + "/" + NameDir(i) +
        "/outputs/" +R5RstOutput));
}
EndBySumVec=EndBySummary(EndByVec, nthreads);
for (unsigned int j=0; j<EndBySumVec.size()-1; j++) {
    if (!EndBySumVec[j].empty()) {
        cout << "Scenarios: " << endl;
        for (unsigned int i=0; i<EndBySumVec[j].size(); i++) {
            if  (EndBySumVec[j][i]==1) {
                cout << i << ", ";
                }
            }
        cout << endl;
        cout << "ended by ";
        switch (j) {
        case 0:
            cout << "errors!"  << endl;
            break;
        case 1:
            cout << "end of allotted time."  << endl;
            break;
        case 2:
            cout << "trip " << endl;
        //  break;
        //case 3:
        //  cout << "reaching steady state."  << endl;
        }
        cout << endl;
    }
```

```
}

//structure of updating keepGoing[]
for (int i=0; i<nthreads; i++) {
    next=false;
    CsvFilePath = (OutDir + "/" + NameDir(i) + "/outputs/" + CsvFile);
    R5RstOutputPath = (OutDir + "/" + NameDir(i) + "/outputs/" + R5RstOutput);
    if (EndBySumVec[0][i]==0) { //if it did not end by errors, then organize.
        FormatData=OrganizeR5Output(R5RstOutputPath, CsvFilePath, i);
    }
    int counter=0;
    if (EndBySumVec[0][i]==0 && counted==false) {
        for (int k=0; k<FormatData[0].size(); k++) {
            if ((FormatData[0][k]==(stateVarTripNames[counter])) &&
                (FormatData[1][k]==(stateVarCodes[counter]) ||
                FormatData[1][k]==(stateVarCodes2[counter]))) {
                stateVarNum[counter] = k;
                if (counter==(stateVarCodes.size()-1)) {
                    counted=true;
                    break;
                }
                else {counter++;} //counter is the position of the state var
            }
        }
    }
    //checks thresholds and pushes "keep going" threads on to a vector
    for (unsigned int j=4; j<(FormatData.size()); j++) {
        if (next==true) {break;}
        for (int n=0; n<stateVarCodes.size(); n++) {
            //coercing strings to doubles
            double temp;
            istringstream(FormatData[j][stateVarNum[n]]) >> temp;
            R5Values[n]=temp;
            if ((stateVarEquiv[n]=="lt") && (R5Values[n]<yellowTripThresh[n]) &&
                (FormatData[j][stateVarNum[n]]!="")) {
                keepGoing.push_back(1);
                next=true;
                break;
```

```
            }
            if ((stateVarEquiv[n]=="gt") && (R5Values[n]>yellowTripThresh[n]) &&
                (FormatData[j][stateVarNum[n]]!="") ) {
                keepGoing.push_back(1);
                next=true;
                break;
            }
            if (j==(FormatData.size()-1)) {
                keepGoing.push_back(0); //if nothing needs to be flagged, push 0
                break;
            }
        }
    }
}
for (unsigned int i=0; i<EndBySumVec[2].size(); i++) {
    if (EndBySumVec[2][i] == 1) { //if it's been flagged to keep going, & tripped
        keepGoing[i]=0;     //remove the keep going flag
        keepGoingEndTime[i]=EndTime; //resets keepGoingEndTime for flagged
    }
}
if (Windex>=3) { // if something is flagged, don't cluster it
    for (unsigned int i=0; i<prevKeepGoing.size(); i++) {
        if (prevKeepGoing[i]==1) {
            EndBySumVec[1][i]=0;  //don't do clustering on this element
        }
    }
}

int EndByTimeStepCounter=0;
int EndByTripCounter=0;
for (unsigned int i=0; i<EndBySumVec[1].size(); i++) {
    if (EndBySumVec[1][i]==1) {EndByTimeStepCounter++;}
    }
for (unsigned int i=0; i<EndBySumVec[2].size(); i++) {
    if (EndBySumVec[2][i]==1) {EndByTripCounter++;}
    }

translator=updateRwindex(Windex, nthreads, EndBySumVec, EndByTripCounter,
```

```
    EndByTimeStepCounter, prevKeepGoing, MCdataVec, ThTransientTranslator,
    timestep);

//goes between MSA indexes and thread indexes
system("R CMD BATCH R_data/updateRwindex.r R_data/updateRwindex.Rout");

if (EndByTripCounter==0) {
    cerr <<"No scenarios ended by trip, skipping plotting alerts ..." << endl;
} else {
    cout << "Plotting tripped data..." << endl;
    system("R CMD BATCH display.r R_data/display.Rout");
}

//adding extra info about the trip from a txt file written by R
//-------
for (int i=0; i<nthreads; i++) {
    if (EndBySumVec[2][i]==1) {
        sstm << OutDir << "/tripRst" << Windex << "_Sc" << i << ".txt";
        extraTripInfo = sstm.str();
        sstm.str("");
        transientExplanation[i].push_back(LoadFile(extraTripInfo)[0]);
        system(("rm " + extraTripInfo).c_str()); //removing temporary file
    }
}

if (EndByTimeStepCounter<2) {
    cerr << "Less than two scenarios completed time histories without flags."
        << endl << "Skipping scenario clustering..." << endl;
} else {
    cout << "Extracting and organizing data; performing PCA..." << endl;
    system("R CMD BATCH PCA.R R_data/PCA.Rout");
    cout << "Performing MSA..." << endl;
    clustMembers=MeanShift(Windex, BW, OutDir, EndBySumVec[1], translator);
    cout << "Rearranging, outputting and plotting data..." << endl;
    system("R CMD BATCH unMSAPCA.R R_data/unMSAPCA.Rout");
    //output
    htmlDisplayWriter(OutDir, InDir, Windex, EndBySumVec, keepGoing,
        clustMembers, transientExplanation);
```

```
                }
                //clear keepGoing[]
                prevKeepGoing.clear();
                prevKeepGoing=keepGoing;
                keepGoing.clear();
                }
            }
        }

#endif
```

A.3. BloodAndGuts.h Source Code

```
// Created by Kevin Makinson
// 2/22/12
// This files contains the misc functions for RAPSS-STA

#ifndef BloodAndGuts_h
#define BloodAndGuts_h
#include <time.h>
#include <vector>
#include <sstream> //for appending strings
#include <iomanip> // for showpoint
#include <stdlib.h> //for system calls in UNIX
#include <fstream>
#include <algorithm>
#include <sstream>
using namespace std;
string author = "Kevin Makinson";

//Serch function returns a vector with the line numbers of where the key is
vector<int> SearchVec(vector<string> &text, string key) {
   //returns a vector of the line numbers of the search term.
   vector<int> LineNums;
   size_t found;
   bool FoundOne = false;
   int size = text.size();
   for (int i=0; i<size; i++) {
       found=text[i].find(key);
       if (found!=string::npos) {
           LineNums.push_back(i);
           FoundOne = true;
       }
       else if (FoundOne == false && i==(size-1)) {
           //Since there is no line "0" this will signify an error
           LineNums.push_back(0);
       }
   }
```

```
   return LineNums;
}

vector<string> LoadFile(string FullFilePath) {
   string line;
   int size = 0;
   ifstream fin(FullFilePath.c_str());
   //counting lines
   while (getline(fin, line)) {
       size++;
   }
   vector<string> text(size, "n/a");
   //This resets fin to the begining
   fin.clear();
   fin.seekg(0);
   //loading the file into a vector of strings: "text"
   for (int i=0; i<size; i++) {
       getline(fin, text[i]);
   }
   fin.close();
   return text;
}

//This function searches for the restart number in an R5 output file
//and returns the restart number
double FindRstNbr(string R5OutputFile) {
   //local Declartions
   string key = "0---Restart no.";
   double RstNbr;
   string RstNbrString;
   int KeyStringSize=15;
   vector<int> LineNums;
   //Loading the file into a vector called "text"
   vector<string> text = LoadFile(R5OutputFile);
   //Search for the line number of the key
   LineNums = SearchVec(text, key);
   //Start at the end of the key string character on the line
   for (int i=KeyStringSize; i<(KeyStringSize+10); i++) {
```

```
       RstNbrString += text[LineNums.back()][i];
           if ((text[LineNums.back()][i])==('w'))
               break;
   }
   //change the string to a double
   istringstream(RstNbrString) >> RstNbr;
   return (RstNbr);
}

//This tells the outside world how the R5 terminated
//1=TimeStep, 2=Trip, 3=Steady State, 0=Error
int R5EndBy(string FileName) {
   vector<string> text = LoadFile(FileName);
   string TimeStep = "0Transient terminated by end of time step cards.";
   string TimeStep3D = " Transient terminated by end of time step cards.";
   string Trip = "0Transient terminated by trip.";
   string Trip3D = " Transient terminated by trip.";
   string sState = "0Transient has reached steady state.";
   string fail = "0******** Transient terminated by failure.";
   int EndBy;
   if (text.back()==fail) {  //can't use a switch statement for strings
       EndBy=0;
   } else if((text.back()==TimeStep) || (text.back()==TimeStep3D) ) {
       EndBy=1;
   } else if ((text.back()==Trip) || (text.back()==Trip3D)) {
       EndBy=2;
   } else if (text.back()==sState) {
       EndBy=3;
   }
   return EndBy;
}

//This function expects a vector EndByVec, and returns a 2D vector with a summary of the
//threads that ended a certain way, 1: time, 2:Trip, 3: Steady State, 0: Errors
//9/4/12 changed to just output 0s and 1s instead of th numbers
vector <vector <int> > EndBySummary(vector<int> EndByVec, int nthreads) {
   vector <int> temp0;
   vector <int> temp1;
```

```
    vector <int> temp2;
    vector <int> temp3;
    vector <vector <int> > EndBySumVec;
    for (int i=0; i<nthreads; i++) {
        switch (EndByVec[i]) {
            case 0:
                temp0.push_back(1);
                temp1.push_back(0);
                temp2.push_back(0);
                temp3.push_back(0);
                break;
            case 1:
                temp0.push_back(0);
                temp1.push_back(1);
                temp2.push_back(0);
                temp3.push_back(0);
                break;
            case 2:
                temp0.push_back(0);
                temp1.push_back(0);
                temp2.push_back(1);
                temp3.push_back(0);
                break;
            case 3:
                temp0.push_back(0);
                temp1.push_back(0);
                temp2.push_back(0);
                temp3.push_back(1);
        }
    }
    EndBySumVec.push_back(temp0);
    EndBySumVec.push_back(temp1);
    EndBySumVec.push_back(temp2);
    EndBySumVec.push_back(temp3);
    return EndBySumVec;
    }

//This guy converts between how the rest of the world does scientific notation
```

```
//and how R5 does it.
string R5SciConv(double num) {
   //convert double to string
   stringstream sstm;
   sstm << scientific << setprecision(2) << num;
   string StringNum = sstm.str();
   sstm.str("");
   //removing the "e"
   return StringNum.erase(4,1);
}

string WriteInitShFile (const char *ShOutput, string InDir, string R5ExePath, string R5Input,
   string R5Output, string R5RstData, string R5H2oData, string OutDir) {
   string FullFilePath = (OutDir + "/" + ShOutput);
   ofstream fout (FullFilePath.c_str());
   fout << "cd " << R5ExePath << endl;
   fout << "relap5.x " << "-i " << InDir << "/" << R5Input << " -o " << OutDir << "/"
       << R5Output << " -r " <<  OutDir << "/" << R5RstData << endl;
   return (FullFilePath);
}

string WriteRstShFile (const char *RstShOutput, string InDir, string R5ExePath, string ThDir,
   string R5RstData, string R5RstInput, string R5RstOutput, string R5H2oData,
   int Windex, string OutDir) {
   string FullFilePath = (OutDir + "/" + ThDir + "/" + RstShOutput);
   ofstream fout (FullFilePath.c_str());
   if (Windex==2) {
       fout << "cp " << OutDir << "/" << R5RstData << " " << OutDir << "/" << ThDir << endl;
       fout << "cp " << InDir << "/Alert.gif " << OutDir << endl; //alerts
       fout << "cp " << InDir << "/tswtabs.css " << OutDir << endl; //buttons
   }
   fout << "cd " << R5ExePath << endl;
   fout << "relap5.x " << "-i " << OutDir << "/" << ThDir << "/inputs/" << R5RstInput
       << " -o " << OutDir << "/" << ThDir << "/outputs/" << R5RstOutput << " -r "
       << OutDir << "/" << ThDir << "/" << R5RstData << endl;
   return (FullFilePath);
}
```

```
//this initializes R with the correct libraries and initial conditions for PCA
void initR(string Rrepos, string libloc, double PCAthreshold, bool dataOut, string OutDir,
   vector <string> stateVarTripNames, vector <string> stateVarCodes,
   vector <string> stateVarEquiv, vector <double> yellowTripThresh,
   vector <double> redTripThresh) {
   string MkLibDir = ("mkdir -p " + libloc);
   string MkRdataDir = ("mkdir -p R_data");
   string RinpPath= ("R_data/initPCA.r");
   system(MkLibDir.c_str());
   system(MkRdataDir.c_str());
   ofstream fout (RinpPath.c_str());
   ifstream fin((libloc+ "/abind").c_str());
   //comments section of input file
   fout << "#!/usr/bin/Rscript" << endl;
   fout << "#" << string(3, ' ') << __DATE__ << endl;
   fout << "#" << string(3, ' ') << "Written by " << author << endl;
   fout << "#" << string(3, ' ')
       <<"This file loads the libraries and initial parameters in R"<< endl;
   fout << "#\n#\n#" << string(70, '-') << endl; //end comments
   fout << "rm(list=ls())" << endl << endl;
   fout << "Rrepos<-\"" << Rrepos << "\"" << endl << "libloc<-\"" << libloc << "\"" << endl;
   fout << "threshold<-" << PCAthreshold << endl;
   fout << "IODir<-\"" << OutDir <<"\"" << endl << "libloc<-\"" << libloc <<"\"" << endl;
   fout << "dataOut<-" << dataOut << endl;
   if(!fin.good()) { //don't install if already installed
       fout << "install.packages(\"corpcor\", repos=Rrepos, lib=libloc)" << endl;
       fout << "install.packages(\"abind\", repos=Rrepos, lib=libloc)" << endl;
       fout << "install.packages(\"MASS\", repos=Rrepos, lib=libloc)" << endl;
       fin.close();
   }

   //thresholds
   fout << "thresholds<-rbind(" << yellowTripThresh[0] << ","
       << redTripThresh[0] << ")" << endl;
   for (unsigned int i=1; i<yellowTripThresh.size(); i++) {
       fout << "thresholds<-cbind(thresholds, rbind(" << yellowTripThresh[i] << ","
           << redTripThresh[i] << "))" << endl;
   }
```

```

    fout << "stateVarTripNames <- c(\"";
    for (unsigned int i=0; i<stateVarTripNames.size(); i++) {
        if (i==0) {
            fout << stateVarTripNames[i];
        } else {
            fout << "\", \"" << stateVarTripNames[i];
        }
    }
    fout << "\")" << endl;

    fout << "equivalence <- c(\"";
    for (unsigned int i=0; i<stateVarEquiv.size(); i++) {
        if (i==0) {
            fout << stateVarEquiv[i];
        } else {
            fout << "\", \"" << stateVarEquiv[i];
        }
    }
    fout << "\")" << endl;

    fout << "stateVarCodes <- c(\"";
    for (unsigned int i=0; i<stateVarCodes.size(); i++) {
        if (i==0) {
            fout << stateVarCodes[i];
        } else {
            fout << "\", \"" << stateVarCodes[i];
        }
    }
    fout << "\")" << endl;
    //--end of thesholds section
    fout << "save.image(\"R_data/RAPSspace.RData\")" << endl;
    fin.close();
    fout.close();
}

//this updates R with each cycle
vector <int> updateRwindex(int Windex, int nthreads, vector <vector <int> > EndBySumVec,
```

```
int EndByTripCounter, int EndByTimeStepCounter, vector <int> prevKeepGoing,
vector < vector <string> > cutSetVec, vector <int> ThTransientTranslator, int timestep) {
vector <int> translator;
int counter=0;
string file= ("R_data/updateRwindex.r");
ofstream fout (file.c_str());
fout << "load(\"R_data/RAPSspace.RData\")" << endl;
fout << "rstNum<-" << Windex << endl;
fout << "thNum<-" << nthreads << endl;
    fout << "IncludeTh<- c(";
    for (unsigned int i=0; i<EndBySumVec[1].size(); i++) {
        if (EndBySumVec[1][i]==1) {
            if (counter==0) {
                fout << i;
                translator.push_back(i);
            } else {
                fout << ", " << i;
                translator.push_back(i);
            }
            counter++;
            }
        }
    fout << ")" << endl;
counter=0; //reset counter for next loop
    fout << "EndByTrip<- c(";
    for (unsigned int i=0; i<EndBySumVec[2].size(); i++) {
        if (EndBySumVec[2][i]==1) {
            if (counter==0) {
                fout << i;
            } else {
                fout << ", " << i;
            }
            counter++;
            }
        }
    fout << ")" << endl;

counter=0; //reset counter for next loop
```

```
       fout << "prevKeepGoing<- c(";
       for (unsigned int i=0; i<prevKeepGoing.size(); i++) {
           if (prevKeepGoing[i]==1) {
               if (counter==0) {
                   fout << i;
               } else {
                   fout << ", " << i;
               }
               counter++;
               }
           }
       fout << ")" << endl;
       fout << "cutSetProbs<- c(\"";
       for (unsigned int i=0; i<cutSetVec.size(); i++) {
               if (i==0) {
                   cutSetVec[i].pop_back();
                   fout << cutSetVec[i].back();
               } else {
                   cutSetVec[i].pop_back();
                   fout << "\",\" " << cutSetVec[i].back();
               }
           }
       fout << "\")" << endl;
       fout << "ThTransientTranslator<- c(";
       for (unsigned int i=0; i<ThTransientTranslator.size(); i++) {
               if (i==0) {
                   fout << ThTransientTranslator[i];
               } else {
                   fout << ", " << ThTransientTranslator[i];
               }
           }
       fout << ")" << endl;
       fout << "timestep <- " << timestep << endl;
       fout << "save.image(\"R_data/RAPSspace.RData\")" << endl;
   return translator;
}

const char *ChangeFont(int ColorCode) {
```

```
   string FullFilePath = "ChangeFont.sh";
   ofstream fout (FullFilePath.c_str());
   fout << "tput setf " << ColorCode << endl << "tput bold" << endl << "exit 0";
   string chmod = ("chmod +x " + FullFilePath);
   system(chmod.c_str());                          //creating executable
   return(FullFilePath.c_str());
}

const char *ResetFont() {
   string FullFilePath = "ResetFont.sh";
   ofstream fout (FullFilePath.c_str());
   fout << "tput sgr0" << endl << "exit 0" << endl;
   string chmod = ("chmod +x " + FullFilePath);
   system(chmod.c_str());                          //creating executable
   return(FullFilePath.c_str());
}

//makes a directory based on the thread ID, and returns a string of directory name
string NameDir(int th_id) {
   stringstream sstm;
   sstm << "Th_"  << th_id << "_data";
   string ThDir = sstm.str();
   sstm.str("");
   return (ThDir);
}


//Trims white space around words grabbed from R5
string TrimSpace(string MyString) {
   string whitespaces (" \t\f\v\n\r");
   size_t endpos = MyString.find_last_not_of(whitespaces);
   size_t startpos = MyString.find_first_not_of(whitespaces);
   if(string::npos != endpos)
       MyString = MyString.substr(0, endpos+1);
   else
       MyString.clear();            // if string is all whitespace
   if(string::npos != startpos)
       MyString = MyString.substr(startpos);
```

```
    else
        MyString.clear();            // if string is all whitespace
    return MyString;
    }

//gets probability info from prp file
vector < vector <string > > getCutSetData(string prpFile) {
    vector <string> textP = LoadFile(prpFile);
    string key = "Minimal cut set probabilities :";
    string prob;
    string word;
    vector <int> keyLocVec = SearchVec(textP, key);
    int keyLoc = keyLocVec[0];
    int linePlace=0;
    bool multiLine=false;
    bool lastLineInSet=false;
    int i=6;
    int keyLocAdder=2;
    int count; //counts how many lines the multline algorithm uses
    vector <vector <string> > cutSetVec;
    vector <string> eventVec; //a single event of a cutset
    keyLocAdder=2;

    while (!textP[keyLoc+keyLocAdder].empty()) {
        count = 0;
        if (!textP[keyLoc+keyLocAdder+1].empty()) {
            if ((textP[keyLoc+keyLocAdder][i-4] != ' ') &&
                (textP[keyLoc+keyLocAdder+1][i-4] == ' ')
                && multiLine==false) {
                multiLine=true;
            }
        }
        if (multiLine==false) { // this statement is only for single lines
            while ((textP[keyLoc+keyLocAdder][i] !=' ') ||
                (textP[keyLoc+keyLocAdder][i+1] != ' ')) {
                word.clear();
                i++;
                while (textP[keyLoc+keyLocAdder][i] != ' ') {
```

```
                    word += textP[keyLoc+keyLocAdder][i];
                    i++;
                }
                eventVec.push_back(word);
                word.clear();
            }
            for (int k=39; k<52; k++) {
                prob += (textP[keyLoc+keyLocAdder][k]);
            }
            eventVec.push_back(prob);
            prob.clear();
        } else {  //for multiline cutsets
            i=6; //resets to the beginning of the cutsets line
            lastLineInSet=false;
            while (lastLineInSet == false) {// this goes to the next cut set number
                if (textP[keyLoc+keyLocAdder+1][2] != ' ') {
                    lastLineInSet=true;
                    for (int k=39; k<52; k++) {
                        prob += (textP[keyLoc+keyLocAdder-count][k]);
                    }
                }
                while ((textP[keyLoc+keyLocAdder][i] !=' ') ||
                    (textP[keyLoc+keyLocAdder][i+1] != ' ')) {
                    // this goes to the end of the line
                    string word;
                    i++;
                    while (textP[keyLoc+keyLocAdder][i] != ' ') { //goes through individual words
                        word += textP[keyLoc+keyLocAdder][i];
                        i++;
                    }
                    eventVec.push_back(word);
                }
                if (lastLineInSet == false) {
                    keyLocAdder++;
                }
                i=6; //resets to the beginning of the cutsets line
                count++;
            }
```

```
       }
       eventVec.push_back(prob);
       cutSetVec.push_back(eventVec);
       eventVec.clear(); //clears event vec
       prob.clear();
       keyLocAdder++; //procedes to next line
       i=6; //resets to the beginning of the cutsets line
   }
   return cutSetVec;
}

//gets Monte Carlo data from mrp file
vector < vector <string > > getMCdata(string mrpFile) {
   vector < vector <string > > MCvec;
   vector <string> textM = LoadFile(mrpFile);
   //below this is pasted data from above
   string key = "Compressed:";
   string prob;
   string word;
   vector <int> keyLocVec = SearchVec(textM, key);
   int keyLoc = keyLocVec[0];
   int linePlace=0;
   bool multiLine=false;
   bool lastLineInSet=false;
   int i=6;
   int keyLocAdder;
   int count; //counts how many lines the multline algorithm uses
   vector <string> eventVec; //a single event of a cutset
   keyLocAdder=4;

   while (!textM[keyLoc+keyLocAdder].empty()) {
       count = 0;
       if (!textM[keyLoc+keyLocAdder+1].empty()) {
           if ((textM[keyLoc+keyLocAdder][i-4] != ' ') &&
               (textM[keyLoc+keyLocAdder+1][i-4] == ' ')
               && multiLine==false) {
               multiLine=true;
           }
```

```
        }
        if (multiLine==false) { // this statement is only for single lines
            while ((textM[keyLoc+keyLocAdder][i] !=' ') ||
                (textM[keyLoc+keyLocAdder][i+1] != ' ')) {
                word.clear();
                i++;
                while (textM[keyLoc+keyLocAdder][i] != ' ') {
                    word += textM[keyLoc+keyLocAdder][i];
                    i++;
                }
                eventVec.push_back(word);
                word.clear();
            }
            for (int k=38; k<73; k++) {
                prob += (textM[keyLoc+keyLocAdder][k]);
            }
            eventVec.push_back(prob);
            prob.clear();
        }
        else {  //for multiline cutsets
            i=6; //resets to the beginning of the cutsets line
            lastLineInSet=false;
            while (lastLineInSet == false) {// this goes to the next cut set number
                if (textM[keyLoc+keyLocAdder+1][2] != ' ') {
                    lastLineInSet=true;
                    for (int k=38; k<73; k++) {
                        prob += (textM[keyLoc+keyLocAdder-count][k]);
                    }
                }
                while ((textM[keyLoc+keyLocAdder][i] !=' ') ||
                    (textM[keyLoc+keyLocAdder][i+1] != ' ')) {
                    // this goes to the end of the line
                    string word;
                    i++;
                    while (textM[keyLoc+keyLocAdder][i] != ' ') { //goes through individual words
                        word += textM[keyLoc+keyLocAdder][i];
                        i++;
                    }
```

```
                    eventVec.push_back(word);
                }
                if (lastLineInSet == false) {
                    keyLocAdder++;
                }
                i=6; //resets to the beginning of the cutsets line
                count++;
            }
        }
        eventVec.push_back(prob);
        MCvec.push_back(eventVec);
        eventVec.clear(); //clears event vec
        prob.clear();
        keyLocAdder++; //procedes to next line
        i=6; //resets to the beginning of the cutsets line
    }
    reverse(MCvec.begin(), MCvec.end()); //puts high probability events on top.
    return MCvec;
}

void doFTA(vector <double> FTApars, string FTAfileName, string FTAdir) {
    string FTAinput = "FTA/fta_input_file";
    ofstream fout1 (FTAinput.c_str());
    fout1 << FTAdir << "/"  << FTAfileName << "/" << FTAfileName << ".fta," << FTApars[0] <<
        "," <<  FTApars[1] << "," << FTApars[2] << "," << FTApars[3];
    fout1.close();

    string cdFilePath = "runFTA.sh";
    ofstream fout (cdFilePath.c_str());
    // if it's already done, don't do it
    ifstream ifile((FTAdir +"/" + FTAfileName + "/" + FTAfileName + ".prp").c_str());
    if (!ifile) {
        fout << "dos2unix " << FTAdir << "/" << FTAfileName << "/"
            << FTAfileName <<".fta" << endl;
        fout << "dos2unix " << FTAdir << "/" << FTAfileName << "/"
            << FTAfileName <<".ped" << endl;
    }
    ifile.close();
```

```
   fout << "cd " << FTAdir << endl;
   fout << "run.sh fta_input_file" << endl;
   fout << "exit 0" << endl;
   fout.close();
   string chmoder = ("chmod +x runFTA.sh");
   system(chmoder.c_str());
   system(cdFilePath.c_str());
}

void ftaFileFixer(string filename) { //this deletes the stuff that LiteFTA doesn't like
   vector <string> text;
   string temp;
   text = LoadFile(filename);
   int i=text[0].size()-1;
   while (text[0][i] != ('\\')) {
       i--;
       if(i==0) {break;}
   }
   if (i!=0) {
       for (int j=(i+1); j<text[0].size(); j++) {
           temp+=text[0][j];
       }
       text[0]=temp;
       ofstream fout(filename.c_str());
       for (int i=0; i<text.size(); i++) {
           fout << text[i] << endl;
       }
       fout.close();
   }
}

vector <vector <string > >  loadSystemData(string sysDataFileName) {
   string word;
   vector <string> row;
   vector <vector <string > > sysData;
   vector<string> text;
   text=LoadFile(sysDataFileName);
```

```
    for (int j=0; j<text.size(); j++) {
        for (int i=0; i<text[j].size(); i++) {
            word+=text[j][i];
            if ( (text[j][i]== ('\t')) || (i==(text[j].size()-1)) ) {
                row.push_back(word.substr(0, word.size()-1));
                word.clear();
            }
        }
        sysData.push_back(row);
        row.clear();
    }
    return sysData;
}

double qualConverter(double LDP301, int vol) {
    double water;
    if (vol==1 && LDP301>0.6681) {
        water=1.0;
    } else {
        water=7.0E-3;
    }
    if (vol==2 && LDP301>0.5800) {
        water=1.0;
    } else {
        water=7.0E-3;
    }
    if (vol==3 && LDP301>0.5240) {
        water=1.0;
    } else {
        water=7.0E-3;
    }
    if (vol==4 && LDP301>0.4072) {
        water=1.0;
    } else {
        water=7.0E-3;
    }
    if (vol==5 && LDP301>0.2904) {
        water=1.0;
```

```
    } else {
        water=7.0E-3;
    }
    if (vol==6 && LDP301>0.1736) {
        water=1.0;
    } else {
        water=7.0E-3;
    }
    if (vol==7 && LDP301>0.0868) {
        water=1.0;
    } else {
        water=1.0;
    }
    return water;
}

double Linterpolate(double A1, double A2, double B1, double B2, double B3) {
    double A3;
    A3=((A1-A2)*((B3-B2)/(B1-B2))+A2);
    return A3;
}

vector <vector <string > > realTimeSimulator(vector<vector<string > > sysData,
    int timestep, string OutDir) {
    vector <vector <string > > realTimeData;
    ofstream fout ((OutDir + "/realTimeData.txt").c_str());
    for (int i=0; i<sysData[0].size(); i++) {
        fout << sysData[0][i] << "\t";
    }
    fout << endl;
    for (int i=0; i<sysData[0].size(); i++) {
        fout << sysData[timestep][i] << "\t";
    }
    fout.close();
    realTimeData=loadSystemData((OutDir + "/realTimeData.txt").c_str());
    return realTimeData;
}
```

```
//Generates Restart Input files
vector <string> RstIptGen(string R5Output, string R5RstInput, string ThDir,
   string ProbType, string ProbOpt, double RstNbr, double EndTime, string MinTimeStep,
   double MaxTimeStep, int CtlMode, int MinEdit, int MajEdit, int RstFreq, int th_id,
   string OutDir, vector <int> prevKeepGoing, vector <string> transient,
   vector <string> stateVarTripNames, vector <string> stateVarCodes,
   vector <string> stateVarEquiv, vector <double> yellowTripThresh,
   vector <double> redTripThresh, int requestTh, vector<vector<string > > sysData) {

   double U;
   double InitPres;
   vector <int> Vbreak;
   int numOfValves;
   srand(time(NULL));
   U=(double)(rand())/(RAND_MAX);      //uniform distribution 0.5
   string FullFilePath= (OutDir + "/" + ThDir + "/inputs/" + R5RstInput);
   ofstream fout (FullFilePath.c_str());
   int varNum=16;
   int varCount=0;
   int loopCount=0;
   double TFavg1, TFavg2;
   double PT301, PT511, PT602, LDP301, FVM602M, FVM602T, TF111, TF121, TF122,
       TF123, TF124, TF131, TF132, TF133, TF134, TF501;
   vector <double> TFlin1;
   vector <vector <string> > transientExplanation;
   transientExplanation.resize(requestTh, vector<string> (0, " "));
   vector<string> singleTransientExplanation;

   while (varCount<varNum) {
       if(sysData[0][loopCount]==("\"PT301_PressurizerPressure\"")) {
           istringstream(sysData[1][loopCount]) >> PT301;
           varCount++;
       } else if (sysData[0][loopCount]==("\"PT511_SGInletPressure_Bundle_1\"")) {
           istringstream(sysData[1][loopCount]) >> PT511;
           varCount++;
       } else if (sysData[0][loopCount]==("\"LDP301_Uncompensated_Level\"")) {
           istringstream(sysData[1][loopCount]) >> LDP301;
           varCount++;
```

```
        } else if (sysData[0][loopCount]==("\"FVM602M_Steam_MassFlow\"")) {
            istringstream(sysData[1][loopCount]) >> FVM602M;
            varCount++;
        } else if (sysData[0][loopCount]==("\"TF111\"")) {
            istringstream(sysData[1][loopCount]) >> TF111;
            varCount++;
        } else if (sysData[0][loopCount]==("\"TF121\"")) {
            istringstream(sysData[1][loopCount]) >> TF121;
            varCount++;
        } else if (sysData[0][loopCount]==("\"TF122\"")) {
            istringstream(sysData[1][loopCount]) >> TF122;
            varCount++;
        } else if (sysData[0][loopCount]==("\"TF123\"")) {
            istringstream(sysData[1][loopCount]) >> TF123;
            varCount++;
        } else if (sysData[0][loopCount]==("\"TF124\"")) {
            istringstream(sysData[1][loopCount]) >> TF124;
            varCount++;
        } else if (sysData[0][loopCount]==("\"TF131\"")) {
            istringstream(sysData[1][loopCount]) >> TF131;
            varCount++;
        } else if (sysData[0][loopCount]==("\"TF132\"")) {
            istringstream(sysData[1][loopCount]) >> TF132;
            varCount++;
        } else if (sysData[0][loopCount]==("\"TF133\"")) {
            istringstream(sysData[1][loopCount]) >> TF133;
            varCount++;
        } else if (sysData[0][loopCount]==("\"TF134\"")) {
            istringstream(sysData[1][loopCount]) >> TF134;
            varCount++;
        } else if (sysData[0][loopCount]==("\"TF501\"")) {
            istringstream(sysData[1][loopCount]) >> TF501;
            varCount++;
        //} else if (sysData[0][loopCount]==("\"IO_FVM602T\"")) {
        } else if (sysData[0][loopCount]==("\"FVM602T_Steam_Temperature\"")) {
            istringstream(sysData[1][loopCount]) >> FVM602T;
            varCount++;
        //} else if (sysData[0][loopCount]==("\"IO_PT602\"")) {
```

```
    } else if (sysData[0][loopCount]==("\"PT602_StemPressure\"")) {
        istringstream(sysData[1][loopCount]) >> PT602;
        varCount++;
    } else if (loopCount==sysData[0].size()) {
        cerr << "something's funky!" << endl;
        break;
    }
    loopCount++;
}

singleTransientExplanation.push_back("Initial conditions perturbed");

//getting units right and varying initial conditions
srand(time(NULL)*8311344973*th_id);//resetting random numbers
U=(double)(rand())/(RAND_MAX);
U=(U/10)+0.90;

PT301=(PT301*6894.757*U);

srand(time(NULL)*2345745*th_id);//resetting random numbers
U=(double)(rand())/(RAND_MAX);
U=(U/10)+0.90;

PT511=(PT511*6894.757*U);

srand(time(NULL)*831176245*th_id);//resetting random numbers
U=(double)(rand())/(RAND_MAX);
U=(U/10)+0.90;

PT602=(PT602*6894.757*U);

//F to K (temp)
TF111=((TF111-32)*5/9)+273.15;
TF121=((TF121-32)*5/9)+273.15;
TF122=((TF122-32)*5/9)+273.15;
TF123=((TF123-32)*5/9)+273.15;
TF124=((TF124-32)*5/9)+273.15;
TF131=((TF131-32)*5/9)+273.15;
```

```
TF132=((TF132-32)*5/9)+273.15;
TF133=((TF133-32)*5/9)+273.15;
TF134=((TF134-32)*5/9)+273.15;
TF501=((TF501-32)*5/9)+273.15;
TFavg1=(TF121+TF122+TF123+TF124)*U/4;

srand(time(NULL)*987654321*th_id);//resetting random numbers
U=(double)(rand())/(RAND_MAX);
U=(U/10)+0.90;

TFavg2=(TF121+TF123+TF124)*U/3;
//converting to meters
LDP301=LDP301*U/39.3701;
//converting lbm/s to kg/s
FVM602M=FVM602M*U/2.205;

//comments section of input file
fout << "*" << string(70, '=') << "\n*\n*\n";
fout << "*" << string(3, ' ') << __DATE__ << endl;
fout << "*" << string(3, ' ') << "Written by " << author << endl;
fout << "*\n*\n*" << string(70, '=') << endl;
//100 card
fout << "100 "<< ProbType << " " << ProbOpt << endl;
//103 (restart) card
//RstNbr will change after the first time
fout << "103 " << RstNbr << endl;
//203 (time) card
fout << "203 " << showpoint << EndTime << ", "  << noshowpoint << MinTimeStep << ", "
    << showpoint << MaxTimeStep << ", " << noshowpoint << CtlMode << ", " << MinEdit
    << ", " << MajEdit << ", " << RstFreq << endl;

//-- Trips section
//This section reduces n trips into a form that R5 can understand
if ((stateVarTripNames.size()!=stateVarCodes.size()) ||
    (stateVarTripNames.size()!=stateVarEquiv.size()) ||
    (stateVarTripNames.size()!=yellowTripThresh.size()) ||
    (stateVarTripNames.size()!=redTripThresh.size())) {
    cerr << "RAPS input file error!" << endl <<
```

```
            "State variables, variable codes, equivalence, or thresholds are not of same size!"
            << endl;
    } else {
        for (unsigned int i=0; i<stateVarTripNames.size(); i++) {
            fout << 500 + i+1 << " " << showpoint << stateVarTripNames[i] << " " <<
                stateVarCodes[i] << " " << stateVarEquiv[i] << " null 0 "<<
                R5SciConv(redTripThresh[i]) << " l" << endl;
        }
        int index=0;
        int index60=1;
        int cardCount=0;
        int adder=0;
        while (index<(stateVarTripNames.size()/2)) { //collect up the 500's
            fout << 600 + index60 << " " << 500 + (index*2)+1 << " or " << 500 + (index*2)+2
                << " l -1.0" << endl;
            index60++;
            index++;
            if ((stateVarTripNames.size()%2==1) && (index==stateVarTripNames.size()/2)) {
                fout << 600 + index60 " " << 500 + (index*2)+1 << " or " << 500 + (index*2)+1
                    << " l -1.0" << endl;
                index60++;
            }
        }
        if(stateVarTripNames.size()>2) {
            index=0;
            cardCount=index60-1;
            for (int k=7; k<=stateVarTripNames.size(); k++) {
                if ((k%4)==3) {adder++;}
            }
            while (index<(adder + cardCount/2 + cardCount%2)) { //collect up the 600's
                fout << 600 + index60 << " " << 600 + (index*2)+1 << " or " << 600 + (index*2)+2
                    << " l -1.0" << endl;
                index60++;
                index++;
            }
        }
        if (stateVarTripNames.size()==1) {
            fout << "600 501" << endl;
```

```
    } else {
    fout << "600" << " " << 600 + index60-1 <<endl;
    }
}

//--- Next section is defining initial conditions from MASLWR data
fout << "*======================================================================" <<endl;
fout << "*                      Primary System" <<endl;
fout << "*======================================================================" <<endl;
fout << "*                      component 100" <<endl;
fout << "* Core including flow plates and regions 1-3" <<endl;
fout << "* Total Length from Problem Specification is 63.01 cm" <<endl;
fout << "*crdno            name            type" <<endl;
fout << "100       coreflow            pipe" <<endl;
fout << "*crdno              nv " <<endl;
fout << "101              6" <<endl;
fout << "*crdno            area     vol" <<endl;
fout << "1000101        8.422-3       6" <<endl;
fout << "*crdno            area     jun" <<endl;
fout << "1000201              0       5" <<endl;
fout << "*crdno          length     vol  " <<endl;
fout << "1000301          0.105       6" <<endl;
fout << "*crdno          volume     vol  " <<endl;
fout << "1000401              0       6" <<endl;
fout << "*crdno           h-ang     vol  " <<endl;
fout << "1000501            0.0       6" <<endl;
fout << "*crdno           v-ang     vol  " <<endl;
fout << "1000601           90.0       6" <<endl;
fout << "*crdno            delz     vol  " <<endl;
fout << "1000701          0.105       6" <<endl;
fout << "*crdno           rough     dhy     vol" <<endl;
fout << "1000801          2.0-6  9.59-3       6" <<endl;
fout << "*Additional Wall Friction from pg 386 Todreas and Kazimi" <<endl;
fout << "*crdno           A1 B1 C1 A2 B2 C2 A3 B3         C3 vol" <<endl;
fout << "1002601           0  0  0  0  0  0  0 0.146432 0.18   1" <<endl;
fout << "1002602           0  0  0  0  0  0  0 0.146432 0.18   2" <<endl;
fout << "1002603           0  0  0  0  0  0  0 0.146432 0.18   3" <<endl;
fout << "1002604           0  0  0  0  0  0  0 0.146432 0.18   4" <<endl;
```

```
fout << "1002605           0  0  0  0  0  0  0 0.146432 0.18   5" <<endl;
fout << "1002606           0  0  0  0  0  0  0 0.146432 0.18   6" <<endl;
fout << "*Junction 3 is used to simulate the extra losses at the grid wires" <<endl;
fout << "*crdno           floss   rloss     jun" <<endl;
fout << "1000901              0       0       2 " <<endl;
fout << "1000902           10.0    10.0       3" <<endl;
fout << "1000903              0       0       5  " <<endl;
fout << "*crdno             ctl     vol  " <<endl;
fout << "1001001              0       6" <<endl;
fout << "*crdno             ctl     jun  " <<endl;
fout << "1001101              0       5" <<endl;
fout << "*crdno             ctl       p       temp              vol" <<endl;
fout << "1001201              3 " <<  R5SciConv(PT301) << "    "
    << Linterpolate(TFavg1, TF132, 1, 6, 1) << "   0 0 0       1" <<endl;
fout << "1001202              3 " <<  R5SciConv(PT301) << "    "
    << Linterpolate(TFavg1, TF132, 1, 6, 2) << "   0 0 0       2" <<endl;
fout << "1001203              3 " <<  R5SciConv(PT301) << "    "
    << Linterpolate(TFavg1, TF132, 1, 6, 3) << "   0 0 0       3" <<endl;
fout << "1001204              3 " <<  R5SciConv(PT301) << "    "
    << Linterpolate(TFavg1, TF132, 1, 6, 4) << "   0 0 0       4" <<endl;
fout << "1001205              3 " <<  R5SciConv(PT301) << "    "
    << Linterpolate(TFavg1, TF132, 1, 6, 5) << "   0 0 0       5" <<endl;
fout << "1001206              3 " <<  R5SciConv(PT301) << "    "
    << Linterpolate(TFavg1, TF132, 1, 6, 6) << "   0 0 0       6" <<endl;
fout << "*==================================================================" <<endl;
fout << "*TOMNOTE: For all of these, pull data from PT-301 for the pressure column" <<endl;
fout << "*Multiply PT-301 by 6894.757 to get the right units" <<endl;
fout << "*For temperature, set volume 1 to TF-121, TF-122, TF-123, and TF-124" <<endl;
fout << "*Don't forget to convert to Kelvin for everything, K = (F * 5/9) □0.93" <<endl;
fout << "*Set Volume 6 temperature to TF-132" <<endl;
fout << "*For volumes 2-5, just do a straight linear average from volume 1 to 6" <<endl;
fout << "*==================================================================" <<endl;
fout << "*crdno             ctl         " <<endl;
fout << "1001300              1" <<endl;
fout << "*crdno          mflowf    mflowg      velj       jun" <<endl;
fout << "1001301           1.50       0.0         0         5" <<endl;
fout << "*-----------------------------------------------------------------" <<endl;
fout << "*                      component 110" <<endl;
```

```
fout << "* Entirety of the Hot Leg" <<endl;
fout << "*crdno            name            type" <<endl;
fout << "110         hotleg            pipe" <<endl;
fout << "*crdno              nv " <<endl;
fout << "111             29" <<endl;
fout << "*crdno            area     vol  " <<endl;
fout << "1100101        3.051-2       5" <<endl;
fout << "1100102        2.308-2       6" <<endl;
fout << "1100103        1.565-2       7" <<endl;
fout << "1100104        8.213-3      29" <<endl;
fout << "*crdno            area     jun" <<endl;
fout << "1100201              0      28" <<endl;
fout << "*crdno          length     vol  " <<endl;
fout << "1100301         0.0445       1" <<endl;
fout << "1100302         0.0762       3" <<endl;
fout << "1100303         0.1112       4" <<endl;
fout << "1100304         0.1111       5" <<endl;
fout << "1100306         0.1223       7" <<endl;
fout << "1100307         0.1020      14" <<endl;
fout << "1100308         0.1019      15" <<endl;
fout << "1100309         0.0540      16" <<endl;
fout << "1100310         0.0384      17" <<endl;
fout << "1100311         0.1011      18" <<endl;
fout << "1100312         0.1052      26" <<endl;
fout << "1100313         0.1050      27" <<endl;
fout << "1100314         0.1011      28" <<endl;
fout << "1100315         0.1433      29" <<endl;
fout << "*crdno          volume     vol  " <<endl;
fout << "1100401              0      29" <<endl;
fout << "*crdno           h-ang     vol  " <<endl;
fout << "1100501            0.0      29" <<endl;
fout << "*crdno           v-ang     vol  " <<endl;
fout << "1100601           90.0      29" <<endl;
fout << "*crdno           rough     dhy      vol" <<endl;
fout << "1100801          2.0-6  0.1971        5" <<endl;
fout << "1100802          2.0-6  0.1776        6" <<endl;
fout << "1100803          2.0-6  0.1488        7" <<endl;
fout << "1100804          2.0-6  0.1022       29" <<endl;
```

```
fout << "*crdno           floss   rloss     jun" <<endl;
fout << "1100901              0       0      16" <<endl;
fout << "1100902           10.0    10.0      17" <<endl;
fout << "1100903              0       0      28   " <<endl;
fout << "*crdno             ctl     vol  " <<endl;
fout << "1101001              0      29" <<endl;
fout << "*crdno             ctl     jun  " <<endl;
fout << "1101101              0      28" <<endl;
fout << "*crdno             ctl       p       temp             vol  " <<endl;
fout << "1101201              3 " <<  R5SciConv(PT301) << "    " << TF132
    << "   0 0 0      29" <<endl;
fout << "*======================================================================" <<endl;
fout << "*TOMNOTE: Set pressure to PT-301, temperature to TF-132" <<endl;
fout << "*======================================================================" <<endl;
fout << "*crdno             ctl        " <<endl;
fout << "1101300              1" <<endl;
fout << "*crdno          mflowf    mflowg      velj       jun" <<endl;
fout << "1101301           1.50       0.0         0        28" <<endl;
fout << "*----------------------------------------------------------------------" <<endl;
fout << "*                      component 300" <<endl;
fout << "* Upper Plenum" <<endl;
fout << "*crdno            name            type" <<endl;
fout << "300       luplenum          branch" <<endl;
fout << "*crdno      nj     ctl" <<endl;
fout << "301      2       1" <<endl;
fout << "*crdno    area  length  volume   h-ang   v-ang    delz   rough    " <<endl;
fout << "3000101  6.7-2  0.1205       0     0.0    90.0  0.1205   2.0-6 " <<endl;
fout << "*crdno     dhy     ctl" <<endl;
fout << "3000102  0.292       0" <<endl;
fout << "*crdno     ctl       p       temp" <<endl;
fout << "3000200      3  " <<  R5SciConv(PT301) << "   " << TF111 << endl;
fout << "*======================================================================" <<endl;
fout << "*TOMNOTE: Set pressure to PT-301, temperature to TF-111" <<endl;
fout << "*======================================================================" <<endl;
fout << "*crdno            from             to     area   floss   rloss      ctl" <<endl;
fout << "3001101      3      201010001  5.675-2       0       0        0 " <<endl;
fout << "3002101      30001       301010001   6.70-2       0       0        0" <<endl;
fout << "*3003101      110290002       3  8.213-3       0       0        0" <<endl;
```

```
fout << "*crdno  mflowf  mflowg    velj" <<endl;
fout << "3001201   1.50     0.0       0 " <<endl;
fout << "3002201    0.0     0.0       0" <<endl;
fout << "*3003201   1.50     0.0       0" <<endl;
fout << "*----------------------------------------------------------------------" <<endl;
fout << "*                      component 301" <<endl;
fout << "* Pressurizer " <<endl;
fout << "*crdno            name            type" <<endl;
fout << "301            pzr            pipe" <<endl;
fout << "*crdno              nv " <<endl;
fout << "3010001              8" <<endl;
fout << "*crdno            area     vol  " <<endl;
fout << "3010101          6.70-2       1" <<endl;
fout << "3010102          4.05-3       2" <<endl;
fout << "3010103          6.70-2       8" <<endl;
fout << "*3010104         5.025-2       9" <<endl;
fout << "*crdno            area     jun" <<endl;
fout << "3010201              0       7" <<endl;
fout << "*crdno          length     vol  " <<endl;
fout << "3010301          0.0881       1" <<endl;
fout << "3010302          0.0352       2" <<endl;
fout << "3010303          0.0560       3" <<endl;
fout << "3010304          0.1168       6" <<endl;
fout << "3010305          0.0868       8" <<endl;
fout << "*3010306          0.1          9" <<endl;
fout << "*crdno          volume     vol  " <<endl;
fout << "3010401              0       8" <<endl;
fout << "*crdno           h-ang     vol  " <<endl;
fout << "3010501            0.0       8" <<endl;
fout << "*crdno           v-ang     vol  " <<endl;
fout << "3010601           90.0       8" <<endl;
fout << "*crdno           rough    dhy     vol" <<endl;
fout << "3010801           2.0-6   0.292       1" <<endl;
fout << "3010802           2.0-6   2.54-2      2" <<endl;
fout << "3010803           2.0-6   0.292       8" <<endl;
fout << "*3010804           2.0-6   0.219       9" <<endl;
fout << "*crdno           floss   rloss     jun" <<endl;
fout << "3010901            20.0    20.0       1" <<endl;
```

```
fout << "3010902            0.0     0.0       7  " <<endl;
fout << "*crdno             ctl     vol  " <<endl;
fout << "3011001              0       8" <<endl;
fout << "*crdno             ctl     jun  " <<endl;
fout << "3011101              0       1" <<endl;
fout << "3011102              0       2" <<endl;
fout << "3011103              0       3" <<endl;
fout << "3011104              0       4" <<endl;
fout << "3011105              0       5" <<endl;
fout << "3011106              0       6" <<endl;
fout << "3011107              0       7" <<endl;
fout << "*crdno             ctl       p         qual             vol" <<endl;
fout << "3011201              2  " <<  R5SciConv(PT301) << "     "
    << qualConverter(LDP301, 1) << "   0 0 0       1  " <<endl;
fout << "3011202              2  " <<  R5SciConv(PT301) << "     "
    << qualConverter(LDP301, 2) << "   0 0 0       2" <<endl;
fout << "3011203              2  " <<  R5SciConv(PT301) << "     "
    << qualConverter(LDP301, 3) << "   0 0 0       3" <<endl;
fout << "3011204              2  " <<  R5SciConv(PT301) << "     "
    << qualConverter(LDP301, 4) << "   0 0 0       4" <<endl;
fout << "3011205              2  " <<  R5SciConv(PT301) << "     "
    << qualConverter(LDP301, 5) << "   0 0 0       5" <<endl;
fout << "3011206              2  " <<  R5SciConv(PT301) << "     "
    << qualConverter(LDP301, 6) << "   0 0 0       6" <<endl;
fout << "3011207              2  " <<  R5SciConv(PT301) << "     "
    << qualConverter(LDP301, 7) << "   0 0 0       7" <<endl;
fout << "3011208              2  " <<  R5SciConv(PT301) << "     "
    << qualConverter(LDP301, 8) << "   0 0 0       8" <<endl;
fout << "*======================================================================" <<endl;
fout << "*TOMNOTE: Set pressure to PT-301" <<endl;
fout << "*======================================================================" <<endl;
fout << "*crdno             ctl        " <<endl;
fout << "3011300              1" <<endl;
fout << "*crdno           flowf     flowg      velj       jun" <<endl;
fout << "3011301            0.0       0.0         0         7" <<endl;
fout << "*----------------------------------------------------------------------" <<endl;
fout << "*                      component 302" <<endl;
fout << "* ADS Vent Line Steam Space" <<endl;
```

```cpp
fout << "*crdno            name            type" <<endl;
fout << "302       PZRsteam          branch" <<endl;
fout << "*crdno      nj     " <<endl;
fout << "3020001      0" <<endl;
fout << "*crdno     area  length  volume   h-ang   v-ang    delz    rough    " <<endl;
fout << "3020101 5.025-2    0.10       0     0.0    90.0    0.10    2.0-6" <<endl;
fout << "*crdno      dhy    ctl" <<endl;
fout << "3020102  0.219       0" <<endl;
fout << "*crdno     ctl      p    qual" <<endl;
fout << "3020200      2 " <<   R5SciConv(PT301) << "    1.0  " <<endl;
fout << "*======================================================================" <<endl;
fout << "*TOMNOTE: Set pressure to PT-301 as usual" <<endl;
fout << "*======================================================================" <<endl;
fout << "*----------------------------------------------------------------------" <<endl;
fout << "*                      component 201" <<endl;
fout << "* Cold Leg" <<endl;
fout << "*crdno            name            type" <<endl;
fout << "201        coldleg            pipe" <<endl;
fout << "*crdno              nv " <<endl;
fout << "2010001             35" << endl;
fout << "*crdno            area     vol  " << endl;
fout << "2010101        5.675-2       1" << endl;
fout << "2010102        4.564-2       2" << endl;
fout << "2010103        4.114-2      11" << endl;
fout << "2010104        4.564-2      12" << endl;
fout << "2010105        5.675-2      22" << endl;
fout << "2010106        4.936-2      23" << endl;
fout << "2010107        4.197-2      24" << endl;
fout << "2010108        3.458-2      35" << endl;
fout << "*crdno            area     jun" << endl;
fout << "2010201              0      34" << endl;
fout << "*crdno          length     vol  " << endl;
fout << "2010301         0.1433       1" << endl;
fout << "2010302         0.1011       2" << endl;
fout << "2010303         0.1050       3" << endl;
fout << "2010304         0.1052      11" << endl;
fout << "2010305         0.1011      12" << endl;
fout << "2010306         0.0384      13" << endl;
```

```
fout << "2010307         0.0540      14" << endl;
fout << "2010308         0.1019      15" << endl;
fout << "2010309         0.1020      22" << endl;
fout << "2010310         0.1223      24" << endl;
fout << "2010311         0.1111      25" << endl;
fout << "2010312         0.1112      26" << endl;
fout << "2010313         0.0762      28" << endl;
fout << "2010314         0.0445      29" << endl;
fout << "2010315         0.1050      35" << endl;
fout << "*crdno          volume     vol  " << endl;
fout << "2010401              0      35" << endl;
fout << "*crdno           h-ang     vol  " << endl;
fout << "2010501            0.0      35" << endl;
fout << "*crdno           v-ang     vol  " << endl;
fout << "2010601          -90.0      35" << endl;
fout << "*crdno           rough     dhy      vol" << endl;
fout << "2010801          2.0-6  0.1778        1" << endl;
fout << "2010802          2.0-6  4.474-2       2" << endl;
fout << "2010803          2.0-6  3.156-2      11" << endl;
fout << "2010804          2.0-6  4.474-2      12" << endl;
fout << "2010805          2.0-6  0.1778       22" << endl;
fout << "2010806          2.0-6  0.1441       23" << endl;
fout << "2010807          2.0-6  0.1148       24" << endl;
fout << "2010808          2.0-6  8.89-2       35" << endl;
fout << "*crdno           floss   rloss      jun" << endl;
fout << "2010901              0       0      34   " << endl;
fout << "*crdno             ctl     vol  " << endl;
fout << "2011001              0      35" << endl;
fout << "*crdno             ctl     jun  " << endl;
fout << "2011101              0      34" << endl;
fout << "*crdno             ctl       p       temp             vol  " << endl;
fout << "2011201              3  " << R5SciConv(PT301) << "   "
    << Linterpolate(TF111, TFavg2, 1, 12, 1) << "   0 0 0       1" << endl;
fout << "2011202              3  " << R5SciConv(PT301) << "   "
    << Linterpolate(TF111, TFavg2, 1, 12, 2) << "   0 0 0       2" << endl;
fout << "2011203              3  " << R5SciConv(PT301) << "   "
    << Linterpolate(TF111, TFavg2, 1, 12, 3) << "   0 0 0       3" << endl;
fout << "2011204              3  " << R5SciConv(PT301) << "   "
```

```
    << Linterpolate(TF111, TFavg2, 1, 12, 4) << "   0 0 0       4" << endl;
fout << "2011205              3  " << R5SciConv(PT301) << "   "
    << Linterpolate(TF111, TFavg2, 1, 12, 5) << "   0 0 0       5" << endl;
fout << "2011206              3  " << R5SciConv(PT301) << "   "
    << Linterpolate(TF111, TFavg2, 1, 12, 6) << "   0 0 0       6" << endl;
fout << "2011207              3  " << R5SciConv(PT301) << "   "
    << Linterpolate(TF111, TFavg2, 1, 12, 7) << "   0 0 0       7" << endl;
fout << "2011208              3  " << R5SciConv(PT301) << "   "
    << Linterpolate(TF111, TFavg2, 1, 12, 8) << "   0 0 0       8" << endl;
fout << "2011209              3  " << R5SciConv(PT301) << "   "
    << Linterpolate(TF111, TFavg2, 1, 12, 9) << "   0 0 0       9" << endl;
fout << "2011210              3  " << R5SciConv(PT301) << "   "
    << Linterpolate(TF111, TFavg2, 1, 12, 10) << "   0 0 0      10" << endl;
fout << "2011211              3  " << R5SciConv(PT301) << "   "
    << Linterpolate(TF111, TFavg2, 1, 12, 11) << "   0 0 0      11" << endl;
fout << "2011212              3  " << R5SciConv(PT301) << "   "
    << Linterpolate(TF111, TFavg2, 1, 12, 12) << "   0 0 0      12" << endl;
fout << "2011213              3  " << R5SciConv(PT301)
    << "   " << TFavg2 << "   0 0 0      35" << endl;
fout << "*======================================================================" << endl;
fout << "*TOMNOTE: Set pressure to PT-301" << endl;
fout << "*======================================================================" << endl;
fout << "*crdno             ctl        " << endl;
fout << "2011300              1" << endl;
fout << "*crdno          mflowf    mflowg      velj       jun" << endl;
fout << "2011301           1.50       0.0         0        34" << endl;
fout << "*----------------------------------------------------------------------" << endl;
fout << "*                      component 202" << endl;
fout << "* Lower Plenum " << endl;
fout << "*crdno            name            type" << endl;
fout << "202        lplenum          branch" << endl;
fout << "*crdno      nj     ctl" << endl;
fout << "2020001      0       1" << endl;
fout << "*crdno    area  length  volume   h-ang   v-ang    delz   rough    " << endl;
fout << "2020101      0   6.2-2  2.63-3     0.0   -90.0  -6.2-2   2.0-6" << endl;
fout << "*crdno     dhy     ctl" << endl;
fout << "2020102 7.61-2       0" << endl;
fout << "*crdno     ctl      p        temp" << endl;
```

```
fout << "2020200      3  " << R5SciConv(PT301) << "   " << TFavg2 << endl;
fout << "*======================================================================" << endl;
fout << "*TOMNOTE: Set pressure to PT-301" << endl;
fout << "*Set temperature to TF-131, TF-133, TF-134" << endl;
fout << "*======================================================================" << endl;
fout << "*======================================================================" << endl;
fout << "*                     Secondary System" << endl;
fout << "*======================================================================" << endl;
fout << "*----------------------------------------------------------------------" << endl;
fout << "*                      component 510" << endl;
fout << "* Feedwater Pump Cheat " << endl;
fout << "*crdno            name            type" << endl;
fout << "510         feedw1         tmdpvol" << endl;
fout << "*crdno    area  length  volume   h-ang   v-ang    delz   rough    " << endl;
fout << "5100101 6.94-5     1.0       0     0.0    90.0     1.0     0.0" << endl;
fout << "*crdno     dhy     ctl" << endl;
fout << "5100102    0.0       0" << endl;
fout << "*crdno     ctl     " << endl;
fout << "5100200      3       " << endl;
fout << "*crdno    time    pres       temp    time    pres      temp" << endl;
fout << "5100201    0.0   "<< R5SciConv(PT511) << "  " << TF501 << "   362.0  "
    << R5SciConv(PT511) << "  " << TF501 << endl;
fout << "5100202  376.0   "<< R5SciConv(PT511) << "  " << TF501 << "   774.0  "
    << R5SciConv(PT511) << "  " << TF501 << endl;
fout << "5100203  780.0   "<< R5SciConv(PT511) << "  " << TF501 << "  1476.0  "
    << R5SciConv(PT511) << "  " << TF501 << endl;
fout << "5100204 1503.0   "<< R5SciConv(PT511) << "  " << TF501 << "   5.0+4  "
    << R5SciConv(PT511) << "  " << TF501 << endl;
fout << "*======================================================================" << endl;
fout << "*TOMNOTE: Set pressure to PT-511" << endl;
fout << "*======================================================================" << endl;
fout << "*----------------------------------------------------------------------" << endl;
fout << "*                      component 051" << endl;
fout << "* Feedwater Supply" << endl;
fout << "*crdno            name            type" << endl;
fout << "051           fwin         tmdpjun" << endl;
fout << "*crdno            from              to     area   " << endl;
fout << "0510101      51001       500010001        0       " << endl;
```

```
fout << "*crdno     ctl     " << endl;
fout << "0510200      1      " << endl;
fout << "*crdno    time      flowf   flowv    velj     time     flowf  flowv   velj" << endl;
fout << "0510201    0.0   "<< FVM602M << "  0.0     0.0    5.0+4  "
    << FVM602M << "  0.0   0.0" << endl;
fout << "0510202  5.0+4   "<< FVM602M << "  0.0     0.0    5.0+5  "
    << FVM602M << "  0.0   0.0" << endl;
fout << "*==================================================================" << endl;
fout << "*TOMNOTE: Set all of the flowfs to FVM-602M" << endl;
fout << "*The number is in lbm/s, divide by 2.205 to switch to kg/s" << endl;
fout << "*==================================================================" << endl;
fout << "*------------------------------------------------------------------" << endl;
fout << "*                      component 500" << endl;
fout << "* Feedwater line before FRV" << endl;
fout << "*crdno            name            type" << endl;
fout << "500         fwline            pipe" << endl;
fout << "*crdno      nv " << endl;
fout << "501      2" << endl;
fout << "*crdno    area     vol  " << endl;
fout << "5000101 6.94-5       2" << endl;
fout << "*crdno    area     jun" << endl;
fout << "5000201      0       1" << endl;
fout << "*crdno  length     vol  " << endl;
fout << "5000301    0.25      2" << endl;
fout << "*crdno  volume     vol  " << endl;
fout << "5000401      0       2" << endl;
fout << "*crdno   h-ang     vol  " << endl;
fout << "5000501    0.0       2" << endl;
fout << "*crdno   v-ang     vol  " << endl;
fout << "5000601   90.0       2" << endl;
fout << "*crdno    delz     vol  " << endl;
fout << "5000701   0.25       2" << endl;
fout << "*crdno   rough     dhy     vol" << endl;
fout << "5000801  2.0-6  0.0127       2" << endl;
fout << "*crdno   floss   rloss     jun" << endl;
fout << "*INPUT LOSS COEFFICIENTS FOR PIPE BENDS" << endl;
fout << "5000901      0       0       1   " << endl;
fout << "*crdno     ctl     vol  " << endl;
```

```
fout << "5001001      0       2" << endl;
fout << "*crdno     ctl     jun  " << endl;
fout << "5001101      0       1" << endl;
fout << "*crdno     ctl       p      temp                  vol" << endl;
fout << "5001201      3  "<< R5SciConv(PT511) << "  " << TF501 << "   0 0 0    2" << endl;
fout << "*======================================================================" << endl;
fout << "*TOMNOTE: Set pressure to PT-511" << endl;
fout << "*Set temperature to TF-501" << endl;
fout << "*======================================================================" << endl;
fout << "*crdno     ctl        " << endl;
fout << "5001300      1" << endl;
fout << "*crdno  mflowf   mflowg    velj     jun" << endl;
fout << "5001301 4.50-1      0.0       0       1" << endl;
fout << "*----------------------------------------------------------------------" << endl;
fout << "*                      component 053" << endl;
fout << "* Feedwater Regulating Valve" << endl;
fout << "*crdno           name            type" << endl;
fout << "053            msv           valve" << endl;
fout << "*crdno           from              to     area    floss    rloss      ctl" << endl;
fout << "0530101      500020002       501010001  6.94-5       0       0     1100" << endl;
fout << "*crdno     ctl      flowf   flowg    velj" << endl;
fout << "0530201      1   "<< FVM602M << "     0.0     0.0  " << endl;
fout << "*======================================================================" << endl;
fout << "*TOMNOTE: Set flowf to FVM-602M" << endl;
fout << "*======================================================================" << endl;
fout << "*crdno    type     " << endl;
fout << "0530300 srvvlv  " << endl;
fout << "*crdno   ctlno   table no" << endl;
fout << "0530301      4         0" << endl;
fout << "*----------------------------------------------------------------------" << endl;
fout << "*                      component 501" << endl;
fout << "* Single Feedwater Line for Modeling of FRV to Branch before SGs " << endl;
fout << "*crdno           name            type" << endl;
fout << "501         fwline            pipe" << endl;
fout << "*crdno      nv " << endl;
fout << "5010001      4" << endl;
fout << "*crdno    area     vol" << endl;
fout << "5010101 1.24-4       4" << endl;
```

```
fout << "*crdno    area     jun" << endl;
fout << "5010201      0       3" << endl;
fout << "*crdno  length     vol" << endl;
fout << "5010301   1.53       1" << endl;
fout << "5010302   0.61       2" << endl;
fout << "5010303   0.22       3" << endl;
fout << "5010304   0.16       4" << endl;
fout << "*crdno  volume     vol" << endl;
fout << "5010401      0       4" << endl;
fout << "*crdno   h-ang     vol" << endl;
fout << "5010501    0.0       1" << endl;
fout << "5010502    0.0       2" << endl;
fout << "5010503    0.0       3" << endl;
fout << "5010504    0.0       4" << endl;
fout << "*crdno   v-ang     vol" << endl;
fout << "5010601   90.0       1" << endl;
fout << "5010602    0.0       2" << endl;
fout << "5010603   90.0       3" << endl;
fout << "5010604    0.0       4" << endl;
fout << "*crdno    delz     vol" << endl;
fout << "5010701   1.53       1" << endl;
fout << "5010702    0.0       2" << endl;
fout << "5010703   0.22       3" << endl;
fout << "5010704    0.0       4" << endl;
fout << "*crdno   rough     dhy     vol" << endl;
fout << "5010801  2.0-6  0.0127       4" << endl;
fout << "*crdno   floss   rloss     jun" << endl;
fout << "*INPUT LOSS COEFFICIENTS FOR PIPE BENDS" << endl;
fout << "5010901      0       0       3   " << endl;
fout << "*crdno     ctl     vol" << endl;
fout << "5011001      0       4" << endl;
fout << "*crdno     ctl     jun" << endl;
fout << "5011101      0       3" << endl;
fout << "*crdno     ctl      p       temp                      vol" << endl;
fout << "5011201      3  "<< R5SciConv(PT511) << "  " << TF501 << "      0 0 0 4" << endl;
fout << "*======================================================================" << endl;
fout << "*TOMNOTE: Set pressure to PT-511" << endl;
fout << "*Set temperature to TF-501" << endl;
```

```
fout << "*======================================================================" << endl;
fout << "*crdno     ctl        " << endl;
fout << "5011300      1" << endl;
fout << "*crdno  mflowf   mflowg    velj     jun" << endl;
fout << "5011301 4.50-1      0.0       0       3" << endl;
fout << "*----------------------------------------------------------------------" << endl;
fout << "*                      component 057" << endl;
fout << "* Junction with SG inlet" << endl;
fout << "*crdno            name            type" << endl;
fout << "057           SGin           valve" << endl;
fout << "*crdno            from              to    area   floss   rloss      ctl" << endl;
fout << "0570101      501040002       505010001       0       0       0        0" << endl;
fout << "*crdno     ctl      flowf   flowg    velj" << endl;
fout << "0570201      1   "<< FVM602M << "     0.0     0.0" << endl;
fout << "*======================================================================" << endl;
fout << "*TOMNOTE: Set flowf to FVM-602M" << endl;
fout << "*======================================================================" << endl;
fout << "*crdno     valve" << endl;
fout << "0570300    trpvlv" << endl;
fout << "*crdno     trip --- 498 is OPEN, 499 is CLOSED" << endl;
fout << "0570301    498" << endl;
fout << "*----------------------------------------------------------------------" << endl;
fout << "*                      component 505" << endl;
fout << "* Steam generator--tube " << endl;
fout << "*crdno            name            type" << endl;
fout << "505         SGtube            pipe" << endl;
fout << "*crdno      nv " << endl;
fout << "5050001     11" << endl;
fout << "*crdno    area     vol  " << endl;
fout << "5050101  1.746-3    11" << endl;
fout << "*crdno    area     jun" << endl;
fout << "5050201      0      10" << endl;
fout << "*crdno  length     vol  " << endl;
fout << "5050301 0.35         1" << endl;
fout << "5050302 0.6048      10" << endl;
fout << "5050303 0.35        11" << endl;
fout << "*crdno  volume     vol  " << endl;
fout << "5050401      0      11" << endl;
```

```
    fout << "*crdno   h-ang     vol   " << endl;
    fout << "5050501    0.0      11" << endl;
    fout << "*crdno   v-ang     vol   " << endl;
    fout << "5050601   90.0      11" << endl;
    fout << "*crdno    delz     vol   " << endl;
    fout << "5050701 0.06545      1" << endl;
    fout << "5050702 0.1131      10" << endl;
    fout << "5050703 0.06545     11" << endl;
    fout << "*crdno   rough     dhy     vol" << endl;
    fout << "5050801  2.0-6  0.0126      11" << endl;
    fout << "*crdno   floss   rloss     jun" << endl;
    fout << "5050901      0       0      10   " << endl;
    fout << "*crdno     ctl     vol   " << endl;
    fout << "5051001      0      11" << endl;
    fout << "*crdno     ctl     jun   " << endl;
    fout << "5051101      0      10" << endl;
    fout << "*crdno     ctl       p       temp                  vol  " << endl;
    fout << "5051201      3  " << R5SciConv(PT511) << "   "
        << Linterpolate(TF501, FVM602T, 1, 11, 1) << "      0 0 0         1" << endl;
    fout << "5051202      3  " << R5SciConv(PT511) << "   "
        << Linterpolate(TF501, FVM602T, 1, 11, 2) << "      0 0 0         2" << endl;
    fout << "5051203      3  " << R5SciConv(PT511) << "   "
        << Linterpolate(TF501, FVM602T, 1, 11, 3) << "      0 0 0         3" << endl;
    fout << "5051204      3  " << R5SciConv(PT511) << "   "
        << Linterpolate(TF501, FVM602T, 1, 11, 4) << "      0 0 0         4" << endl;
    fout << "5051205      3  " << R5SciConv(PT511) << "   "
        << Linterpolate(TF501, FVM602T, 1, 11, 5) << "      0 0 0         5" << endl;
    fout << "5051206      3  " << R5SciConv(PT511) << "   "
        << Linterpolate(TF501, FVM602T, 1, 11, 6) << "      0 0 0         6" << endl;
    fout << "5051207      3  " << R5SciConv(PT511) << "   "
        << Linterpolate(TF501, FVM602T, 1, 11, 7) << "      0 0 0         7" << endl;
    fout << "5051208      3  " << R5SciConv(PT511) << "   "
        << Linterpolate(TF501, FVM602T, 1, 11, 8) << "      0 0 0         8" << endl;
    fout << "5051209      3  " << R5SciConv(PT511) << "   "
        << Linterpolate(TF501, FVM602T, 1, 11, 9) << "      0 0 0         9" << endl;
    fout << "5051210      3  " << R5SciConv(PT511) << "   "
        << Linterpolate(TF501, FVM602T, 1, 11, 10) << "      0 0 0        10" << endl;
    fout << "5051211      3  " << R5SciConv(PT511) << "   "
```

```
    << Linterpolate(TF501, FVM602T, 1, 11, 11) << "      0 0 0        11" << endl;
fout << "*======================================================================" << endl;
fout << "*TOMNOTE: Set pressure to PT-511" << endl;
fout << "*Set volume 1 temperature to TF-501" << endl;
fout << "*Set volume 11 temperature to FVM-602T" << endl;
fout << "*Linear Interpolation for the others" << endl;
fout << "*======================================================================" << endl;
fout << "*crdno     ctl        " << endl;
fout << "5051300      1" << endl;
fout << "*crdno   flowf   flowg    velj     jun" << endl;
fout << "5051301 4.50-1     0.0       0       6" << endl;
fout << "5051302    0.0  4.50-1       0      10" << endl;
fout << "*----------------------------------------------------------------------" << endl;
fout << "*                      component 058" << endl;
fout << "* Junction with SG Outlet to Main Steam Line" << endl;
fout << "*crdno           name            type" << endl;
fout << "058          SGout           valve" << endl;
fout << "*crdno           from              to     area   floss   rloss      ctl" << endl;
fout << "0580101      505110004       507010001       0       0       0        0" << endl;
fout << "*crdno     ctl      flowf   flowg    velj" << endl;
fout << "0580201      1   "<< FVM602M << "  4.50-1     0.0" << endl;
fout << "*======================================================================" << endl;
fout << "*TOMNOTE: Set flowf to FVM-602M" << endl;
fout << "*======================================================================" << endl;
fout << "*crdno     valve" << endl;
fout << "0580300    trpvlv" << endl;
fout << "*crdno     trip --- 498 is OPEN, 499 is CLOSED" << endl;
fout << "0580301    498" << endl;
fout << "*----------------------------------------------------------------------" << endl;
fout << "*                      component 507" << endl;
fout << "* Main Steam Line from Steam Drum to Vortex Meter Inlet" << endl;
fout << "*crdno           name            type" << endl;
fout << "507         MSteam            pipe" << endl;
fout << "*crdno      nv " << endl;
fout << "5070001      8" << endl;
fout << "*crdno    area     vol  " << endl;
fout << "5070101 9.64-4       8" << endl;
fout << "*crdno    area     jun" << endl;
```

```
fout << "5070201      0       7" << endl;
fout << "*crdno  length     vol  " << endl;
fout << "5070301    0.2       1" << endl;
fout << "5070302   0.35       2" << endl;
fout << "5070303   0.34       3" << endl;
fout << "5070304   0.10       4" << endl;
fout << "5070305   0.26       8" << endl;
fout << "*crdno  volume     vol  " << endl;
fout << "5070401      0       8" << endl;
fout << "*crdno   h-ang     vol  " << endl;
fout << "5070501    0.0       3" << endl;
fout << "5070502    0.0       4" << endl;
fout << "5070503    0.0       8" << endl;
fout << "*crdno   v-ang     vol  " << endl;
fout << "5070601    0.0       3" << endl;
fout << "5070602   90.0       4" << endl;
fout << "5070603    0.0       8" << endl;
fout << "*crdno    delz     vol  " << endl;
fout << "5070701    0.0       3" << endl;
fout << "5070702    0.1       4" << endl;
fout << "5070703    0.0       8" << endl;
fout << "*crdno   rough     dhy       vol" << endl;
fout << "5070801  2.0-6   3.5-2       8" << endl;
fout << "*crdno   floss   rloss     jun" << endl;
fout << "*90 degree pipe bend loss" << endl;
fout << "5070901      0       0       1   " << endl;
fout << "5070902      0       0       2" << endl;
fout << "*180 pipe bend loss          " << endl;
fout << "5070903      0       0       3" << endl;
fout << "5070904      0       0       7" << endl;
fout << "*crdno     ctl     vol  " << endl;
fout << "5071001      0       8" << endl;
fout << "*crdno     ctl     jun  " << endl;
fout << "5071101      0       7" << endl;
fout << "*crdno     ctl       p       temp                  vol  " << endl;
fout << "5071201      3  "<< R5SciConv(PT602) << "   " << FVM602T <<"    0 0 0 8" << endl;
fout << "*======================================================================" << endl;
fout << "*TOMNOTE: Set pressure to PT-602" << endl;
```

```
fout << "*Set temperature to FVM-602T" << endl;
fout << "*======================================================================" << endl;
fout << "*crdno     ctl        " << endl;
fout << "5071300      1" << endl;
fout << "*crdno   flowf   flowg    velj     jun" << endl;
fout << "5071301    0.0  4.50-1       0       7" << endl;
fout << "*----------------------------------------------------------------------" << endl;
fout << "*                      component 060" << endl;
fout << "* Main Steam Outlet to Back Pressure Regulator" << endl;
fout << "*crdno            name            type" << endl;
fout << "060          BPRin         sngljun" << endl;
fout << "*crdno            from              to    area   floss   rloss      ctl" << endl;
fout << "0600101      507080004       50800       0       0       0        0" << endl;
fout << "*crdno     ctl   flowf      flowg    velj" << endl;
fout << "0600201      1     0.0   "<< FVM602M << "     0.0" << endl;
fout << "*======================================================================" << endl;
fout << "*TOMNOTE: Set flowg (NOT flowf) to FVM-602M" << endl;
fout << "*======================================================================" << endl;
fout << "*----------------------------------------------------------------------" << endl;
fout << "*                      component 508" << endl;
fout << "* Steam generator--secondary outlet-Backpressure Regulator " << endl;
fout << "*crdno            name            type" << endl;
fout << "508       SGoutlet         tmdpvol" << endl;
fout << "*crdno    area  length  volume   h-ang   v-ang    delz   rough    " << endl;
fout << "5080101   0.01     1.0       0     0.0    90.0     1.0     0.0  " << endl;
fout << "*crdno     dhy     ctl" << endl;
fout << "5080102    0.0       0" << endl;
fout << "*crdno     ctl     " << endl;
fout << "5080200      2       " << endl;
fout << "*crdno    time      p    qual    time      p    qual" << endl;
fout << "5080201    0.0  "<< R5SciConv(PT602) << "   1.0   362.0  "
    << R5SciConv(PT602) << "   1.0" << endl;
fout << "5080202  376.0  "<< R5SciConv(PT602) << "   1.0   774.0  "
    << R5SciConv(PT602) << "   1.0" << endl;
fout << "5080203  780.0  "<< R5SciConv(PT602) << "   1.0  1476.0  "
    << R5SciConv(PT602) << "   1.0" << endl;
fout << "5080204 1503.0  "<< R5SciConv(PT602) << "   1.0  1779.0  "
    << R5SciConv(PT602) << "   1.0" << endl;
```

```
fout << "5080205 1786.0  "<< R5SciConv(PT602) << "   1.0   5.0+4  "
    << R5SciConv(PT602) << "   1.0" << endl;
fout << "*======================================================================" << endl;
fout << "*TOMNOTE: Set pressure to PT-602 for all times" << endl;
fout << "*Leave quality at 1.0" << endl;
fout << "*======================================================================" << endl;
//--

if (th_id>=(requestTh/2)) { //only second half of threads are used for transients
    for (int i=0; i<(transient.size()-2); i++) {
        if (transient[i]==("VV_O")) { //Vent Valve open
            fout << "* RAPSS simulating VentValve Open" << endl;
            singleTransientExplanation.push_back("RAPSS simulating VentValve Open");
            numOfValves=2;
            srand(time(NULL));
            switch(rand()%(numOfValves-1)+1) {
                case 1:
                    Vbreak.push_back(4720);
                    break;
                case 2:
                    Vbreak.push_back(4820);
            }
        }
        if (transient[i]==("VV_C")) { //Vent Valve closed
            fout << "* RAPSS simulating VentValve closed" << endl;
            singleTransientExplanation.push_back("RAPSS simulating VentValve closed");
            numOfValves=3;
            srand(time(NULL));
            switch(rand()%(numOfValves-1)+1) {
                case 1:
                    Vbreak.push_back(4721);
                    break;
                case 2:
                    Vbreak.push_back(4821);
                    break;
                case 3:
                    Vbreak.push_back(90);
            }
```

```
        }
        if (transient[i]==("HCC_F")) { //Hot channel chimney hole
            fout << "* RAPSS simulating Hot channel chimney hole" << endl;
            singleTransientExplanation.push_back("RAPSS simulating Hot channel chimney hole");
            Vbreak.push_back(7510);
        }
        if (transient[i]==("RPV_F")) { //RPV leak
            fout << "* RAPSS simulating RPV leak" << endl;
            singleTransientExplanation.push_back("RAPSS simulating RPV leak");
            Vbreak.push_back(7520);
        }
        if (transient[i]==("CONT_F")) { //Containment Leak
            fout << "* RAPSS simulating Containment Leak" << endl;
            singleTransientExplanation.push_back("RAPSS simulating Containment Leak");
            Vbreak.push_back(7540);
        }
        if (transient[i]==("FLOW_B")) { //Flow blockage
            fout << "* RAPSS simulating Flow Blockage" << endl;
            singleTransientExplanation.push_back("RAPSS simulating Flow Blockage");
            numOfValves=4;
            srand(time(NULL));
            switch(rand()%(numOfValves-1)+1) {
                case 1:
                    Vbreak.push_back(0010);
                    break;
                case 2:
                    Vbreak.push_back(0020);
                    break;
                case 3:
                    Vbreak.push_back(0030);
                    break;
                case 4:
                    Vbreak.push_back(0040);
            }
        }
        if (transient[i]==("FLOW_B1")) { //partial Flow blockage
            fout << "* RAPSS simulating partial Flow Blockage" << endl;
```

```
            singleTransientExplanation.push_back("RAPSS simulating partial Flow Blockage");
            numOfValves=4;
            srand(time(NULL));
            switch(rand()%(numOfValves-1)+1) {
                case 1:
                    Vbreak.push_back(0011);
                    break;
                case 2:
                    Vbreak.push_back(0021);
                    break;
                case 3:
                    Vbreak.push_back(0031);
                    break;
                case 4:
                    Vbreak.push_back(0041);
            }
        }
        if (transient[i]==("SWMup_F")) { //Sump Water Makeup failure
            fout << "* RAPSS simulating Sump Water Makeup failure" << endl;
            singleTransientExplanation.push_back("RAPSS simulating Sump H20 Makeup failure");
            Vbreak.push_back(4520);
            Vbreak.push_back(4620);
        }
        if (transient[i]==("SUMP_O")) { //Sump Open
            fout << "* RAPSS simulating sump open" << endl;
            singleTransientExplanation.push_back("RAPSS simulating Sump Open");
            Vbreak.push_back(4521);
            Vbreak.push_back(4621);
        }
        if (transient[i]==("SECOND_F")) { //secondary loop failure
            fout << "* RAPSS secondary loop failure" << endl;
            singleTransientExplanation.push_back("RAPSS simulating Secondary Loop Failure");
            numOfValves=2;
            srand(time(NULL));
            switch(rand()%(numOfValves-1)+1) {
                case 1:
                    Vbreak.push_back(0570);
                    break;
```

```
                    case 2:
                        Vbreak.push_back(580);
                }
            }

        }
    }
        for (unsigned int i=0; i<Vbreak.size(); i++)  {
            if (prevKeepGoing[th_id]==0) {  //if keep going is true, don't start a new transient
                switch (Vbreak[i]) {
                    case 4720:
                        singleTransientExplanation.push_back(
                            "Component 472 Vent Valve disabled due to problems");
                        fout << "*-----------------------------------------------------" << endl;
                        fout << "*                      component 472" << endl;
                        fout << "*                      disabled due to problems" << endl;
                        //fout << "* PCS-106A" << endl;
                        //fout << "*crdno            name            type" << endl;
                        //fout << "472        PCS106A           valve" << endl;
                        //fout << "*crdno from to area floss rloss ctl" << endl;
                        //fout << "4720101 420020004 421010003 3.18-5 14.0 14.0 1100" << endl;
                        //fout << "*crdno     ctl   flowf   flowg    velj" << endl;
                        //fout << "4720201      1     0.0     0.0     0.0" << endl;
                        //fout << "*crdno    type" << endl;
                        //fout << "4720300 trpvlv  " << endl;
                        //fout << "*crdno trpno 403 is Normal Ops, 499 is blocked, 498 is open"
                        //  << endl;
                        //fout << "4720301    498" << endl;
                        break;
                    case 4820:
                        singleTransientExplanation.push_back("Component 482 Vent Valve OPEN");
                        fout << "*-----------------------------------------------------" << endl;
                        fout << "*                      component 482" << endl;
                        fout << "* PCS-106B" << endl;
                        fout << "*crdno            name            type" << endl;
                        fout << "482        PCS106A           valve" << endl;
                        fout << "*crdno from to area floss rloss ctl" << endl;
                        fout << "4820101 430020004 431010003 3.18-5 14.0 14.0 1100" << endl;
```

```
            fout << "*crdno     ctl   flowf   flowg    velj" << endl;
            fout << "4820201      1     0.0     0.0     0.0" << endl;
            fout << "*crdno    type      " << endl;
            fout << "4820300 trpvlv  " << endl;
            fout << "*crdno trpno 404 is Normal Ops, 499 is blocked, 498 is open"
                << endl;
            fout << "4820301    498" << endl;
            break;
        case 4721:
            singleTransientExplanation.push_back("Component 472 Vent Valve CLOSED");
            fout << "*-----------------------------------------------------" << endl;
            fout << "*                      component 472" << endl;
            fout << "* PCS-106A" << endl;
            fout << "*crdno            name            type" << endl;
            fout << "472        PCS106A           valve" << endl;
            fout << "*crdno from to    area floss rloss ctl" << endl;
            fout << "4720101 420020004 421010003  3.18-5 14.0 14.0 1100" << endl;
            fout << "*crdno     ctl   flowf   flowg    velj" << endl;
            fout << "4720201      1     0.0     0.0     0.0" << endl;
            fout << "*crdno    type      " << endl;
            fout << "4720300 trpvlv  " << endl;
            fout << "*crdno trpno 403 is Normal Ops, 499 is blocked, 498 is open"
                << endl;
            fout << "4720301    499" << endl;
            break;
        case 4821:
            singleTransientExplanation.push_back("Component 482 Vent Valve CLOSED");
            fout << "*-----------------------------------------------------" << endl;
            fout << "*                      component 482" << endl;
            fout << "* PCS-106B" << endl;
            fout << "*crdno            name            type" << endl;
            fout << "482        PCS106A           valve" << endl;
            fout << "*crdno from to area floss rloss ctl" << endl;
            fout << "4820101 430020004 431010003 3.18-5 14.0 14.0 1100" << endl;
            fout << "*crdno     ctl   flowf   flowg    velj" << endl;
            fout << "4820201      1     0.0     0.0     0.0" << endl;
            fout << "*crdno    type      " << endl;
            fout << "4820300 trpvlv  " << endl;
```

```
            fout << "*crdno trpno 404 is Normal Ops, 499 is blocked, 498 is open"
                << endl;
            fout << "4820301    499" << endl;
            break;
        case 90:
            singleTransientExplanation.push_back(
                "Component 009 Connection Pres & ADS Vent Line Steam Space CLOSED");
            fout << "*-----------------------------------------------------" << endl;
            fout << "*                      component 009" << endl;
            fout << "* Connection Pres & ADS Vent Line Steam Space" << endl;
            fout << "*crdno            name            type" << endl;
            fout << "009        PZRADSv           valve" << endl;
            fout << "*crdno from to    area floss rloss ctl" << endl;
            fout << "0090101 301080002 30200 0 0 0 0" << endl;
            fout << "*crdno     ctl  mflowf  mflowg    velj" << endl;
            fout << "0090201      1    0.00     0.0       0" << endl;
            fout << "*crdno     valve" << endl;
            fout << "0090300    trpvlv" << endl;
            fout << "*crdno     trip --- 498 is OPEN, 499 is CLOSED" << endl;
            fout << "0090301    499" << endl;
            break;
        case 7510:
            singleTransientExplanation.push_back(
                "Component 110 Hot Leg Leak Valve OPEN");
            fout << "*-----------------------------------------------------" << endl;
            fout << "*Component 751 --- Component 110 (Hot Leg) Leak Valve" << endl;
            fout << "*-----------------------------------------------------" << endl;
            fout << "*crdno            name            type" << endl;
            fout << "751         HLLkVl           valve" << endl;
            fout << "*cardno FROM TO area floss rloss ctl" << endl;
            fout << "7510101 110150002 201150001 5.0-5 0 0 0" << endl;
            fout << "*cardno    ctl         fvel        gvel    zero" << endl;
            fout << "7510201    0           0.0         0.0     0" << endl;
            fout << "*cardno    valve" << endl;
            fout << "7510300    trpvlv" << endl;
            fout << "*cardno trip 498 is OPEN, 499 is CLOSED" << endl;
            fout << "7510301    498" << endl;
            break;
```

```cpp
case 7520:
    singleTransientExplanation.push_back(
        "Component 201 Cold Leg Leak Valve OPEN");
    fout << "*-----------------------------------------------------" << endl;
    fout << "*Component 752 --- Component 201 (Cold Leg) Leak Valve" << endl;
    fout << "*-----------------------------------------------------" << endl;
    fout << "*crdno            name            type" << endl;
    fout << "752         CLLkVl           valve" << endl;
    fout << "*cardno FROM TO area floss rloss ctl" << endl;
    fout << "7520101 201230002 852010001 5.0-5 0 0 0" << endl;
    fout << "*cardno    ctl         fvel        gvel    zero" << endl;
    fout << "7520201    0           0.0         0.0     0" << endl;
    fout << "*cardno    valve" << endl;
    fout << "7520300    trpvlv" << endl;
    fout << "*cardno    trip  498 is OPEN, 499 is CLOSED" << endl;
    fout << "7520301    498" << endl;
    break;
case 7540:
    singleTransientExplanation.push_back(
        "Component 754 Containment to Cooling Pool Leak Valve OPEN");
    fout << "*-----------------------------------------------------" << endl;
    fout << "* Component 754 - Containment to Cooling Pool leak valve"<<endl;
    fout << "*-----------------------------------------------------" << endl;
    fout << "*cardno           name            type" << endl;
    fout << "754        ConCPLk           valve" << endl;
    fout << "*cardno FROM TO area floss rloss ctl" << endl;
    fout << "7540101 700010002 800010001 5.0-4 0 0 0" << endl;
    fout << "*cardno    ctl         fvel        gvel    zero" << endl;
    fout << "7540201    0           0.0         0.0     0" << endl;
    fout << "*cardno    valve" << endl;
    fout << "7540300    trpvlv" << endl;
    fout << "*cardno    trip  498 is OPEN, 499 is CLOSED" << endl;
    fout << "7540301    498" << endl;
    break;
case 0010:
    singleTransientExplanation.push_back(
        "Component 001 Flow Blockage Connection from Core to Hotleg");
    fout << "*-----------------------------------------------------" << endl;
```

```
                fout << "*                      component 001 Flow Blockage" << endl;
                fout << "*Connection from Core to Hotleg" << endl;
                fout << "*                        disabled due problems"  << endl;
                //fout << "*crdno            name            type" << endl;
                //fout << "001        coreout           valve" << endl;
                //fout << "*crdno from to area floss rloss ctl" << endl;
                //fout << "0010101 100060002 110010001 0 0 0 0" << endl;
                //fout << "*crdno     ctl   flowf   flowg    velj" << endl;
                //fout << "0010201      1    1.50     0.0       0" << endl;
                //fout << "*crdno     valve" << endl;
                //fout << "0010300    trpvlv" << endl;
                //fout << "*crdno     trip --- 498 is OPEN, 499 is CLOSED" << endl;
                //fout << "0010301    499" << endl;

                break;
            case 0020:
                singleTransientExplanation.push_back(
                    "Flow Blockage Connection from Hotleg to Upper Plenum");
                fout << "*-----------------------------------------------------" << endl;
                fout << "*                      component 002 Flow Blockage" << endl;
                fout << "*Connection from Hotleg to Upper Plenum" << endl;
                fout << "*                        disabled due problems"  << endl;
                //fout << "*crdno            name            type" << endl;
                //fout << "002          hotplen           valve" << endl;
                //fout << "*crdno from to area floss rloss ctl" << endl;
                //fout << "0020101 110290002 3  8.213-3 0 0 0" << endl;
                //fout << "*crdno     ctl   flowf   flowg    velj" << endl;
                //fout << "0020201      1    1.50     0.0       0" << endl;
                //fout << "*crdno     valve" << endl;
                //fout << "0020300    trpvlv" << endl;
                //fout << "*crdno     trip --- 498 is OPEN, 499 is CLOSED" << endl;
                //fout << "0020301    499" << endl;
                break;
            case 0030:
                singleTransientExplanation.push_back(
                    "Component 003 Flow Blockage Lower cold leg outlet");
                fout << "*-----------------------------------------------------" << endl;
                fout << "*                      component 003 Flow Blockage" << endl;
```

```
        fout << "*                         disabled due problems"  << endl;
        //fout << "* Lower cold leg outlet" << endl;
        //fout << "*crdno            name            type" << endl;
        //fout << "003         lclout           valve" << endl;
        //fout << "*crdno from to area floss rloss ctl" << endl;
        //fout << "0030101 201350002 20200 0 0 0 0" << endl;
        //fout << "*crdno     ctl   flowf   flowg    velj" << endl;
        //fout << "0030201      1    1.50     0.0       0" << endl;
        //fout << "*crdno     valve" << endl;
        //fout << "0030300    trpvlv" << endl;
        //fout << "*crdno     trip --- 498 is OPEN, 499 is CLOSED" << endl;
        //fout << "0030301    499" << endl;
        break;
    case 0040:
        singleTransientExplanation.push_back(
            "Flow Blockage Connection Plenum into the Core CLOSED");
        fout << "*-----------------------------------------------------" << endl;
        fout << "*                      component 004 Flow Blockage" << endl;
        fout << "*                         disabled due problems"  << endl;
        //fout << "*crdno            name            type" << endl;
        //fout << "004         corein           valve" << endl;
        //fout << "*crdno from to area floss rloss ctl" << endl;
        //fout << "0040101 20200 1 0 0 0 100" << endl;
        //fout << "*crdno     ctl   flowf   flowg    velj" << endl;
        //fout << "0040201      1    1.50     0.0       0" << endl;
        //fout << "*crdno     valve" << endl;
        //fout << "0040300    trpvlv" << endl;
        //fout << "*crdno     trip --- 498 is OPEN, 499 is CLOSED" << endl;
        //fout << "0040301    499" << endl;
        break;
    case 0011:
        singleTransientExplanation.push_back(
            "ParitialFlowBlockage Connection from Core to Hotleg CLOSED");
        fout << "*-----------------------------------------------------" << endl;
        fout << "*                   component 001 ParitialFlowBlockage" << endl;
        fout << "*Connection from Core to Hotleg" << endl;
        fout << "*crdno            name            type" << endl;
        fout << "001        coreout           valve" << endl;
```

```
            fout << "*crdno from to area floss rloss ctl" << endl;
            fout << "0010101 100060002 110010001 1.5-2 0 0 0" << endl;
            fout << "*crdno     ctl   flowf   flowg    velj" << endl;
            fout << "0010201      1    1.50     0.0       0" << endl;
            fout << "*crdno     valve" << endl;
            fout << "0010300    trpvlv" << endl;
            fout << "*crdno     trip --- 498 is OPEN, 499 is CLOSED" << endl;
            fout << "0010301    499" << endl;
            break;
        case 0021:
            singleTransientExplanation.push_back(
                "ParitialFlowBlockage Connection Hotleg to Upper Plenum CLOSED");
            fout << "*-----------------------------------------------------" << endl;
            fout << "*                   component 002 ParitialFlowBlockage" << endl;
            fout << "*Connection from Hotleg to Upper Plenum" << endl;
            fout << "*crdno            name            type" << endl;
            fout << "002          hotplen           valve" << endl;
            fout << "*crdno from to area floss rloss ctl" << endl;
            fout << "0020101 110290002 3 4.0-3 0 0 0" << endl;
            fout << "*crdno     ctl   flowf   flowg    velj" << endl;
            fout << "0020201      1    1.50     0.0       0" << endl;
            fout << "*crdno     valve" << endl;
            fout << "0020300    trpvlv" << endl;
            fout << "*crdno     trip --- 498 is OPEN, 499 is CLOSED" << endl;
            fout << "0020301    499" << endl;
            break;
        case 0031:
            //commented out due to errors
            singleTransientExplanation.push_back(
                "Component 003 ParitialFlowBlockage Lower Cold Leg Outlet CLOSED");
            //fout << "*-----------------------------------------------------"<<endl;
            //fout << "*                   component 003 ParitialFlowBlockage"<<endl;
            //fout << "* Lower cold leg outlet" << endl;
            //fout << "*crdno            name            type" << endl;
            //fout << "003         lclout           valve" << endl;
            //fout << "*crdno from to area floss rloss ctl" << endl;
            //fout << "0030101 201350002 20200   1.7-2 0 0 0" << endl;
            //fout << "*crdno     ctl   flowf   flowg    velj" << endl;
```

```
            //fout << "0030201      1    1.50     0.0       0" << endl;
            //fout << "*crdno     valve" << endl;
            //fout << "0030300    trpvlv" << endl;
            //fout << "*crdno     trip --- 498 is OPEN, 499 is CLOSED" << endl;
            //fout << "0030301    499" << endl;
            break;
        case 0041:
            singleTransientExplanation.push_back(
                "Component 004 ParitialFlowBlockage Core Inlet CLOSED");
            fout << "*-----------------------------------------------------" << endl;
            fout << "*                   component 004 ParitialFlowBlockage" << endl;
            fout << "*crdno            name            type" << endl;
            fout << "004         corein           valve" << endl;
            fout << "*crdno from to area floss rloss ctl" << endl;
            fout << "0040101 20200 1 4.2-3 0 0 100" << endl;
            fout << "*crdno     ctl   flowf   flowg    velj" << endl;
            fout << "0040201      1    1.50     0.0       0" << endl;
            fout << "*crdno     valve" << endl;
            fout << "0040300    trpvlv" << endl;
            fout << "*crdno     trip --- 498 is OPEN, 499 is CLOSED" << endl;
            fout << "0040301    499" << endl;
            break;
        case 4520:
            singleTransientExplanation.push_back(
                "Component 452 Sump Water Makeup CLOSED");
            fout << "*-----------------------------------------------------" << endl;
            fout << "*                 component 452 Sump Water Makeup Fail" << endl;
            fout << "* PCS 108-A" << endl;
            fout << "*crdno            name            type" << endl;
            fout << "452        PCS108A           valve" << endl;
            fout << "*crdno from to area floss rloss ctl" << endl;
            fout << "4520101 401020004 402010003 3.18-5 14.0 14.0 1100" << endl;
            fout << "*crdno     ctl   flowf   flowg    velj" << endl;
            fout << "4520201      1     0.0     0.0     0.0" << endl;
            fout << "*crdno    type      " << endl;
            fout << "4520300 trpvlv  " << endl;
            fout << "*crdno   trpno     401 is Normal Ops, 499 is blocked " << endl;
            fout << "4520301    499" << endl;
```

```
                        break;
                    case 4620:
                        singleTransientExplanation.push_back(
                            "Component 462 Sump Water Makeup CLOSED");
                        fout << "*-----------------------------------------------------" << endl;
                        fout << "*                 component 462 Sump Water Makeup Fail" << endl;
                        fout << "* PCS 108-B" << endl;
                        fout << "*crdno            name            type" << endl;
                        fout << "462        PCS108B           valve" << endl;
                        fout << "*crdno from to area floss rloss ctl" << endl;
                        fout << "4620101 410020004 411010003 3.18-5 14.0 14.0 1100" << endl;
                        fout << "*crdno     ctl   flowf   flowg    velj" << endl;
                        fout << "4620201      1     0.0     0.0     0.0" << endl;
                        fout << "*crdno    type     " << endl;
                        fout << "4620300 trpvlv  " << endl;
                        fout << "*crdno   trpno     402 is Normal Ops, 499 is blocked" << endl;
                        fout << "4620301    499" << endl;
                        break;
                    case 4521:
                        singleTransientExplanation.push_back(
                            "Component 452 disabled due to problems");
                        fout << "*-----------------------------------------------------" << endl;
                        fout << "*                      component 452 SumpOpen" << endl;
                        fout << "*                      disabled due to problems" << endl;

                        //fout << "* PCS 108-A" << endl;
                        //fout << "*crdno            name            type" << endl;
                        //fout << "452        PCS108A           valve" << endl;
                        //fout << "*crdno from to area floss rloss ctl" << endl;
                        //fout << "4520101 401020004 402010003 3.18-5 14.0 14.0 1100" << endl;
                        //fout << "*crdno     ctl   flowf   flowg    velj" << endl;
                        //fout << "4520201      1     0.0     0.0     0.0" << endl;
                        //fout << "*crdno    type     " << endl;
                        //fout << "4520300 trpvlv  " << endl;
                        //fout << "*crdno   trpno 401 is Normal Ops, 499 is blocked " << endl;
                        //fout << "4520301    498" << endl;
                        break;
                    case 4621:
```

```
                singleTransientExplanation.push_back(
                    "Component 462 disabled due to problems");
                fout << "*------------------------------------------------------" << endl;
                fout << "*                      component 462 SumpOpen" << endl;
                fout << "*                      disabled due to problems" << endl;
                //fout << "* PCS 108-B" << endl;
                //fout << "*crdno            name            type" << endl;
                //fout << "462        PCS108B           valve" << endl;
                //fout << "*crdno from to area floss rloss ctl" << endl;
                //fout << "4620101 410020004 411010003 3.18-5 14.0 14.0 1100" << endl;
                //fout << "*crdno     ctl   flowf   flowg    velj" << endl;
                //fout << "4620201      1     0.0     0.0     0.0" << endl;
                //fout << "*crdno    type     " << endl;
                //fout << "4620300 trpvlv  " << endl;
                //fout << "*crdno   trpno     402 is Normal Ops, 499 is blocked" << endl;
                //fout << "4620301    498" << endl;
                break;
            case 0570:
                singleTransientExplanation.push_back(
                    "Component 057 Junction with SG inlet CLOSED");
                fout << "*------------------------------------------------------" << endl;
                fout << "*                 component 057 Secondary Loop Failure" << endl;
                fout << "* Junction with SG inlet" << endl;
                fout << "*crdno            name            type" << endl;
                fout << "057           SGin           valve" << endl;
                fout << "*crdno from to area floss rloss ctl" << endl;
                fout << "0570101 501040002 505010001 0 0 0 0" << endl;
                fout << "*crdno     ctl   flowf   flowg    velj" << endl;
                fout << "0570201      1  4.50-1     0.0     0.0" << endl;
                fout << "*crdno     valve" << endl;
                fout << "0570300    trpvlv" << endl;
                fout << "*crdno     trip --- 498 is OPEN, 499 is CLOSED" << endl;
                fout << "0570301    499" << endl;
                break;
            case 580:
                singleTransientExplanation.push_back(
                    "Component 058 Junction with SG Outlet to Main Steam Line CLOSED");
                fout << "*------------------------------------------------------" << endl;
```

```
                       fout << "*                      component 058" << endl;
                       fout << "* Junction with SG Outlet to Main Steam Line" << endl;
                       fout << "*crdno            name            type" << endl;
                       fout << "058          SGout           valve" << endl;
                       fout << "*crdno from to area floss rloss ctl" << endl;
                       fout << "0580101 505110004 507010001 0 0 0 0" << endl;
                       fout << "*crdno     ctl   flowf   flowg    velj" << endl;
                       fout << "0580201      1     0.0  4.50-1     0.0" << endl;
                       fout << "*crdno     valve" << endl;
                       fout << "0580300    trpvlv" << endl;
                       fout << "*crdno     trip --- 498 is OPEN, 499 is CLOSED" << endl;
                       fout << "0580301    499" << endl;
                       break;
                   default:
                       singleTransientExplanation.push_back("No transients run");
                       fout <<"*                            No transients run" << endl;
                       }
                   }
               }
   //end card (don't comment out on accident!)
   fout <<". end of data" << endl;
   return singleTransientExplanation;
}

//This functions writes the user display in html
void htmlDisplayWriter(string OutDir, string InDir, double RstNum,
   vector <vector <int> > EndBySumVec, vector <int> keepGoing,
   vector<vector<int> > clustMembers, vector<vector<string > > transientExplanation) {
   string displayOutFilePath;
   string displayOutFile;
   string greenOutFile;
   string greenOutFilePath;
   string unstableOutFile;
   string unstableOutFilePath;
   string miscOutFile;
   string miscOutFilePath;
   int redTableCols=0;
   int yellowTableCols=0;
```

```
for (int i=0; i<EndBySumVec[2].size(); i++) {
    if (EndBySumVec[2][i]==1) {
        redTableCols++;
    }
}
if (redTableCols==0) {
    redTableCols=1;
}

for (int i=0; i<keepGoing.size(); i++) {
    if (keepGoing[i]==1) {
        yellowTableCols++;
    }
}
if (yellowTableCols==0) {
    yellowTableCols=1;
}

stringstream sstm;
sstm << "DISPLAY" << RstNum << ".html"; //adding index to the string
displayOutFile = sstm.str();
sstm.str("");
displayOutFilePath = (OutDir + "/" + displayOutFile);

sstm << "green" << RstNum << ".html";   //adding index to the string
greenOutFile = sstm.str();
sstm.str("");
greenOutFilePath = (OutDir + "/" + greenOutFile);

sstm << "unstable" << RstNum << ".html";    //adding index to the string
unstableOutFile = sstm.str();
sstm.str("");
unstableOutFilePath = (OutDir + "/" + unstableOutFile);

sstm << "misc" << RstNum << ".html";    //adding index to the string
miscOutFile = sstm.str();
sstm.str("");
```

```
miscOutFilePath = (OutDir + "/" + miscOutFile);


bool output=false;
ofstream fout(displayOutFilePath.c_str());
//cout << displayOutFilePath << endl;
fout << "<!--Display Engine For RAPSS-STA - Written by Kevin Makinson-->" << endl;
fout << "<html><head>" << endl;
fout << "<meta content=\"text/html; charset=ISO-8859-1\" http-equiv=\"content-type\">";
fout << "<title>RAPSS-STA Display</title>" << endl;
fout << "<link rel=\"stylesheet\" type=\"text/css\" href=\"tswtabs.css\">" << endl;
fout << "</head>" << endl;
fout << "<body>" << endl;
fout << "<strong><font size=\"+2\">RAPSS-STA Output Restart "
    << RstNum << "</font><br>" << endl;
fout << "<br>" << endl;
fout << "<span style=\"text-decoration: underline;\">";
fout << "Red Thresholds Tripped</span></strong><p><br>" << endl;
fout << "<table style=\"border-color: rgb(255, 0, 0); text-align: left; width: "
<< 500*redTableCols <<"px;\" border=\"10\" cellpadding=\"2\" cellspacing=\"2\">"<< endl;
fout << "<tbody>"<< endl;
fout << "<tr>"<< endl;
for (unsigned int i=0; i<EndBySumVec[2].size(); i++) {
    if (EndBySumVec[2][i]==1) {
        //red threshold logic
        //scenario i has red threshold reached
        output=true;
        fout << "<td>" << endl;
        fout << "<ul style=\"color: red;\">" << endl;
        fout << "<img style=\"width: 164px; height: 41px;\" alt=\"\" src=\"Alert.gif\">";
        fout << "<br><ul style=\"color: red;\">" << endl;
        fout << "<li><a href=\"alerts" << RstNum <<".pdf\">Secnario " << i
            << " Plots</a></li>" << endl;
        fout << "<ul>" << endl;
        fout << "<li><a href=\"Th_" << i <<"_data/outputs/rst"<< RstNum
            <<".csv\">Data</a></li>" << endl;
        fout << "</ul>" << endl;
        fout << "<ul style=\"color: red;\">" << endl;
```

```
        for (unsigned int k=0; k<transientExplanation[i].size(); k++) {
            fout << "<li>" << transientExplanation[i][k]  << " </li>" << endl;
        }
        //fout << "<li>Other information</li>" << endl;//transient information goes here
        fout << "</ul>" << endl;
        fout << "</ul>" << endl;
        fout << "</td>" << endl;
    } else if (i==(EndBySumVec[2].size()-1) && (output==false)) {
        //fout << "<ul style=\"color: red;\">" << endl;
        fout << "<td>" << endl;
        fout << "<li> No red trips </li>" << endl;
        fout << "</td>" << endl;
        fout << "</u1>" << endl;
    }
}
fout << "</td>" << endl;
fout << "</tr>" << endl;
fout << "</tbody>" << endl;
fout << "</table>" << endl;
fout << "</li></li></ul></ul><p>" << endl;
output=false;

fout << "<strong><span style=\"text-decoration: underline;\">";
fout << "Yellow Thesholds Tripped</span></strong><p><br>" << endl;
//change this one to be similar to the one above
fout << "<table style=\"border-color: rgb(255, 180, 0); text-align: left; width: "
    << 500*yellowTableCols <<"px;\" border=\"10\" cellpadding=\"2\" cellspacing=\"2\">"
    << endl;
fout << "<tbody>" << endl;
fout << "<tr>" << endl;
for (unsigned int i=0; i<keepGoing.size(); i++) {
    if (keepGoing[i]==1) {
        //yellow threshold logic
        //scenario i has yellow threshold reached
        output=true;
        //fout << "<td style=\"width: 500px;\">" << endl; //just added
        fout << "<td>" << endl;
        fout << "<ul style=\"color: rgb(225, 180, 0);\">" << endl;
```

```
            fout << "<li>Scenario " << i << "</li>" << endl;
            fout << "<ul>" << endl;
            fout << "<li><a href=\"Th_" << i <<"_data/outputs/rst" << RstNum
                << ".csv\">Data</a></li>" << endl;
            fout << "</ul>" << endl;
            fout << "<ul>" << endl;
            for (unsigned int k=0; k<transientExplanation[i].size(); k++) {
                fout << "<li>" << transientExplanation[i][k]  << " </li>" << endl;
            }
            fout << "</ul>" << endl;
            fout << "</td>" << endl;
            } else if (i==(keepGoing.size()-1) && (output==false)) {
                fout << "<td>" << endl;
                fout << "<li> No yellow trips </li>" << endl;
                fout << "</td>" << endl;
                fout << "</u1>" << endl;
        }
    }
    fout << "</td>" << endl;
    fout << "</tr>" << endl;
    fout << "</tbody>" << endl;
    fout << "</table>" << endl;
    fout << "</li></li></ul></ul>" << endl;
    output=false;

    fout << "<p><br>" << endl;
    fout << "</span><br><div id=\"tswcsstabs\">" << endl;
    fout << "<ul>" << endl;
    fout << "<li><a href=\"green" << RstNum
        << ".html\">No Thresholds Tripped</a></li>" << endl;
    fout << "<li><a href=\"unstable" << RstNum
        << ".html\">R5 Model Became Unstable</a></li>" << endl;
    fout << "<li><a href=\"misc" << RstNum
        << ".html\">Miscellaneous Information</a></li>" << endl;
    fout << "</ul>" << endl;
    fout << "</div>" << endl;
    fout << "</body></html>" << endl;
    fout.close();
```

```
    fout.clear();
    //this ends the main page.

    fout.open(greenOutFilePath.c_str()); //double check this does what I want it to.
    fout << "<!--Display Engine For RAPSS-STA - Written by Kevin Makinson-->" << endl;
    fout << "<html><head>" << endl;
    fout << "<meta content=\"text/html; charset=ISO-8859-1\" http-equiv=\"content-type\">";
    fout << "<title>RAPSS-STA Green Thresholds</title>" << endl;
    fout << "</head>" << endl;
    fout << "<body>" << endl;
    fout << "<strong><font size=\"+2\">RAPSS-STA Output Restart "
        << RstNum << "</font><br>" << endl;
    fout << "<br>" << endl;
    fout << "<span style=\"text-decoration: underline;\">No Thesholds Tripped";
    fout <<"</span></strong><br>" << endl;
    fout << "<ul style=\"color: rgb(0, 153, 0);\">" << endl;
    for (unsigned int i=0; i<EndBySumVec[1].size(); i++) {
        if (EndBySumVec[1][i]==1) {
            //green threshold logic
            //scenario i has no threshold reached
            fout << "<li>Secnario " << i << "</li>" << endl;
            fout << "<ul>" << endl;
            fout << "<li><a href=\"Th_" << i <<"_data/outputs/rst"
                << RstNum << ".csv\">Data</a></li>" << endl;
            fout << "</ul>" << endl;
            fout << "<ul>" << endl;
            for (unsigned int k=0; k<transientExplanation[i].size(); k++) {
                fout << "<li>" << transientExplanation[i][k]  << " </li>" << endl;
            }
            fout << "</ul>" << endl;
        }
    }
    fout << "</body></html>" << endl;
    fout.close();
    fout.clear();
//--
    fout.open(miscOutFilePath.c_str());
    fout << "<html><head>" << endl;
```

```
    fout << "<meta content=\"text/html; charset=ISO-8859-1\" http-equiv=\"content-type\">";
    fout << "<title>RAPSS-STA Cluster Information</title>" << endl;
    fout << "</head>" << endl;
    fout << "<body>" << endl;
    fout << "<strong><font size=\"+2\">RAPSS-STA Output Restart "
        << RstNum << "</font><br>" << endl;
    fout << "<br>" << endl;
    fout << "<span style=\"text-decoration: underline;\"></span>";
    fout << "<span style=\"text-decoration: underline;\">Cluster Information";
    fout << "</strong><br>" << endl;
    fout << "</span>" << endl;
    fout << "<ul>" << endl;
    for (unsigned int j=0; j<clustMembers.size(); j++) { //j is cluster number
        fout << "<li><a href=\"clusterPlots" << RstNum
            <<".pdf\">Cluster " << j+1 << " Plot</a></li>" << endl;
        fout << "<ul>" << endl;
        fout << "<li><a href=\"unMSAPCAc" << j+1 << "rst" << RstNum
            << ".csv\">Cluster Data</a></li>" << endl;
        fout << "<li>Cluster Members </li>" << endl;
        fout << "<ul>" << endl;
        for (unsigned int i=0; i<clustMembers[j].size(); i++) { //i is scenario number
            fout << "<li><a href=\"Th_" << clustMembers[j][i] << "_data/outputs/rst" << RstNum
                << ".csv\">Scenario " << clustMembers[j][i] << "</a></li>" << endl;
            fout << "<ul>" << endl;
            fout << "</ul>" << endl;
        }

        fout << "</li>" << endl;
        fout << "</ul>" << endl;
        fout << "</ul>" << endl;
    }

    fout << "</body></html>" << endl;
    fout.clear();
    fout.close();
//--
    fout.open(unstableOutFilePath.c_str());
    fout << "<!--Display Engine For RAPSS-STA - Written by Kevin Makinson-->" << endl;
```

```
    fout << "<html><head>" << endl;
    fout << "<meta content=\"text/html; charset=ISO-8859-1\" http-equiv=\"content-type\">";
    fout << "<title>RAPSS-STA Unstable Scenarios</title>" << endl;
    fout << "</head>" << endl;
    fout << "<body>" << endl;
    fout << "<strong><font size=\"+2\">RAPSS-STA Output Restart " << RstNum
        << "</font><br>" << endl;
    fout << "<p><span style=\"text-decoration: underline;\">R5 Model Became Unstable";
    fout << "</span></strong><br>" << endl;
    for (unsigned int i=0; i<EndBySumVec[0].size(); i++) {
        if (EndBySumVec[0][i]==1) {
            output=true;
            fout << "<ul>" << endl;
            fout << "<li><a href=\"Th_" << i << "_data/outputs/rst" << RstNum << ".p\">Scenario "
                << i << "</a></li>" << endl;
            fout << "<ul>" << endl;
            for (unsigned int k=0; k<transientExplanation[i].size(); k++) {
                fout << "<li>" << transientExplanation[i][k]  << " </li>" << endl;
            }
            fout << "</ul>" << endl;
            fout << "</ul>" << endl;
        }
        if ((i==EndBySumVec[0].size()-1) && (output==false)) { //new
            fout << "<li> No model instabilities on this cycle </li>" << endl;
        }
    }
    fout << "</body></html>" << endl;
    fout.close();
    fout.clear();
}
#endif
```

A.4. OrganizeR5Output.h Source Code

```cpp
//Created by Kevin Makinson
//3/20/12
//This is a header file that organizes the R5 output

#ifndef OrganizeR5Output_h
#define OrganizeR5Output_h
#include "BloodAndGuts.h"

vector < vector<string> > OrganizeR5Output(string R5OutputFilePath, string CsvFilePath,
    int th_id) {
    vector<string> text =  LoadFile(R5OutputFilePath);
    vector<string> row;
    vector<int> data1Sections;
    vector< vector<string> > data1;
    //vector< vector<string> > FormatData;
    int StrtIndx = 1;
    int k;
    int FormatSectionLength;
    int FormatDataWidth;
    int HeaderLength=0;
    int SectionStart;
    int jIndex;
    int jMult;
    int CountTime=0;    //how many times "time" appears.
    long int StartMult=-1;
    bool done=false;
    int FullSections;
    int SetsPerSection=0;
    string word;            //need to use an extra string "word" b/c no push_back += function
    stringstream sstm;

    if (SearchVec(text, "1 time  ").back()==0) {
        cerr << "No minor edit data to be read on thread " << th_id << "!" << endl
            << "Skipping thread... " << endl;
    } else {
```

```
//---------
//grabs the data from the R5 output file and organizes it exactly how it is organized
//in the R5 output, which is not always desirable
for (unsigned int h=0; h<SearchVec(text, "1 time  ").size(); h++) {
    //h is the number of tables to be grabbed from R5
    //j is the the col of the new matrix being created
    //k[h] is the line(row) number in the string in the R5 output file
    k=SearchVec(text, "1 time  ")[h];
    while (CountTime!=2 && done!=true) {
        for (unsigned int j=0; j<text[k].length()/13; j++) {
            //last whole number after division by 13
            for (unsigned int i=StrtIndx+(j*13); i<(StrtIndx+(j+1)*13); i++) {
                //i is the character number line k (overloaded subscript index)
                //take chunks of the string 13 characters at a time
                word += text[k][i];
            }
            word = TrimSpace(word); //trims excess white space around 13 character blocks
            if (word == "time") {
                CountTime++;
            }
            //else if (word == "RELAP5/3.2" ) { //change with different versions of RELAP
            else if ((word == "RELAP5/3.3g") || (word=="steady state")
                || (word=="******** Tran") || (word== "******** temp")
                || (word== "******** Trou")  || (word=="ATHENA-3D Ver")
                || (word=="Number of ele")) {
            CountTime++;        //breaks from while loop
                word = "";          //clear the string "word"
                break;              //breaks from j for loop.
            }
            else if ((word == "Final time=") || (word=="---Restart Su")
                || (word=="******** Tran") || (word== "******** temp")
                || (word== "******** Trou")) {
                done=true;          //breaks from while loop
                break;              //breaks from j for loop.
            }
            row.push_back(word);    //put word into row
            word = "";              //clear the string "word"
        }
```

```
                data1.push_back(row);       //put row into matrix
                row.clear();                //clear row for next iteration
                k++;                        //k is part of "while loop"
            }
            CountTime=0;
            data1.pop_back();   //Deletes the last row int the matrix; this is because the last
            //row is always written before the test is performed to determine stop time
        }
        //---------------------------
        //specialized variables for data organization
        //this guy is tells you where the new sections are in the data.
        for (unsigned int i=0; i<data1.size()-1; i++) {
            if (data1[i][0] == "(sec)") {
                data1Sections.push_back(i-1);
            }
        }
        if (data1Sections.size()==1) {SetsPerSection=1;}

            for (unsigned int i=1; i<SearchVec(text, "1 time  ").size(); i++) {
                //if the data has less than 54 data points (1 section):
                if ((SearchVec(text, "1 time  ")[i]-SearchVec(text, "1 time  ")[i-1]) < 54) {
                    //54 is the max size of a section
                    SetsPerSection = SearchVec(text, "1 time     ").size();
                    break;
                }
                //for data with more than 1 section:
                //Logic: if the same index of the next section isn't the same time
                //it's a new section
                else if (data1[(i-1)*50+(4*i)][0] != data1[(i)*50+(4*(i+1))][0]) {
                    SetsPerSection = i;
                    break;
                }
            }

        data1Sections.push_back(data1.size());
        //adding a final element that is the last element that is the size
        FormatSectionLength = data1Sections[1]-data1Sections[0];
        FullSections = ((data1Sections.size()-1)/SetsPerSection)-1;
```

```
        //The Length of the first section times the number of sections, plus 1
        //plus the length of the final section (which is a different
        //length than the other others)
        int Length = FormatSectionLength*FullSections+1+
            (data1Sections.back()-data1Sections[data1Sections.size()-2]);
        //this allows extra space at the end just in case
        int Width = data1[0].size()*SetsPerSection;
        vector< vector<string> > FormatData(Length, vector<string> (Width));
        //-------------------------------------------------------------------------
        //Now we organize the data.
        for (unsigned int k=0; k<data1Sections.size()-1; k++) {
            if (k==SetsPerSection) {HeaderLength=4;}
            //this makes it so only the first section has headers
                if (k%SetsPerSection==0) {
                    jIndex=0;
                    jMult=0;
                    StartMult++;
                }
                else {
                    jIndex=1;
                    jMult=k%SetsPerSection;
                }

            SectionStart = (FormatSectionLength-HeaderLength)*StartMult;
            FormatDataWidth = data1[data1Sections[k]].size();
            for (int i=data1Sections[k]+HeaderLength; i<data1Sections[k+1]; i++) {
                for (int j=jIndex; j<FormatDataWidth; j++) {
                    if (i==data1Sections[data1Sections.size()-1]-1) {break;} //for RELAP3.3 only
                    FormatData[SectionStart+(i-data1Sections[k])][j+
                        (data1[0].size()-1)*jMult] = data1[i][j];
                }
            }
        }

        //output to a csv file
        ofstream fout(CsvFilePath.c_str());
        for (int i=0; i<Length; i++) {
            for (int j=0; j<Width; j++) {
```

```
                fout << setw(13) << FormatData[i][j] << ",";
            }
            fout << endl;
        }
        return FormatData;
    }

}
#endif
```

A.5. initPCA.r Source Code

```
#!/usr/bin/Rscript
#   Mar  8 2013
#   Written by Kevin Makinson
#   This file loads the libraries and initial parameters in R
#
#
#----------------------------------------------------------------------
rm(list=ls())
Rrepos<-"http://cran.r-project.org"
libloc<-"/nfs/stak/students/m/makinske/lib"
threshold<-0.95
IODir<-"/nfs/chadwick/u1/makinske/R5run/RAPS_data"
libloc<-"/nfs/stak/students/m/makinske/lib"
dataOut<-1
thresholds<-rbind(650,1000)
thresholds<-cbind(thresholds, rbind(3.5e+06,3e+06))
thresholds<-cbind(thresholds, rbind(1e+07,9e+07))
stateVarTripNames <- c("httemp", "p", "p")
equivalence <- c("gt", "lt", "gt")
stateVarCodes <- c("133000101", "100010000", "500010000")
save.image("R_data/RAPSspace.RData")
```

A.6. PCA.r Source Code:

```r
#!/usr/bin/Rscript
# 7/23/12
# Written by Kevin Makinson
# Oregon State university
#
# This code takes the output from RAPS and performs PCA on it,
# outputting the file PC.csv for MSA to use
#
# --------------------------------
load("R_data/RAPSspace.RData")
library(corpcor, lib.loc=libloc) #for psuedoinverse
library(abind, lib.loc=libloc) #for 3-d matricies
library(MASS, lib.loc=libloc) #for ginverse

R5OutFilePaths<-array(0,thNum)
#assigning file paths
#reading data
#plopping it into a 3D matrix

#need this!
kCount=0
for (i in IncludeTh) {
  kCount=kCount+1
  R5OutFilePaths[kCount]<-paste(IODir,"/Th_", i, "_data/outputs/rst", rstNum, ".csv", sep = "")
  R5OutRawData<- read.csv(R5OutFilePaths[kCount], header=TRUE) #change from 1
  if(kCount==1) {
    R5OutRawDataC<-R5OutRawData
  } else if ((dim(R5OutRawDataC)[1])==(dim(R5OutRawData)[1])) {
    R5OutRawDataC<-abind(R5OutRawDataC, R5OutRawData, along=3)
  } else if ((dim(R5OutRawDataC)[1]) > (dim(R5OutRawData)[1])) {
    R5OutRawDataC<-abind(R5OutRawDataC[1:(dim(R5OutRawData)[1]),,], R5OutRawData, along=3)
  } else {
    R5OutRawDataC<-abind(R5OutRawDataC, R5OutRawData[1:(dim(R5OutRawDataC)[1]),], along=3)
  }
}
```

```
thNum<-dim(R5OutRawDataC)[3]

#these two loop checks to see if there are any cols or rows that have "NA" as members.
naCols<-0
for (i in 1:dim(R5OutRawDataC[,,1])[2]) {
  if (is.na(R5OutRawDataC[1,i,1])==TRUE) {
    naCols<-naCols+1
  }
}
naRows<-0
for (i in 4:dim(R5OutRawDataC[,,1])[1]) {
  if(R5OutRawDataC[i,1,1]==("             ")) {
    naRows<-naRows+1
  }
}

#making syntax more readable
dim1<-dim(R5OutRawDataC[,,1])[1]-naRows
dim2<-dim(R5OutRawDataC[,,1])[2]-naCols
units<- R5OutRawDataC[1:3,1:dim2,1]

TrimData<-array(0,c((dim1-3), dim2, thNum), dimnames=dimnames(R5OutRawDataC[4:dim1, 1:dim2,]))

#redefining cols as numeric instead of characters
for (j in 1:thNum) {
  for (i in 1:dim2) {
    TrimData[1:(dim1-3),i,j]<-as.numeric(R5OutRawDataC[4:(dim1),i,j])
  }
}
dim1<-dim1-3 #added 7/25/12

#this is eliminating the first and last time steps because they're funky
#when normalizing
time<-TrimData[(2:(dim(TrimData)[1]-1)),1,1]

#this next section finds the items to remove
#it searches for cols that have difference in sd compared to the mean of 0.00001
for (i in 1:thNum) {
```

```
  delColTemp<-colnames(TrimData[,,i])[1]
  for (j in 1:dim2) {
    if ((mean(TrimData[3:(dim(TrimData)[1]-2),j,i]))==0) {
      delColTemp<-rbind(delColTemp, colnames(TrimData[,,i])[j])
    } else if (0.00001>abs(1-abs(mean(TrimData[3:(dim(TrimData)[1]-2),j,i])
        -sd(TrimData[3:(dim(TrimData)[1]-2),j,i]))/
        abs(mean(TrimData[3:(dim(TrimData)[1]-2),j,i])))) {
      delColTemp<-rbind(delColTemp, colnames(TrimData[,,i])[j])
    }
    }
  if (i==1) {
    delcol<-delColTemp
  } else {
    delcol<-cbind(delColTemp, delcol)
  }
  }

#this loop checks to see if there are any state variables that might need to be
#taken out in one scenario, but not in others!
k<-0 #leave this k here, important
for (i in 1:dim(delcol)[1]) {
  for (j in 1:thNum) {
    if(delcol[i,1]!=delcol[i,j]) {
      k<-k+1
      if (k==1) {
        deldelcol<-delcol[i,j]
      } else if (delcol[i,j]!=tail(deldelcol, n=1)) {
        deldelcol<- rbind(deldelcol, delcol[i,j])
      }
    }
  }
}

#this reduces delcol down to a 1D variable
#delcol will never be zero because it will always have time in it.
if (k!=0) {
  delcol<-delcol[-which(delcol[,1] %in% deldelcol),1]
}
```

```
for (i in 1:thNum) {
  if (i==1) {
    TrimDataTemp2<-TrimData[, -which(colnames(TrimData[,,i]) %in% delcol),i]
  } else {
    TrimDataTemp2<-abind(TrimDataTemp2,TrimData[, -which(colnames(TrimData[,,i])
        %in% delcol),i], along=3)
  }
}
TrimData<-TrimDataTemp2
rm(TrimDataTemp2)

time1<-units[,1] #added to get a better label for time.
units<-cbind(time1, units[, -which(colnames(units[,]) %in% delcol)])

#normalizing data
#dim 1 is the time steps, dim 2 is the state variables
dim1<-length(TrimData[,1,1])-2
dim2<-length(TrimData[1,,1])
normalized<-array(0, c((dim1-1), dim2, thNum)) #occationally there's an NA at the end


for (j in 1:thNum){
  for (i in 1:(dim2)) {
    normalized[,i,j] <- (TrimData[(2:(dim1)),i,j]-mean(TrimData[(2:(dim1)),i,j]))/
        sd(TrimData[(2:(dim1)),i,j])
  }
}
dim1<-(dim1-1)
#--
#reorganizing the data into 2D matrix
#--

for (i in 1:dim1) {
  for (j in 1:thNum){
    if (i==1 && j==1) {
      TwoDnormalized<-normalized[1,1:dim2,1]
    } else {
```

```
      TwoDnormalized<-rbind(TwoDnormalized,normalized[i,1:dim2,j])
    }
  }
}

#---
#Linear Approximation Intervals
#initial conditions:
done<-FALSE
forward<-FALSE #This tells if the data intervals have reached the beginning
backward<-FALSE
adder<-(dim1)%%2
interval1Real<- as.integer(dim1/2)+adder
interval2Real<- as.integer(dim1/2)  #took out -adder on this statement
interval1<- interval1Real*thNum
interval2<- interval2Real*thNum-1
start<- 1
end<-dim(TwoDnormalized)[1]
endReal<-dim1
startReal<- 0
adder<-0 #this is for uneven splits in intervals.
intervalArray<-0
countIntLoop<-0
int1Direction<-FALSE
moveOn<-FALSE
badInterval<-0

while (done==FALSE) {
  countIntLoop<-(countIntLoop+1)
  if (countIntLoop==100) { #to avoid infinite loops
    cat("An error has occured while determining the PCA linear approximation intervals")
    break
  }
  #check if it's gone as small as can be and isn't done yet.
  if ((interval1Real==2) && (interval2Real==2)) {
    cat("An interval did not converge below given threshold! Check badInterval for details.\n")
    moveOn<-TRUE
    badInterval<-rbind(badInterval, startReal, (startReal+interval1Real))
```

```
}
delTReg<- time[(startReal+interval1Real+interval2Real)] - time[(startReal+interval1Real)]
covMatrix1<-cov(TwoDnormalized[start:(start+interval1-1),]) #added -1
covMatrix2<-cov(TwoDnormalized[(start+interval1):(start-1+interval1+interval2),])
if ((countIntLoop==1) && (norm(covMatrix1, type="m")>norm(covMatrix2, type="m"))) {
  int1Direction<-TRUE  #checks to see which direction we initially go
}

if (((norm(((covMatrix2)-(covMatrix1))/(delTReg), type="m") < 0.25)
     && (countIntLoop!=1)) || (moveOn==TRUE)) {
  moveOn<-FALSE
  #check if we've reached the beginning or end
  if ((start==1) && (backward!=TRUE)) {
    forward<-TRUE
    intervalArray<-startReal+interval1Real
  } else if (((start-1+interval1+interval2) == end) && (forward!=TRUE) &&
      (end==dim(TwoDnormalized)[1])) {
    backward<-TRUE
    intervalArray<-(endReal-interval2Real)
  } else if (forward==TRUE) {
    intervalArray<-rbind(intervalArray, (startReal+interval1Real))
  } else {
    intervalArray<-rbind(intervalArray, (endReal-interval2Real))
  }
  #check if we're done
  if ((((start+interval1+interval2) == end) && (forward==TRUE)) ||
    (start==1) && (backward==TRUE)) { #took out -1 7/14/12
    done<-TRUE
    break
    #if we're not done and below threshold look at next interval
  } else if (forward==TRUE) {
    #if not at the end, and the covs look good, and we're going forward advance forward
    start<-start+interval1
    startReal<-startReal+interval1Real
    interval1<-interval2
    interval1Real<- interval2Real
    interval2<- end-start-interval1
    interval2Real<-dim1-startReal-interval1Real
```

```r
    } else if (backward==TRUE) {
      #changing end point and start point going backwards
      interval1<-(end-interval1-interval2)
      interval1Real<- (endReal-interval1Real-interval2Real)
      end<-(end-interval2)
      endReal<-endReal-interval2Real
      interval2<-(end-interval1)
      interval2Real<-(endReal-interval1Real)
      start<- (end-interval1-interval2+1) #+1 is built into others because you add start into it
      startReal<- (endReal-interval1Real-interval2Real)
    }
    #if above threshold:
  } else if ((int1Direction==TRUE) && (forward==FALSE) && (backward==FALSE)) {
    #the initial march towards the beginning.
    adder<-(interval1Real)%%2
    interval1Real<-(as.integer(interval1Real/2)+adder)
    interval1<-(interval1Real*thNum)
    interval2<-(interval1-(adder*thNum))
    interval2Real<-(interval1Real-adder)
  } else if ((int1Direction==FALSE) && (forward==FALSE) && (backward==FALSE)) {
    #the initial march toward the end
    startReal<-startReal+interval1Real
    start<-(start+interval1)
    adder<-interval2Real%%2
    interval2Real<-(as.integer(interval2Real/2)+adder)
    interval2<-(interval2Real*thNum)
    interval1<-(interval2-(adder*thNum))
    interval1Real<-(interval2Real-adder)
  } else if (forward==TRUE) {
    #going forward after backwards initially
    #interval 1 stays the same,
    #split up the second interval and start from the same spot.
    adder<-interval2Real%%2
    interval2Real<-(as.integer(interval2Real/2)+adder)
    interval2<-(interval2Real*thNum)
  } else if (backward==TRUE) { # if backwards=TRUE
    #going backward after forward initially
    adder<-interval1%%2
```

```
    interval1Real<-(as.integer(interval1Real/2)+adder)
    interval1<- (interval1Real*thNum)
    start<-(end-interval1-interval2+1)
    startReal<- (endReal-interval1Real-interval2Real)
  } else {cat("Unknown Error!")}
}
#now to put in the first and last values
if (forward==TRUE) {
  intervalArray<-rbind (intervalArray, dim1)
} else {
  intervalArray<-append(intervalArray, dim1, after=0)
  intervalArray<-rev(intervalArray)
}

#This section is needed (to get n)
#n is the number of principal components to use
for(i in 1:dim2) {
  if ((summary(prcomp(TwoDnormalized))$importance[3,i]) > threshold) {
    n <- i
    if(n==1) { #the next steps won't work with one principal component
      n<-2
    }
    break
  }
}

#----------------------
#PCA step written on 12/10/12
for (j in 1:length(intervalArray)) {
  if(j==1) {
    EigenMatrix<-eigen(cov(TwoDnormalized[1:(intervalArray[1]*thNum),]))$vectors
    RowFeatureVec<-t(EigenMatrix[,1:n])
    RowFeatVecInv<-ginv(RowFeatureVec)
    FinalComps<-RowFeatureVec%*%t(TwoDnormalized[1:(intervalArray[1]*thNum),])
  } else {
    EigenMatrix<-abind(EigenMatrix,
        eigen(cov(TwoDnormalized[(intervalArray[j-1]*thNum+1):
        (intervalArray[j]*thNum),]))$vectors, along=3)
```

```r
    RowFeatureVec<-abind(RowFeatureVec, t(EigenMatrix[,1:n,j]), along=3)
    RowFeatVecInv<-abind(RowFeatVecInv, ginv(RowFeatureVec[,,j]), along=3)
    FinalComps<-cbind(FinalComps, RowFeatureVec[,,j]%*%
        t(TwoDnormalized[(intervalArray[j-1]*thNum+1):(intervalArray[j]*thNum),]))
  }
}
#flip it around so it's compatible with later stuff
TwoDFeatureComps<-t(FinalComps)

#----------------------
#dim1 is replaced with how many timesteps are being analyzed (to break into equal intervals)
#this puts the data into a form so it can be MSA'd
dim1<-(dim(TwoDFeatureComps)[1]/thNum)
MeanShiftReady<-array(0, c(thNum,(dim1*n)))
for (j in 1:thNum) {
  for (k in 1:n) {
    for (i in 1:dim1) {
      MeanShiftReady[j,i+((k-1)*dim1)] <- TwoDFeatureComps[(i-1)*thNum+j,k]
    }
  }
}

#------------------------------------------------------------
# Export to MSA for clustering
# ------
write.table(MeanShiftReady, file=(paste(IODir, "/PC", rstNum, ".csv", sep="")),
    row.names = FALSE, col.names=FALSE, sep=",")
save.image("R_data/RAPSspace.RData")
```

A.7. unMSAPCA.r Source Code:

```
# 7/25/12
# Written by Kevin Makinson
# Oregon State university
# This code takes the data, after performing PCA and MSA, and puts it back together again
# --------------------------------
load("R_data/RAPSspace.RData")
# now importing from MSA
FeatureCompsClusters<-read.csv((paste(IODir, "/clustCenters", rstNum, ".csv", sep="")),
    header=FALSE)
#--------------------------------------------------------------

Nclust<-length(FeatureCompsClusters[,1])
UnMeanShift<-array(0, c(Nclust*dim1, n))
for (i in 1:n) {
  for (j in 1:dim1) {
    if (j==1) {
      UnMeanShiftTemp <- as.matrix(FeatureCompsClusters[,((i-1)*dim1)+1])
    } else {
      UnMeanShiftTemp<-rbind(UnMeanShiftTemp, as.matrix(FeatureCompsClusters[,((i-1)*dim1)+j]))
    }
  }
  UnMeanShift[,i]<- UnMeanShiftTemp
}
UnMeanShift<-t(UnMeanShift)
#----------------
#this one's rewritten! 12/10/12
for (j in 1:length(intervalArray)) {
  if (j==1) {
    unMsTwoDNormalized<-RowFeatVecInv[,,1]%*%UnMeanShift[,1:(intervalArray[j]*Nclust)]
  } else {
    unMsTwoDNormalized<-cbind(unMsTwoDNormalized,
      RowFeatVecInv[,,j]%*%UnMeanShift[,(intervalArray[j-1]*Nclust+1):(intervalArray[j]*Nclust)])
  }
}

```

```
#This loop formats (reorganizes) the data to get into a form similar to normalized
FormatOrigNormData<-array(0, c(dim1,dim2,Nclust))
for (i in 1:dim1) {
  for (j in 1:Nclust) {
    FormatOrigNormData[i,,j] <- unMsTwoDNormalized[,((i-1)*Nclust+j)]
  }
}

#adding back the mean and standard deviation
FinalData <- array(0,c(dim1,dim2,Nclust),
    dimnames=dimnames(TrimData[(2:(dim(TrimData)[1]-2)),,j]))
for (j in 1:Nclust) {
  for (i in 1:dim2) {
    FinalData[,i,j] <- (sd(TrimData[(1:(dim(TrimData)[1]-2)),i,j])*
        (FormatOrigNormData[,i,j]))+mean(TrimData[(1:(dim(TrimData)[1]-2)),i,j])
  }
}

#R doesn't let you assign a unit labels to only 1 dimension
diff<-dim(array(0,c((dim1+3),(dim2+1),Nclust)))[1]-dim(units)[1]
unitsdiff<-rbind(units, array(NA, c(diff, dim(units)[2])))

FinalDataTemp2 <- array(0,c(dim1,(dim2+1),Nclust))
FinalDataTemp3 <- array(0,c((dim1+3),(dim2+1),Nclust), dimnames=dimnames(unitsdiff))

#putting in time and units. -Don't worry about error message here, works like I want
for (j in 1:dim(FinalData)[3]) {
  FinalDataTemp2[,,j]<-(cbind(time, FinalData[,(1:(dim(FinalData)[2])),j]))
  FinalDataTemp3[,,j]<-(rbind(units, FinalDataTemp2[,,j]))
}

FinalData<-FinalDataTemp3
rm(FinalDataTemp2)
rm(FinalDataTemp3)

#output to csv file
#---
if (dataOut==1) {
```

```
  for (j in 1:dim(FinalData)[3]) {
    write.table(FinalData[,,j],
        file=(paste(IODir, "/unMSAPCAc", j, "rst", rstNum, ".csv", sep="")), row.names = FALSE,
                col.names=TRUE, sep=",")
  }
}
#---
# plotting algorithm
#--
if (dataOut==1) {
  pdf(paste(IODir, "/clusterPlots", rstNum, ".pdf", sep=""),onefile=TRUE)
  for (j in 1:dim(FormatOrigNormData)[3]) {
    par(mar=(c(4,4,2,6.5)+0.1)) #sets dimensions
    plot(x=time[1:(length(time)-1)], y=FormatOrigNormData[,1,j], type="l",
         ylim=as.numeric(range(as.numeric(FormatOrigNormData[,,j]))),
         xlab="Time (s)", ylab="Normalized data (unitless)",
         main=paste("Normalized Data Cluster #",j,sep=""))
    for (i in 2:dim(FormatOrigNormData)[2]) {
      lines(x=time[1:(length(time)-1)], y=FormatOrigNormData[,i,j], col=i)
    }
    par(xpd=T)
    #legend(x=(tail(time,1)+(time[2]-time[1])/2.6),
    legend(x=tail(time,1)-7,
           y=(0.2+max(as.numeric(FormatOrigNormData[,,j]))),
           colnames(units[,-1]), lty=1, col = c(1:(dim(units)[2]-1)), bg="white")
  }
  dev.off()
}
save.image("R_data/RAPSspace.RData")
```

A.8. UpdateRwindex.r Source Code:

```
load("R_data/RAPSspace.RData")
rstNum<-3
thNum<-24
IncludeTh<- c(1, 2, 5, 6, 7, 11, 12, 13, 15, 16, 23)
EndByTrip<- c(20)
prevKeepGoing<- c(19)
cutSetProbs<- c("1.011828E-01 ( +/- 1.876009E-03 )  "," 1.008002E-01 ( +/- 1.872459E-03 )
 "," 9.937411E-02 ( +/- 1.859166E-03 )  "," 9.857411E-02 ( +/- 1.851667E-03 )
 "," 9.739150E-04 ( +/- 1.840526E-04 )  "," 1.147828E-03 ( +/- 1.998113E-04 )
 "," 9.391323E-04 ( +/- 1.807361E-04 )  "," 8.347843E-04 ( +/- 1.703996E-04 )
 "," 8.000016E-04 ( +/- 1.668119E-04 )  "," 7.304362E-04 ( +/- 1.593943E-04 )  ")
ThTransientTranslator<- c(0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 0, 1, 2, 3)
timestep <- 433
save.image("R_data/RAPSspace.RData")
```

A.9. Display.r Source Code:

```
#-------------------------------
# Written by Kevin Makinson
# 9/7/12
# Oregon State University
# This is a test program for the display part of the decision engine
#-------------------------------

load("R_data/RAPSspace.RData")
library(abind, lib.loc=libloc)
#------------------
#this loads the seed file in
seedFilePath<- (paste(IODir, "/initout.csv", sep=""))
InitOutRawData<- read.csv(seedFilePath, header=TRUE)

naCols<-0
for (i in 1:dim(InitOutRawData[,])[2]) {
  if (is.na(InitOutRawData[1,i])==TRUE) {
    naCols<-naCols+1
  }
}
naRows<-0
for (i in 4:dim(InitOutRawData[,])[1]) {
  if(InitOutRawData[i,1]==("             ")) {
    naRows<-naRows+1
  }
}

#making syntax more readable
dim1<-dim(InitOutRawData[,])[1]-naRows
dim2<-dim(InitOutRawData[,])[2]-naCols
units<- InitOutRawData[1:3,1:dim2]

SeedData<-array(0,c((dim1-3), dim2), dimnames=dimnames(InitOutRawData[4:dim1, 1:dim2]))

#redefining cols as numeric instead of characters
```

```r
SeedDataTemp<-as.matrix(InitOutRawData[4:dim1,1:dim2])
for (i in 1:dim2) {
  for (j in 1:(dim1-3)) {
   SeedData[j,i]<-as.numeric(SeedDataTemp[j,i])
  }
}
#-----------
#new section! 10/15/12
#this loads the previous restart data instead of the seed data for the prevKeepGoing case
keepGoingTrip<-intersect(prevKeepGoing, EndByTrip)
if (length(keepGoingTrip)!=0) {
  prevRstNum<-rstNum-1
  R5OutFilePaths<-array(0,length(keepGoingTrip))

  kCount=0
  for (i in keepGoingTrip) {
    kCount=kCount+1
    R5OutFilePaths[kCount]<-paste(IODir,"/Th_", i, "_data/outputs/rst",
        prevRstNum, ".csv", sep = "")
    PrevKeepGoingRsti<-paste("PrevKeepGoingRst", i, sep = "")
    assign(PrevKeepGoingRsti, read.csv(R5OutFilePaths[kCount], header=TRUE))
  }
  thNum<-length(keepGoingTrip)

  #these two loop checks to see if there are any cols or rows that have "NA" as members
  naCols<- array(0, length(keepGoingTrip))
  naRows<- array(0, length(keepGoingTrip))

  for (k in 1:length(keepGoingTrip)) {
    PrevKeepGoingRsti<-paste("PrevKeepGoingRst", keepGoingTrip[k], sep = "")
    for (i in 1:dim(get(PrevKeepGoingRsti))[2]) {
      if (is.na(get(PrevKeepGoingRsti)[1,i])==TRUE) {
        naCols[k]<-naCols[k]+1
      }
    }
    for (i in 4:dim(get(PrevKeepGoingRsti))[1]) {
      if(get(PrevKeepGoingRsti)[i,1]==("             ")) {
        naRows[k]<-naRows[k]+1
```

```r
      }
    }
  }
  #making syntax more readable
  dim1<-array(0, length(keepGoingTrip))
  dim2<-array(0, length(keepGoingTrip))

  for (i in 1:length(keepGoingTrip)) {
    PrevKeepGoingRsti<-paste("TripOutRawData", keepGoingTrip[i], sep = "")
    assign(PrevKeepGoingRsti, read.csv(R5OutFilePaths[i], header=TRUE))
    dim1[i]<-dim(get(PrevKeepGoingRsti))[1]-naRows[i]
    dim2[i]<-dim(get(PrevKeepGoingRsti))[2]-naCols[i]
  }

  for (i in 1:length(keepGoingTrip)) { #define the array
    PrevKGDatai<-paste("PrevKGData", keepGoingTrip[i], sep = "")
    assign(PrevKGDatai, array(0,c((dim1[i]-3), dim2[i]),
                            dimnames=dimnames(get(PrevKeepGoingRsti)[4:dim1[i], 1:dim2[i]])))
  }

  #redefining cols as numeric instead of characters
  for (j in 1:length(keepGoingTrip)) {
    PrevKGDatai<-paste("PrevKGData", keepGoingTrip[j], sep = "")
    PrevKeepGoingRsti<-paste("PrevKeepGoingRst", keepGoingTrip[j], sep = "")
    assign(PrevKGDatai, (get(PrevKeepGoingRsti)[4:(dim1[j]),1:dim2[j]]))
  }
}
#-----------
#now loading the tripped runs:
R5OutFilePaths<-array(0,length(EndByTrip))
#assigning file paths
#reading data
#plopping it into a 3D matrix

#need this!
kCount=0
for (i in EndByTrip) {
  kCount=kCount+1
```

```
  R5OutFilePaths[kCount]<-paste(IODir,"/Th_", i, "_data/outputs/rst", rstNum, ".csv", sep = "")
  TripOutRawDatai<-paste("TripOutRawData", i, sep = "")
  assign(TripOutRawDatai, read.csv(R5OutFilePaths[kCount], header=TRUE))
}
thNum<-length(EndByTrip)

#these two loop checks to see if there are any cols or rows that have "NA" as members
naCols<- array(0, length(EndByTrip))
naRows<- array(0, length(EndByTrip))
for (k in 1:length(EndByTrip)) {
  TripOutRawDatai<-paste("TripOutRawData", EndByTrip[k], sep = "")
  for (i in 1:dim(get(TripOutRawDatai))[2]) {
    if (is.na(get(TripOutRawDatai)[1,i])==TRUE) {
      naCols[k]<-naCols[k]+1
    }
  }
  for (i in 4:dim(get(TripOutRawDatai))[1]) {
    if(get(TripOutRawDatai)[i,1]==("             ")) {
      naRows[k]<-naRows[k]+1
    }
  }
}

#making syntax more readable
dim1<-array(0, length(EndByTrip))
dim2<-array(0, length(EndByTrip))

for (i in 1:length(EndByTrip)) {
  TripOutRawDatai<-paste("TripOutRawData", EndByTrip[i], sep = "")
  assign(TripOutRawDatai, read.csv(R5OutFilePaths[i], header=TRUE))
  dim1[i]<-dim(get(TripOutRawDatai))[1]-naRows[i]
  dim2[i]<-dim(get(TripOutRawDatai))[2]-naCols[i]
}

for (i in 1:length(EndByTrip)) { #define the array
  TripDatai<-paste("TripData", EndByTrip[i], sep = "")
  assign(TripDatai, array(0,c((dim1[i]-3), dim2[i]),
  dimnames=dimnames(get(TripOutRawDatai)[4:dim1[i], 1:dim2[i]])))
```

```r
}

#redefining cols as numeric instead of characters
for (j in 1:length(EndByTrip)) {
    TripDatai<-paste("TripData", EndByTrip[j], sep = "")
    TripOutRawDatai<-paste("TripOutRawData", EndByTrip[j], sep = "")
    assign(TripDatai, (get(TripOutRawDatai)[4:(dim1[j]),1:dim2[j]]))
}

threshNumsRawData<- read.csv(seedFilePath, header=FALSE, strip.white=TRUE)

stateVarCodes2=stateVarCodes #strange artifact from R5! Have to remove second to last zero
for (i in 1:length(stateVarCodes)) {
  stateVarCodes2= paste(substring(stateVarCodes,1,7), substring(stateVarCodes,9,9), sep= "  ")
}

#save this
count=0
for (j in 1:length(stateVarTripNames)) {
  for (i in 1:dim2[1]) { #dim2 should all be the same
    if ((as.character(threshNumsRawData[1,i]) == stateVarTripNames[j]) &&
      ((as.character(threshNumsRawData[2,i]) == stateVarCodes[j]) ||
      (as.character(threshNumsRawData[2,i]) == stateVarCodes2[j]))) {
      count=count+1
      if (count==1) {
        threshNums=i
      } else {
        threshNums=c(threshNums, i)
      }
    }
  }
}

TripDatai<-paste("TripData", (EndByTrip[1]), sep = "")
colnames(thresholds)<-colnames(get(TripDatai)[,threshNums[1:length(threshNums)]])

trippedVar<-array(0, length(EndByTrip))
trippedThresh<-array(0, length(EndByTrip))
```

```r
for (k in 1:length(EndByTrip)) { #scenario number
  TripDatai<-paste("TripData", EndByTrip[k], sep = "")
  for (j in 1:length(threshNums)) { #state variables of interest
    for (i in 1:dim(get(TripDatai))[1]) { #row number
      if(equivalence[j]=="lt") {
        if (as.numeric(as.matrix(get(TripDatai)[i,threshNums[j]]))<=(thresholds[2,j])) {
          trippedVar[k]=threshNums[j]
          trippedThresh[k]=j
        }
      } else if (equivalence[j]=="gt") {
        if (as.numeric(as.matrix(get(TripDatai)[i,threshNums[j]]))>(thresholds[1,j])) {
        }
        if (as.numeric(as.matrix(get(TripDatai)[i,threshNums[j]]))>=(thresholds[2,j])) {
          trippedVar[k]=threshNums[j]
          trippedThresh[k]=j
        }
      }
    }
  }
}

#plotting
pdf(paste(IODir, "/alerts", rstNum, ".pdf", sep=""),onefile=TRUE)
par(xpd=F)
#---
for (k in 1:length(EndByTrip)) { #scenario number
  TripDatai<-paste("TripData", EndByTrip[k], sep = "")
  PrevKGDatai<-paste("PrevKGData", EndByTrip[k], sep = "")
  if (EndByTrip[k] %in% keepGoingTrip) {
    #plotting for keepgoing tripped data
    plot(x=as.numeric(as.matrix(get(PrevKGDatai)[,1])),
        y=as.numeric(as.matrix(get(PrevKGDatai)[,trippedVar[k]])), type="l",
         xlim=c(min(as.numeric(as.matrix(get(PrevKGDatai)[,1]))),
         max(as.numeric((as.matrix(get(TripDatai)[,1]))))*1.3),
         ylim=c(min(min(as.numeric(as.matrix(get(PrevKGDatai)[,trippedVar[k]]))),
         min(as.numeric(as.matrix(get(TripDatai)[,trippedVar[k]])))),
         1.1*max(max(as.numeric(as.matrix(get(PrevKGDatai)[,trippedVar[k]]))),
         max(as.numeric(as.matrix(get(TripDatai)[,trippedVar[k]]))))),
```

```
          xlab="Time (s)", ylab=colnames(get(TripDatai))[trippedVar[k]],
          main=paste(colnames(get(TripDatai))[trippedVar[k]],
          "trip; Scen #", EndByTrip[k],
          "\n Probability = ", cutSetProbs[ThTransientTranslator[(EndByTrip[k]+1)]+1]), col=3)
    abline(v=max(as.numeric(as.matrix(get(PrevKGDatai)[,1]))), lty=2)
    text(max(as.numeric(as.matrix(get(PrevKGDatai)[,1]))),
        min(as.numeric(as.matrix(get(PrevKGDatai)[,trippedVar[k]]))),
        labels="previous restart <=", pos=2)
    text(max(as.numeric(as.matrix(get(PrevKGDatai)[,1]))),
    min(as.numeric(as.matrix(get(PrevKGDatai)[,trippedVar[k]]))),
    labels="=> current cycle", pos=4)
  } else {
    plot(x=SeedData[,1], y=SeedData[,trippedVar[k]], type="l",
        xlim=c(mean(SeedData[,1]), max(as.numeric((as.matrix(get(TripDatai)[,1]))))*1.3),
        ylim=c(min(min(SeedData[,trippedVar[k]]),
        min(as.numeric(as.matrix(get(TripDatai)[,trippedVar[k]])))),
        1.1*max(max(SeedData[,trippedVar[k]]),
        max(as.numeric(as.matrix(get(TripDatai)[,trippedVar[k]]))))),
        xlab="Time (s)", ylab=colnames(get(TripDatai))[trippedVar[k]],
        main=paste(colnames(get(TripDatai))[trippedVar[k]], "trip; Scen #", EndByTrip[k],
        "\n Probability = ", cutSetProbs[ThTransientTranslator[(EndByTrip[k]+1)]+1]), col=3)
    abline(v=max(SeedData[,1]), lty=2)
    text(max(SeedData[,1]), min(SeedData[,trippedVar[k]]), labels="seed<=", pos=2)
    text(max(SeedData[,1]), min(SeedData[,trippedVar[k]]), labels="=>transient", pos=4)
  }
  lines(x=as.numeric(as.matrix(get(TripDatai)[,1])),
        y=as.numeric(as.matrix(get(TripDatai)[,trippedVar[k]])), col=3)
  abline(h=thresholds[1,colnames(get(TripDatai))[trippedVar[k]]], col=7, lty=2)
  abline(h=thresholds[2,colnames(get(TripDatai))[trippedVar[k]]], col=2, lty=2)
  text(max(as.numeric((as.matrix(get(TripDatai)[,1]))))*1.3,
  (thresholds[1,colnames(get(TripDatai))[trippedVar[k]]]), labels="Warning", pos=2)
  text(max(as.numeric((as.matrix(get(TripDatai)[,1]))))*1.3,
  (thresholds[2,colnames(get(TripDatai))[trippedVar[k]]]),
  labels="Alert", pos=2)
}
dev.off()

#section for communicating which thresholds were exceeded
```

```r
for (k in 1:length(EndByTrip)) { #scenario number
  TripDatai<-paste("TripData", EndByTrip[k], sep = "")
  #probably also add in state variable codes
  fileConn<-file(paste(IODir,"/tripRst", rstNum, "_Sc", EndByTrip[k], ".txt", sep = ""))
  if ((equivalence[trippedThresh[k]])=="lt") {
    writeLines(paste("Sensor",stateVarTripNames[trippedThresh[k]],
        stateVarCodes[trippedThresh[k]],
        "fell bleow the", thresholds[2,colnames(get(TripDatai))[trippedVar[k]]],
        "red threshold", sep=" ") , fileConn)
  } else {
    writeLines(paste("Sensor",stateVarTripNames[trippedThresh[k]],
    stateVarCodes[trippedThresh[k]],
    "exceeded the", thresholds[2,colnames(get(TripDatai))[trippedVar[k]]],
    "red threshold", sep=" ") , fileConn)
  }
  close(fileConn)
}
save.image("R_data/RAPSspace.RData")
```

A.10. Cluster.h Source Code

```
//Cluster.h
//Created by Diego Mandelli
//Modified by Kevin Makinson

#include <vector>
using namespace std;
#ifndef CLUSTER_H
#define CLUSTER_H

class cluster{
 private:
     int dimensionality;
     int cardinality;
     double *centroid;
     vector <int> datapointsID;
 public:
     cluster() {
         centroid = NULL;
     }
     cluster(const cluster &in);
     cluster(int dimensions, double center[], int pointID);
     ~cluster() {
         if (!centroid) delete [] centroid;
     }
     int getDimensionality ();
     int getCardinality ();
     // void setNew (int dimensions, double center[], int pointID);
     void addPoint (double NewCentroid[], int pointID, int dimensions, int clustCount);
     double* getCentroid ();
};

cluster::cluster(const cluster &in) : datapointsID(in.datapointsID) {
 dimensionality = in.dimensionality;
 centroid = new double[dimensionality];
     for (int i=0; i<dimensionality; i++)
```

```
         centroid[i] = in.centroid[i];
}

cluster::cluster(int dimensions, double center[], int pointID) {
// this method add a new cluster
 dimensionality = dimensions;
 cardinality = 1;
 centroid = new double[dimensionality];
 for (int i=0; i<dimensions; i++)
     centroid[i] = center[i];
 datapointsID.push_back(pointID);
}

void cluster::addPoint (double NewCentroid[], int pointID, int dimensions, int clustCount) {
 //cout << "Point added!" << endl;
 // this method add a new point to an existing cluster and update the cluster center
 cardinality=clustCount-1;
 for (int i=0; i<dimensions; i++) {
     centroid[i] = (centroid[i]*cardinality+NewCentroid[i])/(cardinality+1);
 }
 //dimensionality does not change;
 cardinality++;
 //PointID is the scenario number that gets removed and combined with another cluster
 datapointsID.push_back(pointID);

}
double* cluster::getCentroid() {
 return centroid;
}

int cluster::getCardinality () {
 return cardinality;
}

int cluster::getDimensionality () {
 return dimensionality;
}
#endif
```

A.11. MeanShift.h Source Code

```
// MeanShift.h
// Originally Created on: Aug 9, 2010
// Original Author: Diego Mandelli
// Modified on: 7/24/12
// Modified Author: Kevin Makinson
#ifndef MeanShift_h
#define MeanShift_h
#include <fstream>
#include "cluster.h"
#include <math.h>
#include <vector>
#include <stdlib.h>
#include <cmath>
#include <sstream>
#include <iomanip>
#include <sstream>
//using namespace std;

//Functions
void MeanShiftOperator(double *NewPosition, double *point, double **data,
    double h, int card, int dim);
double LpNorm(double p, double x[], int NDim);
int FindClosestCentroid (vector<cluster> clusterSet, double NewCentroid[],
    int p, int dim, double h);

vector <vector <int> > MeanShift (int Windex, double BW, string OutDir,
    vector <int> IncludeTh, vector <int> translator){

    stringstream sstm;
    string PCfile;
    sstm << "PC" << Windex << ".csv";   //adding index to the string
    PCfile = sstm.str();
    sstm.str("");

    //ifstream fin("data.csv");
```

```
  ifstream fin((OutDir + "/" + PCfile).c_str());
  //vector <int> record2;
  vector <vector <double> > dataVec;  //added KM 7/18/12 for determining the size of the data
  vector <vector <int> > clustMembers;  //added KM 7/23 for storing the cluster membership.
  int index=0;  //added KM for clusterMembers, rename later. 7/23
  double number; //this is for single entries in the data to be pushed onto data vector
  //inputting data into a vector
  while (fin) {
    string s;
    if (!getline(fin, s ))
      break;
    istringstream ss( s );
    vector <double> record;
    while (ss) {
      string s;
      if (!getline( ss, s, ',' ))
        break;
      istringstream(s) >> number;
      record.push_back( number );
    }
    dataVec.push_back( record );
  }
 if (!fin.eof()) {
  cerr << "Cannot open PCA file!\n";
 }


  // Variable definitions
  int cardinality = dataVec.size(); // Number of scenarios
  int dimensionality = dataVec[0].size(); // Number of dimensions for each scenario

  // Access data and store it in a 2D array //
  double **data = new double*[cardinality];
  double **centroid = new double*[cardinality];
  double *pointIn = new double[dimensionality];
  //double BW = 20; //
  int p=2;
  for(int j = 0; j < cardinality; j++) {
```

```
        data[j] = new double[dimensionality];
        centroid[j] = new double[dimensionality];
    }

    for(int i = 0; i < cardinality; i++) {      //reveresed card and dim KM 7.18
        for(int j = 0; j < dimensionality; j++) {
            data[i][j]=dataVec[i][j];
        }
    }

    // End data input session
    // Perform clustering //
    // Initialize the set of clusters (ClusterSet.size() gives size of vector)
    vector<cluster> ClusterSet;
    //#pragma omp parallel for //disabling parallel processing for the time being.
        for(int i = 0; i < cardinality; i++) { //Perform MSM for each data point
            // perform MeanShift for point i and get the centroid for each point
            MeanShiftOperator(centroid[i], data[i], data, BW, cardinality, dimensionality);
        }
        for(int i = 0; i < cardinality; i++) { //Perform MSM for each data point
            if(IncludeTh[translator[i]]==1) { //if it is not flagged as "keep going"
                //update cluster centroid
                int check = FindClosestCentroid (ClusterSet, centroid[i], p, dimensionality, BW);
                if(check==-1) {
                    vector <int> record2; //gotta clear record2 each time!
                    record2.push_back(translator[i]);
                    clustMembers.push_back(record2);
                    index++; //this is the cluster number KM 7/23
                    cluster temp(dimensionality, centroid[i], i);
                    ClusterSet.push_back(temp);
                }
                else {
                    clustMembers[check].push_back(translator[i]);
                    ClusterSet[check].addPoint(centroid[i], i,
                        dimensionality, clustMembers[check].size());
                }
            }
        }
```

```
// End clustering //

    //output cluster centers and membership
    string clustCentFile;
    sstm << "clustCenters" << Windex << ".csv"; //adding index to the string
    clustCentFile = sstm.str();
    sstm.str("");
    ofstream fout((OutDir + "/" + clustCentFile).c_str());
    for (unsigned int j=0; j<ClusterSet.size(); j++) {
        for (int i=0; i<dimensionality; i++) {
            fout << ClusterSet[j].getCentroid()[i] << ",";
        }
        fout << endl;
    }

    string clustMembFile;
    sstm << "clustMemb" << Windex << ".csv";    //adding index to the string
    clustMembFile = sstm.str();
    sstm.str("");
    ofstream f2out((OutDir + "/" + clustMembFile).c_str());
    //cluster membership
    for (unsigned int j=0; j<clustMembers.size(); j++) {
        for (unsigned int i=0; i<clustMembers[j].size(); i++) {
            f2out << clustMembers[j][i] << ",";
        }
        f2out << endl;
    }

//deleting the memory created at run-time
for(int j = 0; j < cardinality; j++) {
    delete [] data[j];
    delete [] centroid[j];
}
delete [] data;
delete [] centroid;
delete [] pointIn;

return clustMembers;
```

```
}


void MeanShiftOperator(double *NewPosition, double *point, double **data, double h,
    int card, int dim){
    double p=2; // Norm type
    double epsilon = h*0.01; // Convergence parameter
    double den=0;
    double modX=0;
    double m_x=0; // initialize m_x: new position - old position
    double *OldPosition;    //changing to dynamic array KM 7/18/12
    OldPosition = new (nothrow) double[dim];
    for (int i=0; i<dim; i++) {
        OldPosition[i]=point[i];  //point is the data
    }

    //double diff[dim];
    double *diff;               //changing to dynamic array KM 7/18/12
    diff = new (nothrow) double[dim];
    for (int j=0; j<dim; ++j)
        NewPosition[j] = 0.0;  //zeros new position
    do {

        for (int i=0; i<card; i++) { // find all the point in the sphere with radius=bandwith/2
            double *pointIn=data[i];
            for (int j=0; j<dim; j++){
                diff[j] = OldPosition[j]-pointIn[j];
            }
            modX = LpNorm(p,diff,dim);
            if (modX < h/2) {
                for (int j=0; j<dim; j++) {
                    NewPosition[j] += pointIn[j] *exp(-(modX*modX)/(h*h));
                }
                den = den + exp(-(modX*modX)/(h*h));
            }
        }
    for (int j=0; j<dim; j++)
        diff[j]=OldPosition[j]-point[j];
```

```
    m_x = LpNorm(p,diff,dim);
    for (int j=0; j<dim; j++)  {  // changed <= to just <
        NewPosition[j] /= den;  //YES this is where this is supposed to be
        OldPosition[j] = (NewPosition[j]);
    }
    } while (m_x > epsilon);

    delete[] OldPosition;  //added these guys KM 7/18/12
    delete[] diff;
}

double LpNorm(double p, double x[], int NDim) {  //changed this KM
    // Determine the p-norm of an NDim-dimensional vector x
    double norm=0;
    double temp=0;
    if (p==0) { // L infinite
        for (int i=0; i<NDim; i++) {
            temp = abs(x[i]);
            if (temp>norm)
                norm=temp;
        }
    }
    else { // Lp
        for (int i=0; i<NDim; i++) {
            norm += pow(abs(x[i]),p);
        }
    norm = (pow(norm,1/p));
    }
    return (norm);
}

int FindClosestCentroid (vector<cluster> clusterSet, double NewCentroid[],
    int p, int dim, double h) {
// Find the closest centroid to NewCentroid and return the position of that point
    int answer = -1;
    double modX;
    //double distanceFromMinimum = 999;
    double distanceFromMinimum = (h/3); //modified by KM
```

```
    double *diff;
    diff= new (nothrow) double[dim];
    for (unsigned int i=0; i<clusterSet.size(); i++) {
        for (int j=0; j<dim; j++) {
            diff[j] = NewCentroid[j]-clusterSet.at(i).getCentroid()[j];
        }
        modX = LpNorm(p,diff,dim);
        if (modX<distanceFromMinimum) {
            distanceFromMinimum = modX;
            answer = i;
        }
    }

delete[] diff;
return (answer);
}
#endif
```

A.12. RAPSS-STA Example Input File

```
* ==================================
* RAPSS-STA input file
* Written by Kevin Makinson
* Oregon State University
* ===================================
* R5 Parameters
* ===================================
* input file
101 Stepup.i
* output file
102 Stepup.p
* water file
103 tpfh2onew
* Restart file parameters
* End Time (seconds)
104 100
* Minimum Time Step
105 1E-7
* Max Time Step
106 0.1
* Control Mode
107 3
* Minor Edit
108 100
* Major Edit
109 100
* Restart Frequency
110 100
* =====================================
* R, PCA and MSA parameters
* =====================================
* PCA Threshold
201 0.95
* BandWidth for MSA
202 4
* Path for R library files to be downloaded into
203 /nfs/stak/students/m/makinske/lib
* Website for downloading R files
204 http://cran.r-project.org
* =====================================
* RAPS Parameters
* =====================================
* Path to RELAP5 executable is stored
301 /usr/local/neapps/relap5-3d/r3d2412ie/relap
```

```
* 1 or 0 (true or false) for output cluster csv and pdf files
302 1
* Directory where the R5 input and water files are stored (IDir)
303 /nfs/chadwick/u1/makinske/R5run
*requested number of threads
304 24
*timestep advancements for "keep going" restart files
305 1000
*timestep advancements from first "seed" run
306 1000
*FTAdir, directory where fta_input_file and run.sh are stored
307 /nfs/stak/students/m/makinske/cpp/FTA
*FTA folder/file nanme, name of model without any file type appended
308 maslwr
*Number of cutsets to grab from OpenFTA.
309 10
*State variables for thresholds (arbitrarily set for now for the MASLWR deck)
310 httemp p p
*R5 model state variable codes
311 133000101 100010000 500010000
*Raise flag if less than (lt) or greater than (gt) given threshold
312 gt lt gt
*Yellow trip for "keep going"
313 650 3.5E+6 1.0E+7
*Red trip, to stop run
314 1000 3E+6 9.0E+7
*FTA parameters - cut set order, unit time, terms, number of monte carlo simulations
315 10 1 10 10000
*Real time simulator data file name
316 IAEASP3.txt
```

B. Appendix B: RAPPS-STA Source Code Explanation

The C++ structure contains one .cpp file, RAPSmain.cpp, as well as five header files, CycleR5.h, BloodAndGuts.h, cluster.h, MeanShift.h, and OrganizeR5Output.h. Cycle R5 contains the primary loop, which controls the cycling of the program. RAPSmain.cpp calls CycleR5, as well as reads the RAPSS-STA input file. Cluster.h and MeanShift.h are used to call the mean shift algorithm, originally written by Diego Mandelli (2011), but modified and updated to serve RAPSS. OrganizeR5Output.h contains the structure for reading R5 output files and extracting the pertinent information. Finally, BloodAndGuts.h contains a plethora of miscellaneous functions called by the program at various times. Although they are included with many compilers, it is also worth noting that the libraries: omp.h, sstream, stdlib.h, vector.h, string, iostream, fstream, stdio.h, iomanip, time.h, math.h, cmath, and sstream are required as well.

There are three, prewritten R files that are called by the structure at various points in the program, PCA.R, display.r, and unMSAPCA.R. PCA.r reads the output from OrganizeR5Output, in the form of a .csv file, runs principal component analysis, and outputs the data in a form that can be easily read by MeanShift.h. UnMSAPCA.r retrieves the information generated by the mean shift algorithm, reorganizes it, performs reverse principal component analysis, and outputs the new clustered data in the same units as the original. Display.r plots the seed data, followed by any scenario that tripped due to a “red” threshold being exceeded (see Section 7), with the accompanying red and yellow thresholds on the plot. These take the form of a PDF, and are generated any time a red threshold is exceeded.

In addition, two other R files are written and called by the C++ structure as the program runs. These are initPCA.r, and updateRwindex.r. InitPCA.r is called only once and initializes R by downloading necessary libraries and passing necessary information from the RAPSS-STA input file. UpdateRwindex.r is called after every cycle. This updates information necessary for data processing from the last cycle.

B.1. RAPSmain.cpp Source Code Explanation

RAPSmain.cpp (see Appendix A.1) is the file that "runs" RAPSS-STA. Lines 015-052 are variable definitions which have been extracted to the top level in order to allow the user to edit these variables via the RAPSS-STA input file (see Section 5.1.1). The main() function of RAPSmain.cpp begins with local variable definitions (lines 055-062). After a call to change the font to green to differentiate it from the standard UNIX terminal (line 062), there is some user interface, asking the user if he or she would like to begin the program. If the user selects "yes," the user is then asked to type the name of the RAPSS input file. Once the input file is read, appropriate directories are created if they do not already exist, and appropriate variables are defined as instructed by the input file. These variables are then are passed onto a single function, CycleR5(). The real meat of the program resides in CycleR5(), which is defined in CycleR5.h (see Appendix A.2), one level below RAPSmain.cpp.

The function, RAPSinputFile(), that reads the input file is defined in RAPSmain.cpp (lines 125-403). LoadFile() (defined in Appendix B.3) is used to read the input file and load it onto a vector, named inputVec[]. The function, RAPSinputFile(), iterates through each line number of the input file. First, it checks for the comment character, "*"; if the first character of the line does not contain the comment character, then it will read the first three characters of the line. This is where the card numbers are expected. Each card number has one or more RAPSS variables associated with it, assigned through a large switch-statement. Since the vector contains only strings, numeric variables are coerced into their given data-type through the use of the istringstream() operator. If the input file contains cards that are not assigned to variables, or if there are errors in the input, an error message is returned for each card incorrectly entered, and the cards that do not contain errors are assigned appropriately.

B.2. CycleR5.h Source Code Explanation

CycleR5.h (see Appendix A.2) is the first level below RAPSmain.cpp and consists of a single function, CycleR5(). Lines 027-080 contain local variable definitions. A large while-statement is the primary control mechanism for RAPSS-STA. The loop continues while the string variable, *answer*, is either “y,” “yes,” “Y,” or “Yes.” *Answer* begins as yes, and is updated by the user at line 178, or 185 with each loop. When the user changes answer to “n,” “no,” “N,” or “No,” it will break from the while-loop, which pops from CycleR5.h to RAPSmain.cpp, outputting, “Thank you for running RAPSS-STA,” and terminating the program.

The first set of tasks (lines 110-124) assigns names to the R5 restart output files, input files, the previous output file, and the name of the raw data output .csv files (to be used in OrganizeR5Output.h, see Appendices A.4 and B.4). These strings are appended with the variable *Windex*, or while-index, which counts the number of times the while-loop is executed. This corresponds to how many restart runs RELAP5 will run.

EndByVec[] (line 309) is a vector that contains a zero, one, or two for each transient corresponding to a termination by: end of time step card, trip, or steady state, respectively, as described in the R5EndBy() bullet in Appendix B.3. EndByVec[] is simply a storage device for R5EndBy(). In a similar way to EndByVec[], RstNbr[] (line 311) is simply a storage device for FindRstNbr(), described in Appendix B.3. The first time through the while-loop, zero is pushed onto RstNbr[] (line 128), but every other time, the restart number from the previous run is pushed onto RstNbr[], just as the code for transient termination type is pushed onto EndByVec[].

When the *Windex* is one, or the first iteration through CycleR5(), several tasks are performed. After some user interface, initR() is called (lines 160-161), which simply writes an initialization R file (initPCA.r) to be used with the appropriate libraries and variables (see Section 5.3). After it calls initPCA.r, the function WriteInitShFile() (see Section 5.3) is called, which creates a simple UNIX shell script that changes to the appropriate directory, calls RELAP5 with the passed information and returns the full file path of the shell script. After the script is read by the system, OrganizeR5Output() is called (line 172) to extract the data from the R5 output (see Appendices A.4 and B.4).

When the *Windex* is not one, (i.e., on the second or greater times through the while loop), it enters into the parallel portion of the code (line 241). If the requested number of threads (requestTh), specified in the RAPSS-STA input file, is greater than the maximum allowable by the system, an appropriate error message is displayed and the number of threads is set to the maximum (lines 217-225). If the number of requested threads is not greater than the maximum, but greater than one, OpenMP is set to use the specified number of threads. If both of the aforementioned arguments are not true (i.e., if the number of requested threads is either less than one, or not a numeric value), OpenMP is set to only use two threads (lines 230-235). Running RAPSS-STA using only one thread will cause the program to be unstable, and is not allowed. After initializing OpenMP using *#pragma omp parallel*, the integer, th_id, is stored as the identification number of the thread (line 244). This is important to differentiate scenarios by the thread number they were run on.

If Windex is two, or the second time through the while-statement, but the first time in the parallel portion, it will make a new data storage directory for each thread (lines 250-253). These take the form of Th_(*th_id*)_data. Inside the directory, there are two more directories created, "inputs" and "outputs," which store the restart input files and R5 output, as well as the organized R5 output in the .csv file format. It then calls RstIptGen() (see Appendix B.3), which writes a restart file for R5 with the appropriate conditions (lines 258-263, 267-273, and 275-281). WriteRstShFile() (lines 283-285) (see Appendix B.3), writes a shell script that runs R5 using the appropriate directory information for the active thread.

If Windex is not two or one, (i.e., on the third or greater times through the while loop), it skips creating the folders, and simply calls RstIptGen() (line 275), which writes a new restart file, running a combination of transients and non-transients with perturbed initial conditions.

Lines 305-340 display to the user, and store in the system how the R5 run ended on each simulation. This is passed to the 2D vector, EndBySumVec[][] (line 314), which stores the output from EndBySummary() (see Appendix B.3), and is used to output to the user how many threads ended by the end of allotted time, trip, steady state, or errors. When prevKeepGoing[] is true for a given thread that has completed its time history, EndBySumVec[1][] is set to false for that thread. Lines 392-404 take out scenarios that are tripped and flagged for keepGoing from clustering contention. This is because to correctly cluster scenarios, they need to be over the same time-space. Scenarios that terminated by trip predict less time than scenarios that ended by their allotted time. Similarly, scenarios that continue going for the next cycle exist in a larger time-space than the normal scenarios.

The vectors keepGoing[] and prevKeepGoing[] (lines 342-391) are used to tell RAPSS-STA whether to terminate a run at the end of the time history, or explore further in time. This is achieved by checking if certain “yellow” thresholds are exceeded. These yellow thresholds are values of state variables that aren’t immediately of concern, but might be in the future. Scenarios that are flagged to keep going will continue where the previous run left off and explore further in time rather than starting from the current time (lines 267-274). These vectors have nthread (the number of threads used) elements, and contain zeros or ones corresponding to keepGoing[th_id] being true or false. The vector prevKeepGoing[] is simply the values of keepGoing[] from the previous cycle.

Finally, the information from the newly completed R5 cycle is passed into R by calling updateRwindex. PCA.R (see Section 2.5.1) is called at line 446. This reads the .csv file from each thread, organizes it for PCA, performs PCA, and prepares it for the mean shift algorithm (see Section 2.5). It then calls MeanShift() (line 448) from MeanShift.h (see Appendix B.11), which outputs the cluster centers as a .csv file. unMSAPCA.R (see Appendix B.7) is called to reorganize the data a final time, and outputs a .csv file for each cluster, as well as a single multi-paged .pdf file plotting the cluster centers for all state variables from scenarios that did not end by trip. For those that did end by trip, only the tripped state variable is plotted with accompanying thresholds and probability information.

The tripped data are plotted through display.r (line 426). The runs that completed their time histories are clustered using PCA and MSA and the clustered data are reorganized and plotted using unMSAPCA.r (line 450). Then, to reset for the next cycle, prevKeepGoing[] is set to keepGoing[], and keepGoing[] is cleared. The function, htmlDisplayWriter(), is called at line 452 to generate the user interface.

B.3. BloodAndGuts.h Source Code Explanation

As suggested by the title of this header file, this file contains the "blood and guts" of RAPSS-STA (See Appendix A.3). It is basically a collection of various functions to be used in other parts of the program. This file can be considered the third layer below RAPSmain.cpp. The functions contained in BloodAndGuts.h will be briefly explained in this section.

- SearchVec() (lines 0019-0037) is one of the most heavily used functions in RAPSS. It expects a vector of strings (the R5 output), labeled text[], and a string *key*. It searches through a data file, and returns a vector of the integer line numbers where the key is located. An example of its use is to locate the key "1 time " to determine the location of the state variable time series data.
- LoadFile() (lines 0039-0057) expects a string of the input file path to load. This function takes a file and loads it into a vector, which can be easily manipulated in C++. It returns the vector, text[], which is a vector of strings. Each index of the vector is a string corresponding to the line number of the input file.

- FindRstNbr() (lines 0061-0081) expects a string of the input file path to search for a restart number in the R5 output. This function calls LoadFile(), then passes the vector obtained by LoadFile() as well as the key, "0---Restart no," in this case, to SearchVec() to obtain a vector of strings. It then searches the line that starts with *key* and grabs the number immediately after the key, and stops when it reaches a "w" character. This is because in the R5 output file, after the word "written" always follows the restart number. It returns the double, RstNbr, which corresponds to the last restart number in the R5 output file. This is potentially used to begin a new restart file where the last run left off.
- R5EndBy() (lines 0085-0104) searches the R5 output file for one of three phrases, "0Transient terminated by end of time step cards," "0Transient terminated by trip," or "0Transient has reached steady state," and returns a 1, 2, or 3, respectively. It also has the capability of returning zero if it does not find one of the three phrases.
- EndBySummary() (lines 0109-0147) is a function that returns a two-dimensional vector and expects a one-dimensional vector of the termination type in the format of R5EndBy(), as well as the number of threads currently in use. To construct the 2-D vector for return, four temporary vectors (temp0-3) are created and the thread number that corresponds to the termination type is pushed onto the temporary vector. These vectors are then pushed onto the 2D vector, which is returned by the function. The first index of the return value, EndBySumVec corresponds to how the transient on the given thread was terminated, 0: errors, 1: time step, 2: trip, and 3: steady state. The second index corresponds to the thread number. The

value associated with the two given indices can be either a zero or one, to signify whether a scenario terminated by a certain way on a certain thread. For example if EndBySumVec[1][8] was equal to one, it would mean that thread 8 ended by the end of time step cards. If EndBySumVec[1][8] was equal to zero, it would mean that thread 8 ended by something other than the end of the time step cards.

- R5SciConv() (lines 0151-0159) is a very simple function which converts scientific notation from how C++ outputs it (i.e., 1.0e+3 = 1000) to how RELAP5 processes scientific notation (i.e., 1.0+3 = 1000).
- WriteInitShFile() (lines 0161-0169) writes a UNIX shell script to change to the appropriate directory and perform the initial RELAP5 run using the designated input, output, restart, and water files. It returns a string of the full file path to the newly written script.
- WriteRstShfile() (0171-0186) writes a UNIX shell script to change to the appropriate directory, and perform the restart RELAP5 run using the designated directory of restart, input, output, restart, and water files. It returns a string of the full file path to the shell script.
- initR() (lines 0188-0260) writes the initialization file for R. This first clears the R memory to avoid any overflow from the last experiment, and makes a new directory, if it doesn't already exist in a location designated by the user, libloc (library location), from card 203 of the RAPSS-STA input file (see Section 5.1.1). It then stores variables and installs the necessary packages for processing in R. This usually takes a few seconds if they are not already installed, as it has to download these from the internet, which is why it is only performed once, instead

of with each restart. It also passes the red and yellow threshold information, state variable trip names, equivalence and state variable codes from the RAPSS-STA input cards (lines 310-314) (see Section 5.1.1).

- UpdateRwindex() (lines 0263-0337) is a function that communicates to R the changes in rstNum and thNum, the number of threads being run. It also passes the threads that ended by timestep (EndBySumVec[1][]) as well as those that ended by trip (EndBySumVec[2][]). In addition, the threads that were designated as "keep going," and the probabilities from LiteFTA are also passed for the display mechanism. ThTransientTranslator is also passed. This is a vector that converts from the way C++ interprets less than nthread values, to the way R interprets them. For example if there are 8 total threads and threads 1, 3, and 6 were not included in data processing, C++ would see this as including threads (0, 2, 4, 5, 7), whereas R would see 5 threads and number them (1, 2, 3, 4, 5). So any attempt to display to the user which threads are contained in which cluster (see Section 5.1.1), would not yield acceptable results. The ThTransientTranslator[] addresses this issue.
- ChangeFont() (lines 0339-0346) writes a short shell script to change the font color of the text, using standard UNIX color codes. This is used to enhance the clarity of the program and differentiate RELAP output from RAPSS output. It returns the location of the shell script.
- RestetFont() (lines 0348-0355) is similar the ChangeFont(), except it changes the font back to its original color.

- NameDir() (lines 0358-0364) is a small function that simply returns a string of Th_(*th_id*)_data where *th_id* is the thread identification number from OpenMP. This is used for creating new directory locations.
- TrimSpace() (lines 0368-0381) expects a string with white space either at the beginning, end, or both of the input string, and trims them to return the string without the whitespace.
- getCutSetData() (lines 0384-0463) returns a 2D vector of strings and expects a .prp file name as well as the number of cutsets to grab. To grab the cutsets, the function first looks for information from the same cutset on the next line, and enters into one of two different procedures depending on the results of that logic, called multLine. If multiLine is false, it will read in each cutset word, separated by a space and pass them, stored as the string "word," to a vector named eventVec[], which represents a single cutset. Word is cleared after each pass to eventVec[]. Once it sees two spaces in a row, it clears eventVec[] proceeds to the next cutset. For multiline cutsets, a new Boolean variable, lastLineInSet is introduced. This is set to true if the next line from the cutset contains information from the next set rather than the set it is currently loading. It tells the function to keep going to the next line to grab more cutset information, or stop, pass the information, and begin loading a new cutset. At the end of the while loop, the character index, i, is set to 6. This corresponds to the character number where each cutset begins in the file. The probability vector, prob[], is also grabbed during this process but in a much simpler manner. Since the probability is always

contained between characters 39 and 52, it simply passes this value from the first line of the cutset onto the prob[] vector.

- getMCdata() (lines 0466-0548) returns a 2D vector of strings and expects a .mrp file name as well as the number of cutsets to grab. It functions in a very similar manner to getCutSetData().
- doFTA() (lines 0550-0576) expects a vector of doubles: FTApars (FTA parameters), the LiteFTA file name, and the directory where the FTA files exist. These are specified by the user in card 315. This function writes the LiteFTA input file (fta_input_file), which simply tells LiteFTA where to look for the .fta file as well as which parameters to use while generating the .prp and .mrp file. It also generates a new shell script named runFTA.sh which changes to the appropriate directory and executes run.sh, which is the shell script that actually runs LiteFTA. It finishes by making them executable using the UNIX command "chmod +x", and running the scripts.
- ftaFileFixer() (lines 0578-0598) is a simple function that expects a file name of an OpenFTA file fault tree to be run in LiteFTA. LiteFTA expects the data in a certain format, and ftaFileFixer() puts it in that format.
- loadSystemData() (lines 0600-0618) uses LoadFile to load a .csv file and returns a vector of strings, commonly called sysData[][].
- qualConverter() (lines 620-658) take the value of LDP301 in the MASLWR output data, and determines what the value of qual will be at the next restart. Qual is basically an indication of if there is water at that location or not. A value of 1.0 indicates water, a value of 7.0E-3 indicates not water.

- Linterpolate() (lines 660-664) performs a linear interpolation of the input variables.
- realTimeSimulator() (lines 0666-0680) simply loads whatever timestep has been written to realTimeData.txt.
- RstIptGen() (lines 0682-2071) See section B.3.1.
- htmlDisplayWriter() (lines 2074-2345) is the function that creates the html display. Lines 2088-2104 count how many times the red and yellow trips have happened in the scenarios to use in creating the red and yellow boxes in the display (see Section 7). Lines 2106-2125 name and create the files to be used, DISPLAY#.html, green#.html, unstable#.html, and misc#.html. Where the # symbol is used to represent the cycle number. DISPLAY#.html is created in lines 2147-2240. Green#html is the file that shows all scenarios that did not trip, in other words, any R5 run that ended by the end of the time step cards, and is written on lines 2242-2273. Misc#.html is written on lines 2275-2311 and contains the cluster information about the scenarios. Unsable#.html is written on lines 2313-2344 and contains the scenarios that ended by the R5 model becoming unstable, and a description of what that particular scenario was attempting to model.

B.3.1 RstIptGen()

RstIptGen() (lines 0682-2071) is the largest function in BloodAndGuts.h (and in RAPSS-STA). It writes the restart files for each thread. Local variable definitions are contained in lines 0691-0708. Lines 0710-0766 search the MASLWR data for the variables of interest and coerce them (from strings to doubles) onto local variables.

These state variables are perturbed in lines 0770-0787 and 0802-0804 across random numbers between 0.95 and 1.05 by multiplying by U. U begins as a uniform distribution, from 0 to 1. This is achieved by dividing a given random number by the maximum allowable random number set by the system (library dependent, but guaranteed to be at least 32767). This produces a uniform distribution between 0 and 1. That number is then divided by 10 and added to 0.90, producing a uniform distribution between 0.95 and 1.05. The value of U is reset using a combination of system time and arbitrary multipliers to insure a wide variety of random numbers.

The temperatures are converted from Fahrenheit to Kelvin (lines 790-799) and redundant thermocouple measurements are averaged (line 0800).

To begin to write the restart file, there are first the obligatory data, author, and description comment card (lines 0813-0816). Then, it writes the 100 card (problem type), 103 card (restart number) which uses the restart data (containing geometry information) from the previous run, and the 203 (time) card.

Next, RstIptGen() sets the termination trips (i.e., the “red” trips). RELAP 5 has a rather strange way of prototyping trips. First, the trips themselves need to be stated. For example, the line:

501 p 100010000 lt null 0 6.00+06 l

states that trip number 501 is set to true when the pressure in component 100010000 is less than $6x10^6$. The user specifies how many of these he or she needs in the input file. The 600 card is the RELAP5 “terminate transient by trip” card. However, there can only be one trip entered on the 600 card. For simulations with more than one end-by trip desired, cards 601-699 are used to combine the logic from cards 501-599. However, cards 601-699 can only tolerate two entries, therefore multiple cards are necessary to add up the desired amount of trips. For example, if there are trips for both pressure and temperature, cards might look like:

501 p 100010000 lt null 0 6.00+06 l

502 httemp 501000101 gt null 0 1.10+03 l

601 501 or 502 l -1.0

603 601

Or, if there are three (or an odd number of) outputs, one of the 501-599 cards is repeated by RAPSS-STA on the last 601-699 card (602 in the lines below):

501 p 100010000 lt null 0 6.00+06 l

502 httemp 501000101 gt null 0 1.10+03

503 velgj 209000000 lt null 0 2.00+00 l

601 501 or 502 l -1.0

602 503 or 503 l -1.0

603 601 or 602 l -1.0

600 603

This is achieved through a rather non-intuitive bit of logic. First RAPSS-STA simply lists the trips and increments (using the index, *i*) the 500 card for each trip. Next, the 601-699 cards collect up the 501-599s. The index, *index60* is used to increment the 601-699 cards. The index, *index*, is used in the 601-699 cards to refer to the 501-599 cards. *Index*, starts at zero and is multiplied by two and either has a one or two added to it. This is because there are two entries per 601-699 card and the 501-599 cards start at zero. For example, the first time through the loop, index is 0, so the 601 card is:

500+(index*2)+1, and 500+(index*2)+2, in other words, 501 and 502. This goes on until index is equal to the number of state variable trips divided by two (because there are two 501-599 trips listed for each 601-699 card). If the number of trips is odd, the same logic is used, except on the last time through, the 501-599 cards it references are identical. This redundancy was added to significantly simplify the logic, while still maintaining functionality.

If the number of trips is one, then simply the 501 card is called in the 600 card. If the size is not two, or one (greater than two) a final bit of logic is invoked. It turns out that for seven or more trips, the way the 601-699 cards collect the 500 cards, (index*2)+1, starts to depart from the desired result. For iterations greater than 7, the card numbers become off by one, increasing again by one for every four cycles. The integer *adder* was created to handle this problem, it increments every fourth cycle past 7 by checking when the modulus of the index by four is equal to three, in other words, when the next index is divisible by four.

The next section of RstIptGen() (lines 0877-1561) is used to define the start conditions of a given R5 run based on measurements from the MASLWR facility. To do this, any component where a value is set needs the entire card reproduced in the restart file. To accomplish this, large sections of the R5 input was regenerated with new values in certain places. Each time a state variable is set to a value from the MASLWR facility, it was marked with a comment card beginning with the word “TOMNOTE:”

The next section of RstIptGen() (lines 1564-2071) is used to simulate transient conditions. If the thread id is greater than or equal to the number of threads divided by two (the second half of the threads), then a transient condition is simulated. Otherwise, normal operating conditions are simulated with perturbed initial conditions. The vector, transient[], is passed from CycleR5(), and contains cut sets from the fault tree that produces a core-damage transient with the highest probability. The first section (lines 1564-1675), pushes transient codes onto the vector, Vbreak[]. The transient codes correspond to the next section, which contains the appropriate R5 output to simulate a given transient. For example, if one of the entries in transient[] is RPV_F, the code 7520, is passed to Vbreak[], which corresponds to case 7520 (line 1793), and simulates the a reactor pressure vessel leak (through the use of the “breaker valve” on the cold leg). For transients with multiple occurrence paths, a random occurrence path is chosen. For example, if FLOW_B (line 1610) was read from transient[], then one of four codes for “breaker valves” would be passed to Vbreak.

The logic for the code number of the breaker valves is fairly straight forward. The code number corresponds to the component number of the breaker valve that simulates the given transient followed by a zero for fully engaged valve, or a one for partially engaged valve.

B.4. OrganizeR5Output.h Source Code Explanation

OrganizeR5Output.h (see Appendix A.4) was written to search a RELAP5 output for state variable time series information and write it to a .csv file. The function, OrganizeR5Output() was written to expect two strings: the file path for the R5 output, and the path for the writing of the organized .csv file. The first loop in OrganizeR5Output() (lines 039-081) grabs string fragments, starting with “1 time .” The two spaces after “time” differentiate it from the possible instances of this in the input file (which is always displayed at the beginning of the R5 output file). The string fragments are grabbed thirteen characters at a time (the thirteen characters is simply a result of the inherent organizational structure in the R5 output), and after trimming the extra white space at beginning and the end of the string fragments, pushes them onto a new 2D vector of strings, named data1[][]. The function knows when to stop recording time series data and move to the next section when it reaches either "RELAP5/3.3g" or "steady state" in the R5 output. It is necessary for the loop read the last line to tell the program that the previous line was the last one with data, but avoid writing the new line, because it does not contain any data. Line 079 accomplishes this by *popping back* (deleting) the final entry into the data1[][] vector (data1.pop_back()) whenever it reaches the end of a set or section. The function knows when it has reached the end of the final section by looking for the key words "Final time=" or "---Restart Su."

Unfortunately, at this point, data1[][] still contains labels, units, time variables for every set. This results in every 50 time steps, new state variables (including all the corresponding labels and units) are displayed for the same 50 time steps until all state variables are output. This is complete chaos from a data analysis perspective because it does not follow a consistent time series, and repeated display of labels and units nullify any attempt to, say, take the average of a column. The time variable also appears multiple times corresponding to how many total sets appear in the R5 output.

The next three loops solve these problems. The first (lines 085-089) pushes the line number of the start of each set from data1[][] (not the R5 output, as SearchVec() would) onto a new vector, named data1Sections[]. Next (lines 092-106), the SearchVec() of "1 time " for the R5 output file is used to discover how many sets of state variables per section exist. This is determined by the loop which assigns a single integer value to the variable "SetsPerSection." The loop works by comparing the first element from the time series in one set from data1[][] with the first time series element in the next set. If the time series element is the same, it's still in the same section. If the time is different, it is a new section.

Now that the above information is known, it's possible to begin organizing the data. The second to last major loop (lines 122-144) cycles from zero to the number of sections in data1[][] (data1Sections.size()). An if-else statement was added to account the varying number of state variable groupings (SetsPerSection) depending on the user input. The integer, HeaderLength, is initialized to zero at the beginning of the code. HeaderLength determines whether the labels and units in the R5 output are included in the organized output. Labels and units are necessary only for the first section. The first statement (line 125) waits until k, the index of the number of sections, reaches the value of SetsPerSection, indicating the first full section has been read and organized. After that, labels and units are redundant, so HeaderLength is set and left at four in order to start four lines (the length of the labels and units in an R5 output) later than it did in the first section.

If the index (k) of the loop's *modulus* (remainder) of SetsPerSection is zero, jIndex and jMult are set to 0; StatMult is also incremented; otherwise, jIndex is set to unity, and jMult is set to the modulus of k and the sets per section (lines 125-133). These values are used in the nested two upcoming loops.

This next bit of logic is some of the more complex built for the current version of RAPSS-STA; for this reason, they will be explained step by step. First, the integer, SectionStart, is found by subtracting the integer, FormatSectionLength, from the

HeaderLength (line 135). FormatSectionLength is simply the difference in line numbers between the first two sections (data1Sections[1]-data1Sections[0], line 115). StartMult is incremented only when the modulus of k by the SetsPerSection is zero. In other words, StartMult increments every time a new section starts. SectionStart, then is the length of a set, multiplied by how many sections have been completed. This is used to control where the length-variable of FormatData[][] starts.

The integer, FormatDataWidth, is set to the width of the section (line 136). For example, if there are 18 state variables, with ten variables per section, FormatDataWidth would be ten the first time through the loop, and eight the second time.

The next nested for-loop (lines 137-144) controls the indexing of i and j, the length and width variables, respectively, of data1[i][j]. The index, i, runs from the beginning of the set at k in data1[][], and grabs the units for the first section, but skips them for all other sections, by the addition of HeaderLength. The index, i, basically runs from the beginning of a set to the end of a set (data1Sections[k+1]). The width index, j, is cycled by starting from either 0, the first word, corresponding to the time series variable, or 1, corresponding to the first variable besides the time series variable (controlled by jIndex). This way, FormatData[][] removes the multiple time series variable problem described above. The index, j, runs until the end of the width of a given set.

The next if-statement (line 139) designates the end of all sections. The statement basically says, if the index, i, (the length index) reaches the size of the length of data1Sections[], break from the loop.

Now we're finally ready for the most important statement of the whole function, the organization of the data (line 140). The goal is to take data1[][], which is separated by sets of state variables, 10 across, and 50 in length, and move them to the appropriate spot in the new format vector, FormatData[][], corresponding to all state variable labels and units displayed as the first row, and the state variable time series values occupying the remaining rows. The indices, i and j, are simply used for data1[i][j], which limits the index manipulation only to the FormatData[][] vector. The length index for FormatData[][] begins at SectionStart (length of the set, multiplied by how many sections have been completed), and is added to i (the line number of data1[][]) subtracted by the line number from the beginning of the set (data1Sections[k]). This makes sure the state variables from new sections are added to the bottom of the desired state variable columns.

The length index, j, for FormatData[][] is equally as non-intuitive. It begins with jIndex (one or zero depending on whether it's the first time though the loop or not, to grab the time variable the first time, but not more than once). The multiplier, jMult, multiplies the width of the current section from data1[][]. jMult is the modulus of k and the sets per section. This tells FormatData[][] how many sets of columns (state variables) to skip when recording data. This removes the requirement of only ten state variables per set. The rows now appear as a more logical structure, with the time variable first, followed by however many state variables are output by R5.

The final loop (line 147-155) simply outputs the formatted data into a .csv file given by the second parameter OrganizeR5Output() expects.

B.5. initPCA.r Source Code Explanation

This script is generated by the C++ structure to pass important variables to the R world. This is meant to initialize R and prepare it for principal component analysis. This script is only ran once, at the beginning of RAPSS-STA. *Threshold* (line 11) is the variance threshold for PCA to cut off when performing dimensionality reduction *Thresholds* (lines 15-17) is the array that holds the "red" and "yellow" thresholds defined in *stateVarTripNames* (line 18) and *equivalence* (line 19). *StateVarCodes* is an array that simply holds the component numbers from R5 for each of the state variables listed in *stateVarTripNames*. Equivalence is an array holds either "gt" for greater than, or "lt" for less than. This is used later to test whether the values in stateVarTripNames are above or below the values listed in *thresholds.*

B.6. PCA.r Source Code Explanation

The script, PCA.r (Appendix A.6), does quite a bit more than simply perform PCA. First, it reads the .csv output file generated by OrganizeR5Output() for each thread, and stores it in a 3-dimensional array (**time steps** *x* **state variables** *x* **thread**) (lines 022-036). Included in these arrays are small amounts of null data, generated by unused memory allocation during OrganizeR5Output(). The null data are identified and deleted (lines 039-050), as well as any state variable with a standard deviation below a given threshold across each scenario to avoid interference with further analysis (lines 073-089). Next, because the data was read as a combination of characters and numeric values, R assumed each value is a type of character value. They were redefined, and read into a new array, TrimData, using the R function: as.numeric() (lines 059-065). Time and the state variable units were also trimmed at this point and stored for use later. It was discovered much later that read.csv has a parameter that can be modified to act the same as the as.numeric() function.

The first and last data points for each state variable are ignored due to the peculiar feature of RELAP5 outputting constant values for all but the first and last data points when a state variable is allegedly constant over a given time series. The state variable would be flagged for deletion if it were constant for all except for the first and last values. If any state variables were flagged for deletion in one scenario, but not another, the variables were not trimmed (lines 113-120).

The next step is to normalize the data to set it up for PCA. To accomplish this, the state variables' means were subtracted and the quantity was divided by the standard deviation to obtain new state variable columns with a mean of zero and standard deviation of one (lines 134-139).

After the data was normalized, it had to be arranged in such a way to obtain a single eigenvector matrix (the rotations), as opposed to an eigenvector matrix for each thread. The goal of performing PCA was to reduce the number of state variables for input to the mean shift algorithm. To do this, the first time steps of each scenario were grouped together, followed by the second time steps and so on (see Section 6.2). The result was for a single state variable, 24 measurements (assuming 24 threads) of each time step. The new matrix was named TwoDnormalized (lines 145-153) in PCA.r, and has dimensions (**number of threads***time steps** *x* **state variables**).

Lines 155-288 are related to the Automated Linear Approximation Interval Sequencer (ALIAS) and are described in section B.6.1.

The array, EigenMatrix, represents entire set of eigenvectors (line 294). The array, RowFeatureVec (295), represents only the eigenvectors that correspond to the eigenvalues that add up to the user defined variance threshold. The array, RowFeatVecInv (line 296), is the inverse of RowFeatureVec, to be used later. The array, FinalComps (line 297), is composed of the components themselves, to be used with the mean shift algorithm.

Instead of organizing the data by state variable, and incorporating the scenarios into the state variables, the data must be organized by scenario, and the principal components (formally state variables) are grouped similar to Equation 5.1. This matrix is labeled MeanShiftReady and is written to a .csv file as “PC,” followed by the restart number that it is analyzing from RAPSS-STA (lines 315-322), to be read in the mean shift algorithm.

B.6.1 The Automated Linear Approximation Interval Sequencer (ALAIS)

The Automated Linear Approximation Interval Sequencer (ALAIS) begins at line 155 of PCA.r. The logic makes the most sense, however, by starting at line 189, which splits the total time interval in half and finds the covariance matrices of these two sections. Lines 192-194 checks to see if the norm of the difference in covariance matrices divided by the change in time is below a certain threshold, as described in Equation (6.8). When it’s not below the threshold, the intervals must be split further. Whichever interval’s covariance norm is the greatest determines which interval gets split. At this point, a Boolean flag, int1Direction, is activated (line 193), which tells the rest of the statement which direction it’s headed. True indicates marching towards the beginning, false, indicates marching towards the end. We will discuss the case when int1Direction is true first.

To avoid an infinite number of interval tests, after the initial direction is determined, and the norm of the difference in covariance matrices is not below the set threshold, it must continue splitting up the interval closest to the beginning until the threshold is reached (lines 238-244). Once the norm of the difference in covariance matrices is below the set threshold, and the first interval starts at 1 (meaning it is at the beginning), the Boolean flag, *forward*, is set to true (line 201). The location of the split between the first and second intervals is then stored for later use in intervalArray. The first interval is then set to the second interval, and the second interval is made to extend all the way to the end (lines 254-260). If these intervals do not pass the test, the second interval would be cut in half (line 250), and the test performed again. Once it is determined that the interval is small enough, it will store the data in intervalArray, set the first interval equal to the second, and extend the second interval all the way to the end (lines 254-260 again). If it does not pass the test is will continue splitting intervals using the aforementioned procedure. If it does pass the test, it will break from the loop, and add in the ending data point for use later.

For the first case when deciding which direction to initially head, if the second interval's covariance norm is greater, int1Direction is set to false, and very similar logic is executed by first marching toward the end, then once the end is found and the difference in covariance matrices is below the threshold, it would set the "second" interval equal to the "first," and the "first" equal to the beginning. ALIAS would then march towards the beginning in a very similar manner to how it marched toward the end, described in the above paragraph.

Unfortunately, decreasing the size of the interval doesn't always decrease the norm of the difference in covariance matrices. It could be, in rare cases, that the state variables are actually less correlated along a smaller time interval than they are over a larger time interval. In this case, when the intervals decrease to a size of two and still don't pass the norm of the covariance matrices test, ALIAS will activate a Boolean flag, *moveOn*, record the interval that did not pass the test in the array, badInterval, and move on as if it did pass the test. This usually happens at the very beginning of the data, when possibly the initial conditions of RELAP5 do not match the steady state conditions a short time after the system has had a chance to respond to the initial conditions.

B.7. unMSAPCA.r Source Code Explanation

The first step of unMSAPCA.r is to load the data in the form of a single matrix (**cluster size** *x* (**time steps***principal components**) in size). Lines 014-023 convert it into the same form as the feature vector, RowFeatureVec, from PCA.r (see section B.6) ((**cluster size***time steps**) *x* **principal components**). After that, the inverse of the feature vector matrix, RowFeatVecInv (lines 027-034) is used to transform from principal components to state variables for each linear approximation interval obtained in the ALIAS algorithm (see section B.6.1) into physically meaningful data.

At this point, we have a single 2D matrix of the form: ((**scenario clusters***time steps**) *x* **state variables**). Lines 036-042 convert the data into a 3D matrix of the form (**time steps** *x* **state variables** *x* **scenario clusters**). The data are then unnormalized by adding in the mean and multiplying by the standard deviation (lines 044-052). Time and units are added back in lines 055-059. Finally, when the Boolean data output variable is set to true, the data, FinalData is output, in the form of one .csv file for each cluster (lines 071-079), and one, multi-paged .pdf of the normalized cluster centers (lines 083-101).

B.8. UpdateRwindex.r Source Code Explanation

This is a script that is run with every cycle of RAPSS-STA. It updates the cycle number, *rstNum* (line 02), and which scenarios ended by end of time step, *IncludeTh* (line 04), ended by trip, *EndByTrip* (line 05), and which scenarios were started in previous cycles, *prevKeepGoing* (line 06). It also updates the probabilities of transients occurring from the fault tree, *cutSetProbs* (line 07). It also passes a variable, *ThTransientTranslator* (line 12), which tells R which transient was run on which thread.

B.9. Display.r Source Code Explanation

The purpose of this script is to display the “alerts.pdf” plots. After loading the seed file (for restart runs that ended by trip) and searching for NA data (lines 012-027), it redefines the data as numeric instead of characters (lines 037-039). It was later discovered that read.csv has a parameter that does this for the user, but it was never implemented. Lines 042-100 load the previous restart in the same manner as above for the threads that are marked as *keepGoing*. Lines 101-158 load each of the scenarios that end by trips in the same manner as before. Lines 161-165 create another array, *stateVarCodes2*, which contain the state variable codes from R5 with the second to last zero removed. This is to account for a strange artifact in the R5 output that occasionally removes the second to last zero in the minor edit variable outputs. Lines 168-181 count how many thresholds to look for. Lines 183-207 determine which thresholds the scenarios tripped. Plotting occurs in lines 213-261. Inside the plotting structure, lines 216-230 plot the output if it has been flagged as *keepGoing*. This plots the last cycle’s data along with the current cycle’s data. If the scenario was not flagged as *keepGoing*, it is plotted in lines 238-250. This plots the seed data, followed by current cycle’s data.

Lines 251-260 plot and label the threshold lines. Finally, lines 263-280 create a simple .txt file for communicating back to the C++ structure which variables ended by trip in a way that the user can understand. This is eventually read, and spit out in the html interface.

B.10. Cluster.h Source Code Explanation

This file was unchanged from the original source code. See Mandelli (2011) for details.

B.11. MeanShift.h Source Code Explanation

Rather than spend time “reinventing the wheel”, RAPSS uses a version of the mean shift algorithm written in C++ mostly by Mandelli (2011), however, the code was adapted in several places to account for errors and to serve RAPSS.

- Firstly, the file itself was changed from .cpp file to a .h file for integration into RAPSS.
- The data input section (lines 029-060) was added to push the data into a vector (DataVec[][]) to dynamically determine the size (DataVec.size()) of the input data at run time, instead of the user manually typing the dimensions of the data.
- The loop in lines 78-82 seemed to confuse cardinality with dimensionality; those were switched in the RAPSS version.
- Utilization of parallel computing through OpenMP was disabled (line 088) for this version of RAPSS.

- A 2D vector, clustMembers (loaded in line 100), was added to keep track of the cluster membership. This vector was also passed as the return value of the function, MeanShift().
- The outputs (lines 115-125), output a.csv file clustCenters_restart#.csv, and clustMemb_restart#.csv, where _restart# is the restart iteration of RAPSS when MeanShift() is called.
- The new memory created at run time was deleted at the end of MeanShift() (lines 141-147).
- In the MeanShiftOperator() function, OldPosition[dim] and diff[dim] were changed to dynamic arrays because dim (dimensionality) was now determined at run time (lines 160, 167), and deleted later to avoid dangling memory (lines 195-196).
- The loop at line 186 was changed from j<=dim to j<dim to avoid accessing null memory.
- The statement: NewPosition[j] /= den (line 190) was moved into the appropriate loop. Otherwise it would clear den (denominator) each time through the while statement, which caused den to only account for cluster sizes of two members or less. On a personal note, this error was by far the hardest to identify.
- In FindClosestCentroid(), distanceFromMinimum was changed from 999 to h/3, where h is the bandwidth (line 225).

B.12. RAPSS-STA Input File Explanation

This file has three sections, RELAP5 parameters, PCA and MSA parameters, and RAPSS parameters. Lines that begin with the comment character “*” are not read. Lines that begin with a three-digit number are followed by information the user wishes to pass to RAPSS-STA. Cards 104, 105, 106, 107, 108, 109, and 110 must be identical to the time card in the RELAP5 seed input.

RELAP5 Parameters:

- Card 101: R5 initial input file. This is the file that contains all necessary geometry and heat structure information referred to as the "seed." It will be run before any other transients are simulated. While its purpose is to achieve steady state, this file is usually run in "transient" mode. Steady state mode reduces the specific heats of all metallic components in order to make the system respond quicker to thermal changes. This is not desired in RAPSS. For more information, see the RELAP5 manual, Section 2.2.3.2, Volume V.
- Card 102: R5 initial output file. This is simply the name passed for the output file from the seed calculation.
- Card 103: Water file. This is the water data file used by R5. The file must reside in the same directory as the R5 seed input file.
- Card 104: End time. This is the end time of the R5 seed run in seconds. Note: this value must be the same as designated in the R5 seed input file. It will eventually be added to in the restart runs.
- Card 105: Minimum time step. A parameter passed to the R5 restart runs, suggested value: 1E-7. It is only used in rare circumstances.
- Card 106: Maximum time step. This is also known as the "preferred" time step. This, in combination with end time, reliably controls the speed of the R5 restart calculations. For greater speed, and lower resolution, choose a larger time step;

for slower speeds and higher resolution, choose a smaller time step. This value cannot be below the minimum time step, specified on card 105.

- Card 107: Control mode. A parameter passed to the R5 restart runs, suggested value: 3. This value is only modified in rare circumstances, which are beyond the scope of this dissertation. For more information see the RELAP5 Manual Appendix A.
- Card 108: Minor edit. The minor edit frequency (in seconds) for the R5 restart runs.
- Card 109: Major edit. The major edit frequency (in seconds) for the R5 restart runs.
- Card 110: Restart frequency. The frequency (in seconds) for restart data files to be written.

PCA and MSA Parameters:

- Card 201: PCA threshold. This variable is used in principal component analysis (see Sections 2.5 and 6.4). This is a number, less than 1, which represents the amount of desired variance captured by PCA. Suggested value: greater than 0.9.
- Card 202: MSA bandwidth. This is the bandwidth used in the mean shift algorithm (see Sections 2.5 and 6.6). It controls the cluster size, and membership. Smaller bandwidths yield more clusters with fewer members. Larger bandwidths yield fewer clusters with more members per cluster.
- Card 203: R library file path. This is the file path to the desired location for storing R libraries. If the location does not already exist, it will be created. The path should start, but not end with a forward-slash (“/”).

- Card 204: R website. This is the website RAPSS will access to download R libraries. Suggested address: http://cran.r-project.org.

RAPSS Parameters:

- Card 301: Path to RELAP5 executable file directory. The path should start, but not end with a forward-slash ("/"). For example: /nfs/chadwick/a1/neapps/rhel5/RELAP5_MOD3.3/Executables/linuxifc/relap5.x
- Card 302: Cluster information output. This is a binary/Boolean value, (0 for false, 1 for true), indicating whether the user desires clustering information plots and .csv files.
- Card 303: Input directory. This is the directory where the initial R5 seed file is stored, as well as the R5 water file. A directory within this will be created for the RAPSS data.
- Card 304: Number of threads. Desired number of threads to run RAPSS with. If the number is greater than the maximum number of threads on the cluster, RAPSS will use the maximum number. If the user enters "max," then the maximum number of threads will be used.
- Card 305: Restart time increase for "keep going" scenarios. This is the desired amount of time, in seconds, to be increased if any scenarios are flagged to "keep going."
- Card 306: Restart time increase from the end of the initial seed file.
- Card 307: Fault tree directory. This is the directory where the fault tree.fta and .ped files are kept. Path should start, but not end with a forward-slash, e.g., /nfs/stak/students/m/makinske/cpp/FTA/cookTree

- Card 308: FTA file name. This is the name of the .fta and .ped files. These files should be named the same. The name is the entry for card 308 without any file type appended, for example "Cook" would be entered if the fault tree file was named Cook.fta.
- Card 309: Number of cutsets to process from LiteFTA.
- Card 310: Minor edit state variables used for determining yellow and red thresholds. These should be entered with a space between them, for example: p httemp velgj.
- Card 311: R5 model state variable codes. These are the variable codes used in R5 to differentiate pressure in one component from pressure in another. These correspond to the order in card 310 and should also be entered sequentially, separated by a space, for example 100010000 501000101 209000000.
- Card 312: Equivalence. For the state variables specified in cards 310-311. This card is used to answer the question, "Should RAPSS raise flags when the values are greater than (gt) or less than (lt) the value given in card 313?" Each entry should be separated by a space. For example, lt gt lt.
- Card 313: Yellow trip. These are the values that RAPSS looks for in the state variables (minor edit requests) with the equivalence given in card 312 to signal that a scenario should keep running for another cycle. In other words, these thresholds represent something that looks like it might be interesting in the future, but hasn't gotten there yet. Each entry should be in the same order as in cards 310-312 and separated by a space. For example, 7E+6 800 2.8

- Card 314 Red trip. These are the values that RAPSS looks for in the state variables (minor edit requests) with the equivalence given in card 312 to signal that a scenario should be terminated and information displayed to the user. Each entry should be in the same order as in cards 310-313 and separated by a space. For example 6E+6 1100 2.0.
- Card 315 LiteFTA parameters. These are the parameters associated with LiteFTA. They are cut set order, unit time (for Monte Carlo calculations), chosen terms, and number of Monte Carlo simulations. Each entry should be entered in the order described above and be separated by a space. For example: 10 1 10 10000.
- Card 316 Real time simulator data file name. This is the large data file expected in the format from the previous MASLWR experiments. RAPSS-STA reads this file and outputs the data incrementally in real time, simulating the data output if the MASLWR facility was running

C. Appendix C: RAPPS-EOC Source Code

Appendix C contains the source code for RAPSS-EOC. Readers are encouraged to read Appendix D, which explains in detail the processes at work in RAPSS-EOC. Pmain.cpp contains the operational structure at work in RAPSS-EOC. CyclePlume.h is the main control structure of RAPSS-EOC and controls the cycling of the plume program and data analysis. FunctionsEOC.h is a collection of functions used in RAPSS-EOC, similar to BloodAndGuts.h (Appendix A.3). GridOrganizer.r takes the concentration grids from each scenario and groups them for mean shift analysis. PlumeDisplay.r contains the scripts that plot the plume. InitR.r is a short script that initializes the global parameters used in R. UpdateRwindex.r updates the cycle numbers among a few other parameters with every cycle of RAPSS-EOC. Finally, a sample RAPSS-EOC input file is provided in Appendix C.9.

C.1. Pmain.cpp Source Code

```
//1/8/13
//Written by Kevin Makinson
//Oregon State University
//This is the main control structue for RAPSS-EOC

#include "functionsEOC.h"
#include "plumeProgram.h"
#include "cyclePlume.h"
#include "MeanShift.h"
#include "cluster.h"

void RAPSSinputFile(string RAPSSinput); //function that reads input file, defined below
double Q;
double hE;
double z;
int gridResolution;
int maxY;
int dt;
int runAheadTime;
int requestTh;
int BW;
int plumeStartTime;
int simulationStartTime;
int realTimeSpeedUp;
string AMPMplumeStartTime;
string AMPMsimulationStartTime;
string answer, RAPSSinput, inDir, outDir, Rrepos, libloc, windDataFile, windObsDataFile;
vector<size_t> positions;
size_t pos;

int main() {

    system(ChangeFont(2));
    cout << "Welcome to RAPSS-EOC" << endl << "Written by Kevin Makinson"
        << endl << "Last compiled on " << __DATE__ << " at " << __TIME__
```

```
        << endl << "Begin run? (y/n)" << endl;
    cin >> answer;

    if ((answer == "n") || (answer == "N") || (answer == "no") || (answer=="No")) {
        cout << "Thank you for running RAPSS" << endl;
        system(ResetFont());
        return 0;
    }
    while ((answer != "n") && (answer != "y") && (answer != "Y") && (answer != "N") &&
        (answer != "yes") && (answer != "no") && (answer != "Yes") && (answer != "No")) {
                cout << "You did not enter a \"y\" or an \"n\"!" << endl;
                cout << "Begin run? (y/n)" << endl;
                cin >> answer;
            }

    cout << "Please type the name of RAPSS-EOC input file (e.g., input.rapss): ";
    cin >> RAPSSinput;
    cout << endl;

    ifstream fin((RAPSSinput).c_str());
    cout << "Reading: " << (RAPSSinput).c_str() << endl;
    while (!fin) { //added the break statement
        cout << "File does not exist!" << endl
            << "Please carefully type the name of RAPSS-EOC input file, or type \"exit\": ";
        cin >> RAPSSinput;
        cout << endl;
        ifstream fin(RAPSSinput.c_str());
        if (RAPSSinput=="exit") {
            cout << "Thank you for running RAPSS-EOC" << endl;
            system(ResetFont());
            fin.close();
            remove("ChangeFont.sh");
            remove("ResetFont.sh");
            return 0;
        }
        else if (fin.good()) {break;} //added because the "while" statement doesn't work
    }
    fin.close();
```

```
    RAPSSinputFile(RAPSSinput.c_str()); //reads input file
    cout << "Plume has been active since: "<< plumeStartTime << " " << AMPMplumeStartTime
        << endl;
    cout << "Time is now: " << simulationStartTime << " " << AMPMsimulationStartTime << endl;
    //converts to 24 scale.
    if ((AMPMsimulationStartTime=="PM") && (simulationStartTime!=12)) {
        simulationStartTime+=12;
    }
    if ((AMPMplumeStartTime=="PM") && (plumeStartTime!=12)) {
        plumeStartTime+=12;
    }

    outDir = (inDir + "/RAPSS_data"); //Assigning Output Directory inside the input directory
    system(("rm -rf " + outDir).c_str()); //this removes it if it already exists (to overwrite)
    string CreateDataDir= ("mkdir -p " + outDir);
    system(CreateDataDir.c_str()); //creates a directory for data output

    //Here's what "runs" the program
    cyclePlumeProgram(Q, hE, z, maxY, gridResolution, runAheadTime, plumeStartTime,
        simulationStartTime, dt, realTimeSpeedUp, requestTh, BW, inDir, outDir,
        windDataFile, windObsDataFile, Rrepos, libloc);

    cout << "Thank you for running RAPSS-EOC" << endl;
    system(ResetFont());
    remove("ChangeFont.sh");
    remove("ResetFont.sh");
    return 0;

}

//function for reading input file
void RAPSSinputFile(string RAPSSinput) {
    //variables for this program
    int cardNo;
    vector <string> inputVec;
    inputVec = LoadFile(RAPSSinput);
    for (unsigned int i=0; i<(inputVec.size()); i++) {
```

```
        if (inputVec[i][0] != '*') {
            istringstream(string(inputVec[i].begin(), inputVec[i].begin()+3)) >> cardNo;
            switch (cardNo) {
                case 101: //R5 Parameters: Cards 100-199
                    istringstream(string((inputVec[i].begin()+4), inputVec[i].end())) >> BW;
                    break;
                case 102:
                    libloc= string((inputVec[i].begin()+4), inputVec[i].end());
                    break;
                case 103:
                    Rrepos= string((inputVec[i].begin()+4), inputVec[i].end());
                    break;
                case 201:
                    istringstream(string((inputVec[i].begin()+4), inputVec[i].end()))
                        >> requestTh;
                    break;
                case 202:
                    inDir= string((inputVec[i].begin()+4), inputVec[i].end());
                    break;
                case 203:
                    istringstream(string((inputVec[i].begin()+4), inputVec[i].end()))
                        >> realTimeSpeedUp;
                    break;
                case 301:
                    istringstream(string((inputVec[i].begin()+4), inputVec[i].end()))
                        >> gridResolution;
                    break;
                case 302:
                    istringstream(string((inputVec[i].begin()+4), inputVec[i].end()))
                        >> maxY;
                    break;
                case 303:
                    istringstream(string((inputVec[i].begin()+4), inputVec[i].end()))
                        >> hE;
                    break;
                case 304:
                    istringstream(string((inputVec[i].begin()+4), inputVec[i].end()))
                        >> z;
```

```cpp
            break;
        case 305:
            istringstream(string((inputVec[i].begin()+4), inputVec[i].end()))
                >> Q;
            break;
        case 306:
            istringstream(string((inputVec[i].begin()+4), inputVec[i].end()))
                >> dt;
            break;
        case 307:
            istringstream(string((inputVec[i].begin()+4), inputVec[i].end()))
                >> runAheadTime;
            break;
        case 308:
            positions.clear();
            positions.push_back(0);
            pos = string((inputVec[i].begin()+4), inputVec[i].end()).find(" ", 0);
            if (pos==string::npos) {
                istringstream(string((inputVec[i].begin()+4), inputVec[i].end()))
                    >> plumeStartTime;
            } else {
                while(pos !=string::npos) {
                    positions.push_back(pos);
                    pos = string((inputVec[i].begin()+4),
                        inputVec[i].end()).find(" ", pos+1);
                }
                for (int j=0; j<positions.size(); j++) {
                    if (j==0) {
                        istringstream(string((inputVec[i].begin()+4+positions[j]),
                            (inputVec[i].begin()+4+positions[j+1]))) >> plumeStartTime;
                    } else if (j==(positions.size()-1)) {
                        AMPMplumeStartTime=(string((inputVec[i].begin()+5+positions[j]),
                            (inputVec[i].end())));
                    }
                }
            }
            break;
        case 309:
```

```
                    positions.clear();
                    positions.push_back(0);
                    pos = string((inputVec[i].begin()+4), inputVec[i].end()).find(" ", 0);
                    if (pos==string::npos) {
                        istringstream(string((inputVec[i].begin()+4), inputVec[i].end()))
                            >> simulationStartTime;
                    } else {
                        while(pos !=string::npos) {
                            positions.push_back(pos);
                            pos = string((inputVec[i].begin()+4),
                                inputVec[i].end()).find(" ", pos+1);
                        }
                        for (int j=0; j<positions.size(); j++) {
                            if (j==0) {
                                istringstream(string((inputVec[i].begin()+4+positions[j]),
                                    (inputVec[i].begin()+4+positions[j+1])))
                                        >> simulationStartTime;
                            } else if (j==(positions.size()-1)) {
                                AMPMsimulationStartTime=(string((inputVec[i].begin()+5+positions[j]),
                                    (inputVec[i].end())));
                            }
                        }
                    }
                    break;
                case 310:
                    windDataFile = string((inputVec[i].begin()+4), inputVec[i].end());
                    break;
                case 311:
                    windObsDataFile = string((inputVec[i].begin()+4), inputVec[i].end());
                    break;
                default:
                    cout <<  "Card not read:" << endl;
                    cout << string(inputVec[i].begin(), inputVec[i].begin()+3) << endl;
            }

        }
    }
}
```

C.2. CyclePlume.h Source Code

```cpp
//Created by Kevin Makinson
//1/25/13
//This is a header file to go with Pmain.cpp for RAPSS-EOC
//This is the main control structure

#ifndef cyclePlume_h
#define cyclePlume_h
#include <omp.h>
#include "MeanShift.h"

void cyclePlumeProgram(double Q, double hE, double z, int maxY, int gridResolution,
    int runAheadTime, int plumeStartTime, int simulationStartTime, int dt, int realTimeSpeedUp,
    int requestTh, int BW, string inDir, string outDir, string windDataFile,
    string windObsDataFile, string Rrepos, string libloc) {

    //these are defined locally.
    stringstream sstm;
    double theta, windSpeedAvg, windSpeed;
    double t1, t2;
    int stabClass;
    int windDir;
    string answer="y";
    string MkThDirPath;
    string currentStateOut;
    string conditions;
    string prevCurrentState;
    int k=1;
    int Windex=0;
    int numOfCycles=0;
    int cycleCounter=1;
    int realTime=0;
    int oldRealTime;
    int th_id, nthreads;        //thread identifier & # of threads
    bool firstRead=true;
    vector <string> ThDir;  //name of thread directory
```

```
vector <vector <string > > sysData;
vector <vector <string > > realTimeData;
vector <string>  transientExplanation;
vector < vector <double> > grid;
vector < vector <double> > gridTemp;
vector <vector <int> > flagVec;
vector < vector <double> > log10grid;
vector <vector <int> > clustMembers;
transientExplanation.resize(requestTh, "");
flagVec.resize(3, vector <int> (requestTh, 0));
stabClass=1; //keeping stab class the same for now

sysData=loadWindObsData((inDir + "/" + windObsDataFile ).c_str());

while ((answer == "y") || (answer == "yes") || (answer == "Y") || (answer == "Yes")) {
    Windex++;

    if (Windex==2) { //copying files over
        sstm << "cp " << inDir << "/Alert.gif " << outDir << endl; //alerts
        system(sstm.str().c_str());
        sstm.str("");
        sstm << "cp " << inDir << "/tswtabs.css " << outDir << endl; //buttons
        system(sstm.str().c_str());
        sstm.str("");
    }

    if (numOfCycles<=cycleCounter) { //only go into this loop when you're out of cycles
        if (Windex==1) {
            cout << "Begin run? (y/n)" << endl;
            cin >> answer;
            initR(Rrepos, libloc, outDir, requestTh, Windex, runAheadTime,
                gridResolution, maxY, Q);
            system("R CMD BATCH R_data/initR.r R_data/initR.Rout");
            t1=omp_get_wtime(); //starts timer
        } else {
            cout << "Next run index is " << Windex << ". Continue run? (y/n)" << endl;
            cin >> answer;
        }
```

```
            //if the program doesn't recognize the response, tell the use to enter a y or n
            while ((answer != "n") && (answer != "y") && (answer != "Y") && (answer != "N")
                && (answer != "yes") && (answer != "no") && (answer != "Yes")
                && (answer != "No")) {
                cout << "You did not enter a \"y\" or an \"n\"!" << endl;
                cout << "Next run index is " << Windex << " Continue run? (y/n)" << endl;
                cin >> answer;
            }
            if ((answer == "n") || (answer == "no") || (answer == "N") || (answer == "No")) {
                break;
            }
            cout << "How many cycles would you like to run? Type 0 to exit. " << endl;
            cin >> numOfCycles;
            while (cin.fail())  {
                cin.clear(); //repairing buffer
                cin.ignore(10000,'\n'); //clearing buffer
                cout << "Please carefully the number of cycles.  Type 0 to exit." << endl;
                cin >> numOfCycles;
            }

            if (numOfCycles<1) {break;}
            system("read -p \"Press the [Enter] key to continue...\"");
            cycleCounter=0;

        }
        cycleCounter++;

        //real time simulator reproduces data until given timestep
        t2=omp_get_wtime(); //grabs time
        //realTime = int(t2-t1);
        oldRealTime=realTime;
        realTime =  int((t2-t1)*realTimeSpeedUp/3600);

        cout << "---------------Cycle " << cycleCounter << "---------------" << endl;
        cout << "Time sampled: " << (realTime+simulationStartTime) << ":00" << endl;
        realTimeData=realTimeSimulator(sysData, plumeStartTime,
            (realTime+simulationStartTime),outDir);

```

```
        updateRwindex(Windex, (realTime+simulationStartTime)); //updating info in R
        system("R CMD BATCH R_data/updateRwindex.r R_data/updateRwindex.Rout");

        //determining average wind speed for a given direction
        if (Windex==1) { //if first time through, go through next loop normally
            k=0;
            firstRead=true;
            oldRealTime=realTime-1; //just make it different than real time for the first time
        } else { //if already been through, load the grid, and then procede
            k=1+oldRealTime+(simulationStartTime-plumeStartTime);
            sstm << outDir << "/currentState" << Windex-1 << ".csv";//adding index to the string
            prevCurrentState = sstm.str();
            sstm.str("");
            grid=loadGridData(prevCurrentState); //loaded unlogged grid data
        }
        while ((k<(realTimeData.size()-1)) && (realTime-oldRealTime!=0)) {
            k++;
            istringstream(string(realTimeData[k][3])) >> windDir;
            istringstream(string(realTimeData[k][2])) >> windSpeedAvg;
            theta=(windDir+90)*(pi/180); //changes from North=0 to North=pi/2
            if (firstRead==true) {
                cout << "Creating estimate of current state..." << endl;
                system(ChangeFont(4));
                cout << "Simulating: " << 1 << " hour(s) with a wind coming from the"
                    << radianDirTranslator(theta) << " with a speed of " << windSpeedAvg
                    << " m/s" << endl;
                grid=PlumeProgram(Q, windSpeedAvg, hE, theta, z, stabClass, maxY,
                    gridResolution, 1, dt); //runs the plume program
                log10grid.resize(grid.size(), vector<double> (grid[0].size(), 0));
                system(ChangeFont(2));
                firstRead=false;
            } else {
                cout << "Creating estimate of current state for the next wind direction..."
                    << endl;
                system(ChangeFont(4));
                //load currentState(Windex-1).csv and put it into grid
                //if time is different than when the last cycle past
                //simulate  the number of time steps of the difference.
```

```
                cout << "Simulating: " << 1 << " hour(s) with a wind coming from the "
                    << radianDirTranslator(theta) << " with a speed of " << windSpeedAvg
                    << " m/s" << endl;

                gridTemp=PlumeProgram(Q, windSpeedAvg, hE, theta, z, stabClass, maxY,
                    gridResolution, 1, dt); //runs the plume program
                system(ChangeFont(2));
                for (int j=0; j<grid.size(); j++) {
                    for (int i=0; i<grid[0].size(); i++) {
                        grid[j][i]+=gridTemp[j][i];
                        log10grid[j][i]=log10(grid[j][i]);
                    }
                }
            }
        }
        sstm << outDir << "/currentState" << Windex << ".csv";  //adding index to the string
        currentStateOut = sstm.str();
        sstm.str("");
        printGrid(log10grid, currentStateOut);

        //prepping for parallel section:
        if (requestTh>omp_get_max_threads()) {
            requestTh=omp_get_max_threads();
            omp_set_num_threads(requestTh);
            cerr << "Number of threads requested greater than maximum allowable by the system."
                << endl
                <<  "Setting number of threads to " << omp_get_max_threads() << endl;
            system("read -p \"Press the [Enter] key to continue...\"");
        }
        else if (requestTh>1) {
            omp_set_dynamic(0); // turn off dynamic teams
            omp_set_num_threads(requestTh);
        }
        else {
            cerr << "Invalid thread number request.  Setting number of threads to 2." << endl;
            omp_set_num_threads(2);
            system("read -p \"Press the [Enter] key to continue...\"");
        }
```

```
        sstm << "conditions"  << Windex << ".txt";
        conditions = sstm.str();
        sstm.str("");
        //------
        //parallel section:
        cout <<"Spawning threads..." << endl;
        system(ChangeFont(4));
        #pragma omp parallel shared(nthreads, flagVec, transientExplanation)
        {
            vector < vector <double> > log10gridPrediction;
            vector < vector <double> > gridPrediction;

            double thetaVary, windSpeedVary;
            int th_id;
            th_id = omp_get_thread_num();
            #pragma omp single
            {
                nthreads = omp_get_num_threads();
            }
            if (Windex==1) {
                #pragma omp critical  //restricts the execution of the associated statement
                {
                    //ThDir.push_back(NameDir(th_id));
                    MkThDirPath = ("mkdir " + outDir + "/" + NameDir(th_id)); //making directories
                    system(MkThDirPath.c_str());
                }
                #pragma omp barrier
            }

            //truely parallel portion.

            thetaVary=sampleWind(inDir + "/" + windDataFile, th_id)[0]; //windrose data
            windSpeedVary=sampleWind(inDir + "/" + windDataFile, th_id)[1];
            ofstream fout((outDir + "/" + NameDir(th_id)+ "/" + conditions).c_str());
            fout << "WindSpeed: " << windSpeedVary << endl;
            fout << "Theta (radians): " << thetaVary << endl;
            fout.close();
```

```
            #pragma omp barrier //needed?
            gridPrediction=PlumeProgram(Q, windSpeedVary, hE, thetaVary, z, stabClass, maxY,
                gridResolution, runAheadTime, dt); //runs the plume program
            cout << "Simulating: " << runAheadTime << " hour(s) with a wind coming from the "
                << radianDirTranslator(thetaVary) << " with a speed of " << windSpeedVary
                << " m/s" << endl;
            sstm << "Simulating: " << runAheadTime << " hour(s) with a wind coming from the "
                << radianDirTranslator(thetaVary) << " with a speed of " << windSpeedVary
                << " m/s" << endl;
            transientExplanation[th_id] = sstm.str();
            sstm.str("");
            #pragma omp barrier //needed?
            log10gridPrediction.resize(grid.size(), vector<double> (grid[0].size(), 0));
            for (int j=0; j<grid.size(); j++) {
                for (int i=0; i<grid[0].size(); i++) {
                    gridPrediction[j][i]+=grid[j][i];
                    log10gridPrediction[j][i]=log10(gridPrediction[j][i]);
                }
            }
            #pragma omp barrier //needed?
            if (log10gridPrediction[25][100]>15) {
                flagVec[2][th_id]=1;        //red trip
            } else if (log10gridPrediction[25][100]>5) {
                flagVec[1][th_id]=1;        //yellow trip
            }   else {
                flagVec[0][th_id]=1;        //green
            }
            printGrid(log10gridPrediction, (outDir + "/" + NameDir(th_id) +"/futureState.csv"));
            #pragma omp barrier //needed?
        } // end of parallel portion

        system(ChangeFont(2));
        cout << "Plume program run on " << nthreads << " threads" << endl;
        cout << "Organizing Grid... " << endl;
        system("R CMD BATCH gridOrganizer.R R_data/gridOrganizer.Rout");
        cout << "MeanShift analysis... " << endl;
        clustMembers=MeanShift(Windex, BW, outDir, nthreads);
        cout << "Outputting display..." << endl;
```

```
        system("R CMD BATCH plumeDisplay.R R_data/plumeDisplay.Rout");

        htmlDisplayWriter(outDir, inDir, Windex, flagVec, clustMembers, transientExplanation);

        cout << "Cycle complete!" << endl;
    } // end of (y/n) while loop
} //end of function
#endif
```

C.3. FunctionsEOC.h Souce Code

```
//Created by Kevin Makinson
//1/23/13
//This is a header file to go with Pmain.cpp for RAPSS-EOC
//This contains the majority of the functions used

#ifndef functionsEOC_h
#define functionsEOC_h
#include <iostream>
#include <cmath>
#include <fstream>
#include <vector>
#include <string>
#include <sstream> //for appending strings
#include <stdlib.h> //for system calls in UNIX
#include <time.h>
using namespace std;

//listing of the fuctions
vector<string> LoadFile(string FullFilePath);
vector <vector <string > >  loadSystemData(string sysDataFileName);
vector <vector <double > >  loadGridData(string sysDataFileName);
vector <vector <string > > realTimeSimulator(vector<vector<string > > sysData, int timestep,
    string OutDir);
double windDirTranslator(string windDir);
double sigYFinder(int stabClass, double hE, double x);
double sigZFinder(int stabClass, double hE, double x);
const char *ChangeFont(int ColorCode);
const char *ResetFont();
string NameDir(int th_id);
void printGrid(vector < vector <double> > grid, string outputFileName);
vector<int> SearchVec(vector<string> &text, string key);
vector <vector<string> > loadWindData (string windDataFile);
double windRoseNumTranslator(int windRoseNum);
double *sampleWind(string windDataFile, int th_id);
vector <vector<string> > loadWindObsData (string windDataFile);
```

```
void updateRwindex(int Windex, int time);
int scenClustTranslator(int scenNum, vector<vector<int> > clustMembers);
bool firstClustMember(int scenNum, vector<vector<int> > clustMembers);
string radianDirTranslator(double radians);
double const pi=3.14159;

vector<string> LoadFile(string FullFilePath) {
    string line;
    ifstream fin(FullFilePath.c_str());
    vector<string> text;
    while (getline(fin, line)) {
        text.push_back(line);
    }
    fin.close();
    return text;
}

vector <vector <string > >  loadSystemData(string sysDataFileName) {
    string word;
    vector <string> row;
    vector <vector <string > > sysData;
    vector<string> text;
    text=LoadFile(sysDataFileName);
    for (int j=0; j<text.size(); j++) {
        for (int i=0; i<text[j].size(); i++) {
            word+=text[j][i];
            if (text[j][i]== (',')) {
                row.push_back(word.substr(0, word.size()-1));
                word.clear();
            } else if (i==(text[j].size()-1) ) {
                row.push_back(word.substr(0, word.size())); //last word doesn't have a comma
                word.clear();
            }
        }
        sysData.push_back(row);
        row.clear();
    }
    return sysData;
```

```
}

vector <vector <double > >  loadGridData(string sysDataFileName) {
    string word;
    vector <double> row;
    vector <vector <double > > sysData;
    vector<string> text;
    double number;
    text=LoadFile(sysDataFileName);
    for (int j=0; j<text.size(); j++) {
        for (int i=0; i<text[j].size(); i++) {
            word+=text[j][i];
            if (text[j][i]== (',')) {
                istringstream(word) >> number;
                number=pow(10, number); //unlogging it
                row.push_back(number);
                word.clear();
            } else if (i==(text[j].size()-1) ) {
                istringstream(word) >> number;
                number=pow(10, number);
                row.push_back(number);
                word.clear();
            }
        }
        sysData.push_back(row);
        row.clear();
    }
    return sysData;
}

vector <vector <string > > realTimeSimulator(vector<vector<string > > sysData,
    int startTimeStep, int timestep, string outDir) {
    vector <vector <string > > realTimeData;
    ofstream fout ((outDir + "/realTimeData.csv").c_str());
    fout << "Time,AM/PM,Windspeed (m/s),Direction" << endl;

    for (int j=(startTimeStep-1); j<timestep; j++) {
        for (int i=0; i<sysData[j].size(); i++) {
```

```
            if (i==sysData[j].size()-1) {
                fout << sysData[j][i];
            } else {
            fout << sysData[j][i] << ",";
            }
        }
        fout << endl;
    }
    fout.close();
    //done writing
    //now reading the file just written
    realTimeData=loadSystemData((outDir + "/realTimeData.csv").c_str());
    return realTimeData;
}

//this expects wind "coming from" a direction, not "headed"
double windDirTranslator(string windDir) {
    double theta;
    if(windDir=="W") {
            theta=0;
    } else if (windDir=="WNW") {
        theta=(pi/8)*1;
    } else if (windDir=="NW") {
        theta=(pi/8)*2;
    } else if (windDir=="NNW") {
        theta=(pi/8)*3;
    } else if (windDir=="N") {
        theta=(pi/8)*4;
    } else if (windDir=="NNE") {
        theta=(pi/8)*5;
    } else if (windDir=="NE") {
        theta=(pi/8)*6;
    } else if (windDir=="ENE") {
        theta=(pi/8)*7;
    } else if (windDir=="E") {
        theta=(pi/8)*8;
    } else if (windDir=="ESE") {
        theta=(pi/8)*9;
```

```
    } else if (windDir=="SE") {
        theta=(pi/8)*10;
    } else if (windDir=="SSE") {
        theta=(pi/8)*11;
    } else if (windDir=="S") {
        theta=(pi/8)*12;
    } else if (windDir=="SSW") {
        theta=(pi/8)*13;
    } else if (windDir=="SW") {
        theta=(pi/8)*14;
    } else if (windDir=="WSW") {
        theta=(pi/8)*15;
    }
    return theta;
}

double sigYFinder(int stabClass, double hE, double x) {
    double pY, qY, pZ, qZ, pX, qX, sigY, sigZ, sigX;
    static double ret[3];
    qX=qY=qZ=0.92;
    switch (stabClass) {
        case 1: //unstable
        case 2:
            pX=pY=0.14;
            pZ=0.53;
            break;
        case 3: //stable
        case 4:
            pX=pY=0.06;
            pZ=0.15;
            break;
        case 5: //stable
        case 6:
            pX=pY=0.02;
            pZ=0.04;
            break;
    }
    sigY=pY*pow(x,qY);
```

```
    sigX=pX*pow(x,qX);
    return sigY;
}

double sigZFinder(int stabClass, double hE, double x) {
    double pY, qY, pZ, qZ, pX, qX, sigY, sigZ, sigX;
    static double ret[3];
    qX=qY=qZ=0.92;
    switch (stabClass) {
        case 1: //unstable
        case 2:
            pX=pY=0.14;
            pZ=0.53;
            break;
        case 3: //stable
        case 4:
            pX=pY=0.06;
            pZ=0.15;
            break;
        case 5: //stable
        case 6:
            pX=pY=0.02;
            pZ=0.04;
            break;
    }
    return sigZ;
}

const char *ChangeFont(int ColorCode) {
    string FullFilePath = "ChangeFont.sh";
    ofstream fout (FullFilePath.c_str());
    fout << "tput setf " << ColorCode << endl << "tput bold" << endl << "exit 0";
    string chmod = ("chmod +x " + FullFilePath);
    system(chmod.c_str());                          //creating executable
    fout.close();
    return(FullFilePath.c_str());
}
```

```
const char *ResetFont() {
    string FullFilePath = "ResetFont.sh";
    ofstream fout (FullFilePath.c_str());
    fout << "tput sgr0" << endl << "exit 0" << endl;
    string chmod = ("chmod +x " + FullFilePath);
    system(chmod.c_str());                              //creating executable
    fout.close();
    return(FullFilePath.c_str());
}

//makes a directory based on the thread ID, and returns a string of directory name
string NameDir(int th_id) {
    stringstream sstm;
    sstm << "Th_"  << th_id << "_data";
    string ThDir = sstm.str();
    sstm.str("");
    return (ThDir);
}

void printGrid(vector < vector <double> > grid, string outputFileName) {
    ofstream fout(outputFileName.c_str());
    int gridSize2=grid.size();
    for (int j=0; j<(gridSize2); j++) {
            for (int i=0; i<(gridSize2); i+=1) {
                if (i==((gridSize2)-1)) {
                    fout << grid[j][i] << endl;
                } else {
                    fout << grid[j][i] << " , ";
                }
            }
        }
        fout.close();
    }

void initR(string Rrepos, string libloc, string OutDir, int thNum, int Windex, int runAheadTime,
    int gridResolution, int maxY, double Q) {
    string MkLibDir = ("mkdir -p " + libloc);
    string MkRdataDir = ("mkdir -p R_data"); //added 8/13/12
```

```
    string RinpPath= ("R_data/initR.r");
    system(MkLibDir.c_str());
    system(MkRdataDir.c_str());
    ofstream fout (RinpPath.c_str());
    ifstream fin((libloc+ "/abind").c_str());
    //comments section of input file
    fout << "#!/usr/bin/Rscript" << endl;
    fout << "#" << string(3, ' ') << __DATE__ << endl;
    fout << "#" << string(3, ' ') << "Written by Kevin Makinson" << endl;
    fout << "#" << string(3, ' ')
        <<"This file loads the libraries and initial parameters in R"<<endl;
    fout << "#\n#\n#" << string(70, '-') << endl; //end comments
    fout << "rm(list=ls())" << endl << endl;
    fout << "Rrepos<-\"" << Rrepos << "\"" << endl << "libloc<-\"" << libloc << "\"" << endl;
    fout << "IODir<-\"" << OutDir <<"\"" << endl << "libloc<-\"" << libloc <<"\"" << endl;
    fout << "thNum<- " << thNum << endl;
    fout << "rstNum<-" << Windex << endl;
    fout << "runAheadTime<- " << runAheadTime << endl;
    fout << "gridResolution<- " << gridResolution << endl;
    fout << "maxY<- " << maxY << endl;
    fout << "releaseAmt<- " << Q << endl;
    if(!fin.good()) { //don't install if already installed
        fout << "install.packages(\"abind\", repos=Rrepos, lib=libloc)" << endl;
        fin.close();
    }

    fout << "save.image(\"R_data/RAPSspace.RData\")" << endl;
    fin.close();
    fout.close();
}

//this updates R with each cycle
void updateRwindex(int Windex, int time) {
    //int counter=0;
    string file= ("R_data/updateRwindex.r");
    ofstream fout (file.c_str());
    fout << "load(\"R_data/RAPSspace.RData\")" << endl;
    fout << "rstNum<-" << Windex << endl;
```

```
    fout << "currentTime<-" << time << endl;
    fout << "save.image(\"R_data/RAPSspace.RData\")" << endl;
    fout.close();
}

//this one needs ot be included.
//Serch function returns a vector with the line numbers of where the key is
vector<int> SearchVec(vector<string> &text, string key) {
    //returns a vector of the line numbers of the search term.
    vector<int> LineNums;
    size_t found;
    bool FoundOne = false;
    int size = text.size();
    for (int i=0; i<size; i++) {
        found=text[i].find(key);
        if (found!=string::npos) {
            LineNums.push_back(i);
            FoundOne = true;
        }
        else if (FoundOne == false && i==(size-1)) {
            //Since there is no line "0" this will signify an error
            LineNums.push_back(0);
        }
    }
    return LineNums;
}

//Trims white space around words
string TrimSpace(string MyString) {
    string whitespaces (" \t\f\v\n\r");
    size_t endpos = MyString.find_last_not_of(whitespaces);
    size_t startpos = MyString.find_first_not_of(whitespaces);
    if(string::npos != endpos)
        MyString = MyString.substr(0, endpos+1);
    else
        MyString.clear();            // if string is all whitespace

    if(string::npos != startpos)
```

```
        MyString = MyString.substr(startpos);
    else
        MyString.clear();            // if string is all whitespace
    return MyString;
    }



vector <vector<string> > loadWindData (string windDataFile) {
    vector<string> text;
    text=LoadFile(windDataFile.c_str());
    vector <string> row;
    vector <vector <string > > windData;
    //int count=0;
    string word;
    int startLine, endLine;

    if (SearchVec(text, "Range").back()==0) {
        cerr << "No valid wind data.  Please see http://www.raws.dri.edu/index.html for data"
            << endl;
    } else {
        startLine=SearchVec(text, "Range").back();
        endLine=SearchVec(text, "Calm").back();
        word.clear();
        for (int j=startLine; j<endLine; j++) { //line nums
            for (int i=0; i<text[j].length(); i++) { //character nums
                word += text[j][i];
                if ((text[j][i] == ' ') || (i==(text[j].length()-1))) {
                    if ((word!=" ") && (word!="  ") && (word!="   ") && (word!="    ")
                        && (word!="     ")) {
                        word=TrimSpace(word);
                        row.push_back(word);
                        word.clear();
                    }
                }

            }
            windData.push_back(row);
```

```
            row.clear();
        }
    }
    return windData;
    }

//this changes it from how the windrose does it (N is 0)
//To how the rest of the program does it (E is 0)
double windRoseNumTranslator(int windRoseNum) {
    double theta;
    double pi=3.14159;

    switch(windRoseNum) {
        case 0: // N
            theta=(pi/8)*4;
            break;
        case 1: // NNE
            theta=(pi/8)*5;
            break;
        case 2: // NE
            theta=(pi/8)*6;
            break;
        case 3: // ENE
            theta=(pi/8)*7;
            break;
        case 4: // E
            theta=(pi/8)*8;
            break;
        case 5: // SE
            theta=(pi/8)*9;
            break;
        case 6: // SSE
            theta=(pi/8)*10;
            break;
        case 7: // SSE
            theta=(pi/8)*11;
            break;
        case 8: // S
```

```
            theta=(pi/8)*12;
            break;
        case 9: // SSW
            theta=(pi/8)*13;
            break;
        case 10: // SW
            theta=(pi/8)*14;
            break;
        case 11: // WSW
            theta=(pi/8)*15;
            break;
        case 12: // W
            theta=(pi/8)*16;
            break;
        case 13: // WNW
            theta=(pi/8)*1;
            break;
        case 14: // NW
            theta=(pi/8)*2;
            break;
        case 15: // NNW
            theta=(pi/8)*3;
            break;
    }

    return theta;
}

double *sampleWind(string windDataFile, int th_id) {
    static double ret[2];
    int bins;
    int directions;
    int currentDirection;
    double windDirMaxProb=0;
    double windSpeed, temp1, temp2;
    vector <vector <string > > windData;
    windData=loadWindData(windDataFile);
    bins=(windData.size()-4);
```

```
    directions=(windData[1].size()-1);
    vector< vector<double> > windSpeedProb;
    vector<double> windDirProb (directions, 0);
    vector<double> CDFwindDir (directions, 0);
    vector< vector<double> > CDFwindSpeedProb;
    windSpeedProb.resize(bins, vector<double> (directions, 0));
    CDFwindSpeedProb.resize(bins, vector<double> (directions, 0));
    double U1, U2;
    srand(time(NULL)*932174973*th_id+1);
    U1=((double)(rand())/((double)RAND_MAX));
    srand(time(NULL)*534041066*th_id+1); //multiplied by OSU student ID
    U2=((double)(rand())/((double)RAND_MAX));

    for (int i=1; i<=directions; i++) { //start at 1 instead of 0 bc of "Total(%)" is the 0 word
        istringstream(string(windData[bins+3][i].begin(), windData[bins+3][i].end()))
            >> windDirProb[i-1];
        windDirMaxProb+=windDirProb[i-1];
    }
    for (int i=0; i<directions; i++) {
        for (int j=0; j<bins; j++) {
            istringstream(string(windData[j+3][i+3].begin(), windData[j+3][i+3].end()))
                >> windSpeedProb[j][i];
            if (windDirProb[i]!=0) { //to avoid dividing by zero
                windSpeedProb[j][i] = (windSpeedProb[j][i] / windDirProb[i]);
            }
            if(j==0) {
                CDFwindSpeedProb[j][i] = windSpeedProb[j][i];
            } else {
                if (CDFwindSpeedProb[j-1][i] + windSpeedProb[j][i]>1) {
                    CDFwindSpeedProb[j][i]=1; //to correct for rounding errors
                } else {
                    CDFwindSpeedProb[j][i] = CDFwindSpeedProb[j-1][i] + windSpeedProb[j][i];
                }
            }
        }
    }
    //creating a CDF for windDirection
    for (int i=0; i<directions; i++) {
```

```
        windDirProb[i] = (windDirProb[i]/windDirMaxProb);
        if(i==0) {
                CDFwindDir[i] = windDirProb[i];
            } else {
                if ((CDFwindDir[i] + windDirProb[i])>1) {
                    CDFwindDir[i]=1; //to correct for rounding errors
                } else {
                    CDFwindDir[i] = CDFwindDir[i-1] + windDirProb[i];
                }
            }
        if ( (U2<=CDFwindDir[i]) && (U2>(CDFwindDir[i]-windDirProb[i]) )) {
            currentDirection=i;
        }
    }

    for (int j=0; j<bins; j++) {
        if ( (U1<=CDFwindSpeedProb[j][currentDirection]) &&
            (U1>(CDFwindSpeedProb[j][currentDirection]-windSpeedProb[j][currentDirection])) ) {
            istringstream(string(windData[j+3][0].begin(), windData[j+3][0].end())) >> temp1;
            istringstream(string(windData[j+3][2].begin(), windData[j+3][2].end())) >> temp2;
            windSpeed=(temp1+temp2)/2;
        }
    }
    ret[0]=windRoseNumTranslator(currentDirection); //theta
    ret[1]=windSpeed;
    return ret;
}

vector <vector<string> > loadWindObsData (string windDataFile) {
    int startLine, endLine;
    vector<string> text;
    text=LoadFile(windDataFile);
    vector <string> row;
    vector <vector <string > > windObsData;
    vector <vector <string > > windObsDataOutput;
    string word;

    if (SearchVec(text, "Hour").back()==0) {
```

```
        cerr << "No valid wind observation data.";
        cerr << "Please see http://www.raws.dri.edu/index.html" << endl;
    } else {
        startLine=SearchVec(text, "Hour").back();
        endLine=SearchVec(text, "DAILY STATISTICS").back();
        word.clear();
        for (int j=startLine; j<endLine; j++) { //line nums
            for (int i=0; i<text[j].length(); i++) { //character nums
                word += text[j][i];
                if ((text[j][i] == ' ') || (i==(text[j].length()-1))) {
                    if ((word!=" \t") && (word!=" \t\t") && (word!=" \t\t\t")
                        && (word!=" \t\t\t\t") && (word!=" \t\t\t\t\t")
                        &&  (word!="\t") && (word!="\t\t") && (word!="\t\t\t")) {
                        word=TrimSpace(word);
                        row.push_back(word);
                        word.clear();
                    }
                }

            }
            windObsData.push_back(row);
            row.clear();
        }
    }

    //Outputting only wind direction, speed, and time
    row.clear();
    for (int j=4; j<windObsData.size(); j++) { //line nums
        for (int i=0; i<5; i++) { //col nums
            if (i!=2) {
            row.push_back(windObsData[j][i]);
            }
        }
        windObsDataOutput.push_back(row);
        row.clear();
    }

    return windObsDataOutput;
```

```
    }

int scenClustTranslator(int scenNum, vector<vector<int> > clustMembers) {
    int clustNum;
    for (int j=0; j<clustMembers.size(); j++) {
        for (int k=0; k<clustMembers[j].size(); k++) {
            if (clustMembers[j][k]==scenNum) {
                clustNum=j;
            }
        }
    }
    return (clustNum+1);
}
//returns true only if the scenario is the first member of a cluster
bool firstClustMember(int scenNum, vector<vector<int> > clustMembers) {
    bool first=false;
    for (int j=0; j<clustMembers.size(); j++) {
        if (clustMembers[j][0]==scenNum) {
            first=true;
        }
    }
    return first;
}

string radianDirTranslator(double radians) {
    string windDir;

    if ((radians > (pi/16)*15) && (radians<=(pi/16)*17)) {
        windDir="W";  //pi
    } else if (((radians>(pi/16)*17) && (radians<=(pi/16)*19)) ||
    ((radians>(pi/16)*17+2*pi) && (radians<=(pi/16)*19+2*pi))) {
        windDir="WSW";
    } else if (((radians>(pi/16)*19) && (radians<=(pi/16)*21)) ||
    ((radians>(pi/16)*19+2*pi) && (radians<=(pi/16)*21+2*pi))) {
        windDir="SW";
    } else if (((radians>(pi/16)*21) && (radians<=(pi/16)*23)) ||
    ((radians>(pi/16)*21+2*pi) && (radians<=(pi/16)*23+2*pi))) {
        windDir="SSW";
```

```
    } else if (((radians>(pi/16)*23) && (radians<=(pi/16)*25)) ||
    ((radians>(pi/16)*23+2*pi) && (radians<=(pi/16)*25+2*pi))) {
        windDir="S";
    } else if (((radians>(pi/16)*25) && (radians<=(pi/16)*27)) ||
    ((radians>(pi/16)*25+2*pi) && (radians<=(pi/16)*27+2*pi))) {
        windDir="SSE";
    } else if (((radians>(pi/16)*27) && (radians<=(pi/16)*29)) ||
    ((radians>(pi/16)*27+2*pi) && (radians<=(pi/16)*29+2*pi))) {
        windDir="SE";
    } else if (((radians>(pi/16)*29) && (radians<=(pi/16)*31)) ||
    ((radians>(pi/16)*29+2*pi) && (radians<=(pi/16)*31+2*pi))) {
        windDir="ESE";
    } else if (((radians>(pi/16)*31) && (radians<=(pi/16)*33)) ||
    ((radians>(pi/16)*31+2*pi) && (radians<=(pi/16)*33+2*pi))) {
        windDir="E";
    } else if (((radians>(pi/16)*1) && (radians<=(pi/16)*3)) ||
    ((radians>(pi/16)*1+2*pi) && (radians<=(pi/16)*3+2*pi))) {
        windDir="ENE";
    } else if (((radians>(pi/16)*3) && (radians<=(pi/16)*5)) ||
    ((radians>(pi/16)*3+2*pi) && (radians<=(pi/16)*5+2*pi))) {
        windDir="NE";
    } else if (((radians>(pi/16)*5) && (radians<=(pi/16)*7)) ||
    ((radians>(pi/16)*5+2*pi) && (radians<=(pi/16)*7+2*pi))) {
        windDir="NNE";
    } else if (((radians>(pi/16)*7) && (radians<=(pi/16)*9)) ||
    ((radians>(pi/16)*7+2*pi) && (radians<=(pi/16)*9+2*pi))) {
        windDir="N";
    } else if (((radians>(pi/16)*9) && (radians<=(pi/16)*11)) ||
    ((radians>(pi/16)*9+2*pi) && (radians<=(pi/16)*11+2*pi))) {
        windDir="NNW";
    } else if (((radians>(pi/16)*11) && (radians<=(pi/16)*13)) ||
    ((radians>(pi/16)*11+2*pi) && (radians<=(pi/16)*13+2*pi))) {
        windDir="NW";
    } else if (((radians>(pi/16)*13) && (radians<=(pi/16)*15)) ||
    ((radians>(pi/16)*13+2*pi) && (radians<=(pi/16)*15+2*pi))) {
        windDir="WNW";
    }
    return windDir;
```

```
}

void htmlDisplayWriter(string OutDir, string InDir, double RstNum,
    vector <vector <int> > flagVec, vector<vector<int> > clustMembers,
    vector<string> transientExplanation) {
    string displayOutFilePath;
    string displayOutFile;
    string greenOutFile;
    string greenOutFilePath;
    string unstableOutFile;
    string unstableOutFilePath;
    string miscOutFile;
    string miscOutFilePath;
    int redTableCols=0;
    int yellowTableCols=0;

    for (int i=0; i<flagVec[2].size(); i++) {
        if (flagVec[2][i]==1) {
            redTableCols++;
        }
    }
    if (redTableCols==0) {
        redTableCols=1;
    }

    for (int i=0; i<flagVec[1].size(); i++) {
        if (flagVec[1][i]==1) { //double check!
            yellowTableCols++;
        }
    }
    if (yellowTableCols==0) {
        yellowTableCols=1;
    }

    stringstream sstm;
    sstm << "DISPLAY" << RstNum << ".html"; //adding index to the string
    displayOutFile = sstm.str();
    sstm.str("");
```

```
displayOutFilePath = (OutDir + "/" + displayOutFile);

sstm << "green" << RstNum << ".html";   //adding index to the string
greenOutFile = sstm.str();
sstm.str("");
greenOutFilePath = (OutDir + "/" + greenOutFile);

sstm << "unstable" << RstNum << ".html";    //adding index to the string
unstableOutFile = sstm.str();
sstm.str("");
unstableOutFilePath = (OutDir + "/" + unstableOutFile);

sstm << "misc" << RstNum << ".html";    //adding index to the string
miscOutFile = sstm.str();
sstm.str("");
miscOutFilePath = (OutDir + "/" + miscOutFile);


bool output=false;
ofstream fout(displayOutFilePath.c_str());
fout << "<!--Display Engine For RAPSS-EOC - Written by Kevin Makinson-->" << endl;
fout << "<html><head>" << endl;
fout << "<meta content=\"text/html; charset=ISO-8859-1\"";
fout << "http-equiv=\"content-type\"><title>RAPSS-EOC Display</title>" << endl;
fout << "<link rel=\"stylesheet\" type=\"text/css\" href=\"tswtabs.css\">" << endl;
fout << "</head>" << endl;
fout << "<body>" << endl;
fout << "<strong><font size=\"+2\">RAPSS-EOC Output Restart " << RstNum
    << "</font><br>" << endl;
fout << "<br>" << endl;
fout << "<span style=\"text-decoration: underline;\">City X in danger</span></strong><p><br>"
    << endl;
fout << "<table style=\"border-color: rgb(255, 0, 0); text-align: left; width: "
    << 500*redTableCols <<"px;\" border=\"10\" cellpadding=\"2\" cellspacing=\"2\">"<< endl;
fout << "<tbody>"<< endl;
fout << "<tr>"<< endl;
for (unsigned int i=0; i<flagVec[2].size(); i++) {
    if ((flagVec[2][i]==1) && (firstClustMember(i, clustMembers))) {
```

```
            //red threshold logic
            //scenario i has red threshold reached
            output=true;
            fout << "<td>" << endl;
            fout << "<ul style=\"color: red;\">" << endl;
            fout << "<img style=\"width: 164px; height: 41px;\" alt=\"\" src=\"Alert.gif\"><br>";
            fout << "<ul style=\"color: red;\">" << endl;
            fout << "<li><a href=\"plumeRst" << RstNum <<"Cl"
                << scenClustTranslator(i, clustMembers) << ".pdf\">Cluster "
                << scenClustTranslator(i, clustMembers) << " Plots</a></li>" << endl;
            fout << "<ul style=\"color: red;\">" << endl;
            fout << "<li>" << transientExplanation[i]  << " </li>" << endl;
            fout << "</ul>" << endl;
            fout << "</ul>" << endl;
            fout << "</td>" << endl;
        } else if (i==(flagVec[2].size()-1) && (output==false)) {
            fout << "<td>" << endl;
            fout << "<li> No red trips </li>" << endl;
            fout << "</td>" << endl;
            fout << "</u1>" << endl;
        }
    }
    fout << "</td>" << endl;
    fout << "</tr>" << endl;
    fout << "</tbody>" << endl;
    fout << "</table>" << endl;
    fout << "</li></li></ul></ul><p>" << endl;
    output=false;


    fout << "<strong><span style=\"text-decoration: underline;\">City X Possibly in Danger";
    fout << "</span></strong><p><br>" << endl;
    //change this one to be similar to the one above
    fout << "<table style=\"border-color: rgb(255, 180, 0); text-align: left; width: "
        << 500*yellowTableCols <<"px;\" border=\"10\" cellpadding=\"2\" cellspacing=\"2\">"
        << endl;
    fout << "<tbody>" << endl;
    fout << "<tr>" << endl;
```

```
    for (unsigned int i=0; i<flagVec[1].size(); i++) {
        if ((flagVec[1][i]==1) && (firstClustMember(i, clustMembers))) {
            //red threshold logic
            output=true;
            fout << "<td>" << endl;
            fout << "<ul style=\"color: rgb(255, 180, 0);\">" << endl;
            fout << "<li><a href=\"plumeRst" << RstNum <<"Cl"
                << scenClustTranslator(i, clustMembers) << ".pdf\">Cluster "
                << scenClustTranslator(i, clustMembers) << " Plots</a></li>" << endl;
            fout << "<ul style=\"color: rgb(255, 180, 0);\">" << endl;
            fout << "<li>" << transientExplanation[i]  << " </li>" << endl;
            fout << "</ul>" << endl;
            fout << "</ul>" << endl;
            fout << "</td>" << endl;
        } else if (i==(flagVec[1].size()-1) && (output==false)) {
            fout << "<td>" << endl;
            fout << "<li> No yellow thresholds exceeded </li>" << endl;
            fout << "</td>" << endl;
            fout << "</u1>" << endl;
        }
    }
    fout << "</td>" << endl;
    fout << "</tr>" << endl;
    fout << "</tbody>" << endl;
    fout << "</table>" << endl;
    fout << "</li></li></ul></ul><p>" << endl;
    output=false;

    fout << "<p><br>" << endl;
    fout << "</span><br><div id=\"tswcsstabs\">" << endl;
    fout << "<ul>" << endl;
    fout << "<li><a href=\"green" << RstNum
        << ".html\">No Thresholds Tripped</a></li>" << endl;
    fout << "<li><a href=\"unstable" << RstNum
        << ".html\">Model Became Unstable</a></li>" << endl;
    fout << "<li><a href=\"misc"
        << RstNum<< ".html\">Miscellaneous Information</a></li>" << endl;
    fout << "</ul>" << endl;
```

```
    fout << "</div>" << endl;
    fout << "</body></html>" << endl;
    fout.close();
    fout.clear();
    //this ends the main page.

    fout.open(greenOutFilePath.c_str());
    fout << "<!--Display Engine For RAPSS-EOC - Written by Kevin Makinson-->" << endl;
    fout << "<html><head>" << endl;
    fout << "<meta content=\"text/html; charset=ISO-8859-1\" http-equiv=\"content-type\">";
    fout << "<title>RAPSS-EOC No Thresholds</title>" << endl;
    fout << "</head>" << endl;
    fout << "<body>" << endl;
    fout << "<strong><font size=\"+2\">RAPSS-EOC Output Restart " << RstNum
        << "</font><br>" << endl;
    fout << "<br>" << endl;
    fout << "<span style=\"text-decoration: underline;\">";
    fout << "City Not In Danger</span></strong><br>" << endl;
    fout << "<ul style=\"color: rgb(0, 153, 0);\">" << endl;
    for (unsigned int i=0; i<flagVec[0].size(); i++) {
        if ((flagVec[0][i]==1) && (firstClustMember(i, clustMembers))) {
            output=true;
            fout << "<td>" << endl;
            fout << "<ul style=\"color: green;\">" << endl;
            fout << "<li><a href=\"plumeRst" << RstNum <<"Cl"
                << scenClustTranslator(i, clustMembers) << ".pdf\">Cluster "
                << scenClustTranslator(i, clustMembers) << " Plots</a></li>" << endl;
            fout << "<ul>" << endl;
            fout << "<li>" << transientExplanation[i]  << " </li>" << endl;
            fout << "</ul>" << endl;
            fout << "</ul>" << endl;
            fout << "</td>" << endl;
        } else if (i==(flagVec[2].size()-1) && (output==false)) {
            fout << "<td>" << endl;
            fout << "<li> Everything's in danger! Run away! </li>" << endl;
            fout << "</td>" << endl;
            fout << "</u1>" << endl;
        }
```

```
    }
    fout << "</td>" << endl;
    fout << "</tr>" << endl;
    fout << "</tbody>" << endl;
    fout << "</table>" << endl;
    fout << "</li></li></ul></ul><p>" << endl;
    output=false;
    fout.close();
    fout.clear();

    fout.open(miscOutFilePath.c_str());
    fout << "<html><head>" << endl;
    fout << "<meta content=\"text/html; charset=ISO-8859-1\" http-equiv=\"content-type\">";
    fout << "<title>RAPSS-EOC Cluster Information</title>" << endl;
    fout << "</head>" << endl;
    fout << "<body>" << endl;
    fout << "<strong><font size=\"+2\">RAPSS-EOC Output Restart " << RstNum << "</font><br>"
        << endl;
    fout << "<br>" << endl;
    fout << "<span style=\"text-decoration: underline;\"></span>";
    fout << "<span style=\"text-decoration: underline;\">Cluster Information</strong><br>"
        << endl;
    fout << "</span>" << endl;
    for (unsigned int j=0; j<clustMembers.size(); j++) { //j is cluster number
        fout << "<ul>" << endl;
        fout << "<li>Cluster " << j+1 << " Plot Members</li>" << endl;
        fout << "<ul>" << endl;
        for (unsigned int i=0; i<clustMembers[j].size(); i++) { //i is scenario number
            //fout << "<ul>" << endl;
            fout << "<li>Scenario " << clustMembers[j][i] << "</li>" << endl;
        }
        fout << "</ul>" << endl;
        fout << "</ul>" << endl;
    }
    fout << "</body></html>" << endl;
    fout.clear();
    fout.close();

```

```
    fout.open(unstableOutFilePath.c_str());
    fout << "<!--Display Engine For RAPSS-EOC - Written by Kevin Makinson-->" << endl;
    fout << "<html><head>" << endl;
    fout << "<meta content=\"text/html; charset=ISO-8859-1\" http-equiv=\"content-type\">";
    fout << "<title>RAPSS-EOC Unstable Scenarios</title>" << endl;
    fout << "</head>" << endl;
    fout << "<body>" << endl;
    fout << "<strong><font size=\"+2\">RAPSS-EOC Output Restart " << RstNum
        << "</font><br>" << endl;
    fout << "<p><span style=\"text-decoration: underline;\">";
    fout << "Model Became Unstable</span></strong><br>" << endl;
    fout << "<li> No model instabilities on this cycle </li>" << endl;
    fout << "</body></html>" << endl;
    fout.close();
    fout.clear();
}
#endif
```

C.4. PlumeProgram.h Source Code

```
//Written by Kevin Makinson
//1/24/13
//This simulates a integrated puff model

#ifndef plumeProgram_h
#define plumeProgram_h
#include <fstream>
#include <math.h>

vector < vector <double> > PlumeProgram(double Q, double u, double hE, double theta, double z,
 int stabClass, int maxY, int gridResolution, int timeInterval, int dt) {
     timeInterval=timeInterval*3600;  //converts hours to seconds
     //Q is release activity
     //sigY, sigZ, sigX are std dev of lateral, vertical, and horizontal diffusion
     double sigY, sigZ, sigX, Xconc;
     double x, y, Xrot, Yrot;
     theta=(theta+pi); //changes to where wind is going (plume equation)
     //this converts it from how wind directions are normally expressed, to how the
     //plume program expects it.
     vector <double> row;
     vector < vector <double> > grid;
     int maxSquares;
     maxSquares=(maxY/gridResolution);

     for (int t=1; t<=(timeInterval/dt); t++) {
         for (int j=(maxSquares*-1); j<maxSquares; j+=1) {
             y=(j*gridResolution);
             for (int i=(maxSquares*-1); i<maxSquares; i+=1) {
                 x=(i*gridResolution);
                 //mapping x and y to rotated axes
                 Xrot=x*cos(theta)-y*sin(theta);
                 Yrot=x*sin(theta)+y*cos(theta);
                 if (Xrot<=0) { //Stuff behind the plume, just put in 0.0001
                     Xconc=(0.0001);
                 } else {
```

```cpp
                     sigY=sigX=sigYFinder(stabClass, hE, Xrot);
                     sigZ=sigZFinder(stabClass, hE, Xrot); //not needed for ground level
                     //for cartesian coordinates (puff)
                     Xconc= (dt*(Q/(pow(2*pi, 1.5)*sigY*sigZ*sigX)) *
                                 exp(-0.5*pow((Xrot-u*t*dt)/sigX, 2)) *
                                 (exp(-0.5*(pow(z-hE,2)/(pow(sigZ,2)))) +
                                 exp(-0.5*(pow(z+hE,2)/(pow(sigZ,2))))) *
                                 exp(-0.5*pow(Yrot/sigY,2)));
                     if (Xconc<(0.001)) {
                         Xconc=(0.0001);
                     }
                 }
                 if (t==1) {
                     if (i==(maxSquares-1)) {
                         row.push_back(Xconc);
                         grid.push_back(row);
                         row.clear();
                     } else {
                         row.push_back(Xconc);
                     }
                 } else {
                     grid[j+maxSquares][i+maxSquares]+=Xconc;
                 }
             }
         }
     }
 return grid;
}
#endif
```

C.5. GridOrganizer.R Source Code

```r
#!/usr/bin/Rscript
# 1/31/13
# Written by Kevin Makinson
# Oregon State university
#
# This code takes the grid structure from RAPSS-EOC and
# turns it into something MSA can use.
# --------------------------------#
load("R_data/RAPSspace.RData")
library(abind, lib.loc=libloc)
EOCOutFilePaths<-array(0,thNum)
#assigning file paths
#reading data
#plopping it into a 3D matrix
#need this!
kCount=0
for (i in 0:(thNum-1)) {
  kCount=kCount+1
  EOCOutFilePaths[kCount]<-paste(IODir,"/Th_", i, "_data/futureState.csv", sep = "")
  EOCOutRawData<- read.csv(EOCOutFilePaths[kCount], header=FALSE) #change from 1
  if(kCount==1) {
    EOCOutRawDataC<-EOCOutRawData
  } else if ((dim(EOCOutRawDataC)[1])==(dim(EOCOutRawData)[1])) {
    EOCOutRawDataC<-abind(EOCOutRawDataC, EOCOutRawData, along=3)
  } else if ((dim(EOCOutRawDataC)[1]) > (dim(EOCOutRawData)[1])) {
    EOCOutRawDataC<-abind(EOCOutRawDataC[1:(dim(EOCOutRawData)[1]),,], EOCOutRawData, along=3)
  } else {
    EOCOutRawDataC<-abind(EOCOutRawDataC, EOCOutRawData[1:(dim(EOCOutRawDataC)[1]),], along=3)
  }
}

MeanShiftReady<-array(0, c(thNum, dim(EOCOutRawDataC)[1]*dim(EOCOutRawDataC)[2]))

#complicated part goes here.
for (j in 1:thNum) {
```

```r
  for (i in 1:(dim(EOCOutRawDataC)[1])) {
    MeanShiftReady[j,((dim(EOCOutRawDataC)[1]*(i-1))+1):(dim(EOCOutRawDataC)[1]*i)]<-
      (EOCOutRawDataC[i,,j])
  }
}

write.table(MeanShiftReady, file=(paste(IODir, "/meanShiftReady", rstNum, ".csv", sep="")),
 row.names = FALSE, col.names=FALSE, sep=",")

save.image("R_data/RAPSspace.RData")
```

C.6. PlumeDisplay.R Source Code

```
#!/usr/bin/Rscript
# --------
# Written by Kevin Makinson
# 1/23/13
# This is designed for RAPSS-EOC to produce the approperiate plots
# --------

load("R_data/RAPSspace.RData")
library(abind, lib.loc=libloc)
maxSquares<-maxY/gridResolution
x<- -maxSquares:(maxSquares-1)
y<- -maxSquares:(maxSquares-1)
outer.radius = maxSquares
breaks = seq(-2, log10(releaseAmt), by = 2)
contour<-F #this should come from the user
#--------

clustCenter<-read.csv(paste(IODir, "/clustCenters", rstNum, ".csv", sep=""), header=FALSE)
if (dim(clustCenter)[2]<2) {
  unMSA<-array(0, c(dim(EOCOutRawDataC)[1], dim(EOCOutRawDataC)[2]))
} else {
  unMSA<-array(0, c(dim(EOCOutRawDataC)[1], dim(EOCOutRawDataC)[2], dim(clustCenter)[2]))
}

for (j in 1:dim(clustCenter)[2]) {
  for (i in 1:(dim(EOCOutRawDataC)[1])) {
    if (dim(clustCenter)[2]<2) {
      unMSA[i,]<-as.matrix(clustCenter[((dim(EOCOutRawDataC)[1]
        *(i-1))+1):(dim(EOCOutRawDataC)[1]*i),1])
    } else {
      unMSA[i,,j]<-as.matrix(clustCenter[((dim(EOCOutRawDataC)[1]
        *(i-1))+1):(dim(EOCOutRawDataC)[1]*i),j])
    }
  }
}
```

```
#current state
z<-(read.csv(paste(IODir, "/currentState", rstNum, ".csv", sep=""), header=FALSE))

for (j in (1:dim(clustCenter)[2])) {
  if (dim(clustCenter)[2]<2) {
    z<-abind(z, (unMSA[,]), along=3)
  } else {
    z<-abind(z, (unMSA[,,j]), along=3)
  }
}

#this loop fixes a problem much later involving the inverting of the y variables.
ztemp<-array(0, c(dim(z)[1], dim(z)[2], dim(z)[3]))
for (j in 1:dim(z)[3]) {
  for (i in 1:dim(z)[2]) {
    ztemp[i,,j]<-z[(dim(z)[2]-i+1),,j]
  }
}
z<-ztemp

#define a color pallette,
rgb.palette<-colorRampPalette(c("gray25","red", "yellow", "green", "gray85", "white"),
    space="rgb")
col <- rev(rgb.palette(length(breaks) - 1))

nlevels = length(breaks)-1
contours = TRUE
legend = TRUE
axes = TRUE
circle.rads = pretty(c(0,outer.radius))

#gererating plot
for (k in 2:dim(z)[3]) {
  pdf(paste(IODir, "/plumeRst", rstNum, "Cl", k-1, ".pdf", sep=""),onefile=TRUE)
  par(mai = c(1,1.5,1.5,1.6))
  for (j in 1:2) {
    if (j==2) {
```

```
    j<-k
  }
  if (j==1) {
    image(x = (min(x):max(x)), y = (min(y):max(y)), t(as.matrix(z[,,j])), useRaster = TRUE,
          asp = 1, axes = FALSE, xlab = "", ylab = "", col = col, breaks = breaks,
          main=paste("Current Plume \nat: ", currentTime, ":00", sep="") )
    points(x=0, y=75, pch=19, cex=4)
    text(x=4, y=75, labels="City X", pos=4)
  } else {
    image(x = (min(x):max(x)), y = (min(y):max(y)), t(as.matrix(z[,,j])), useRaster = TRUE,
          asp = 1, axes = FALSE, xlab = "", ylab = "", col = col, breaks = breaks,
          main=paste("Cluster ", j-1, "\n Run-Ahead ", runAheadTime, " hr from: ",
          currentTime, ":00", sep=""))
    points(x=0, y=75, pch=19, cex=4)
    text(x=4, y=75, labels="City X", pos=4)
  }
  # adding contour Lines if user wishes
  if (contour==T) {
    CL <- contourLines(x = (min(x):max(x)), y = (min(y):max(y)),
      t(as.matrix(z)), levels = breaks)
    A <- lapply(CL, function(xy){
      lines(xy$x, xy$y, col = gray(.2), lwd = .5)
    })
  }

  #---------
  RMat <- function(radians){
    matrix(c(cos(radians), sin(radians), -sin(radians), cos(radians)), ncol = 2)
  }

  circle <- function(x, y, rad = 1, nvert = 500){
    rads <- seq(0,2*pi,length.out = nvert)
    xcoords <- cos(rads) * rad + x
    ycoords <- sin(rads) * rad + y
    cbind(xcoords, ycoords)
  }

  # draw circles
```

```
  if (missing(circle.rads)){
    circle.rads <- pretty(c(0,outer.radius))
  }

  for (i in circle.rads){
    lines(circle(0, 0, i), col = "#66666650")
  }

  axis.rads <- c(0, (pi/8), 2*(pi/8), 3*(pi/8), 4*(pi/8), 5*(pi/8), 6*(pi/8), 7*(pi/8))
  r.labs <- c("E", "ENE", "NE", "NNE", "N", "NNW", "NW", "WNW")
  l.labs <- c("W", "WSW", "SW", "SSW", "S", "SSE", "SE", "ESE")

  for (i in 1:length(axis.rads)){
    endpoints <- zapsmall(c(RMat(axis.rads[i]) %*% matrix(c(1, 0, -1, 0)
      * outer.radius,ncol = 2)))
    segments(endpoints[1], endpoints[2], endpoints[3], endpoints[4], col = "#66666650")
    endpoints <- c(RMat(axis.rads[i]) %*% matrix(c(1.1, 0, -1.1, 0) * outer.radius, ncol = 2))
    text(endpoints[1], endpoints[2], r.labs[i], xpd = TRUE)
    text(endpoints[3], endpoints[4], l.labs[i], xpd = TRUE)
  }

  axis(2, pos = -1.24*outer.radius, at = sort(union(circle.rads,-circle.rads)), labels = NA)
  text( -1.25*outer.radius, sort(union(circle.rads, -circle.rads)),
      gridResolution*sort(union(circle.rads, -circle.rads)), xpd = TRUE, pos = 2)

  #label on the Y-axis
  text(x=-1.25*outer.radius, y=-1.25*outer.radius, xpd = TRUE, labels="Distance (m)" )
  ylevs <- seq(-outer.radius, outer.radius, length = nlevels + 1)
  rect(1.2 * outer.radius, ylevs[1:(length(ylevs) - 1)], 1.3 *
      outer.radius, ylevs[2:length(ylevs)], col = col, border = NA, xpd = TRUE)
  #y direction
  rect(1.2 * outer.radius, min(ylevs), 1.3 * outer.radius, max(ylevs),
      border = "#66666650", xpd = TRUE)
  #color scale:
  text(1.3 * outer.radius, ylevs,labels=paste("10^", round(breaks, 1)), pos = 4, xpd = TRUE)
  text(x= (1.3*outer.radius), y=-1.25*(outer.radius),labels="Concentration (Bq/m^2)",
      xpd = TRUE)
}
```

```
dev.off()
}
#---------
save.image("R_data/RAPSspace.RData")
```

C.7. initR.r Source Code

```r
#!/usr/bin/Rscript
#   Mar  7 2013
#   Written by Kevin Makinson
#   This file loads the libraries and initial parameters in R
#
#----------------------------------------------------------------------
rm(list=ls())
Rrepos<-"http://cran.r-project.org"
libloc<-"/nfs/stak/students/m/makinske/lib"
IODir<-"/nfs/stak/students/m/makinske/RAPSS-EOC/RAPSS_data"
libloc<-"/nfs/stak/students/m/makinske/lib"
thNum<- 8
rstNum<-1
runAheadTime<- 3
gridResolution<- 100
maxY<- 10000
releaseAmt<- 1e+30
save.image("R_data/RAPSspace.RData")
```

C.8. updateRwindex.R Source Code

```
load("R_data/RAPSspace.RData")
rstNum<-1
currentTime<-5
save.image("R_data/RAPSspace.RData")
```

C.9. Sample RAPSS-EOC Input File

```
* ===================================
* RAPSS-EOC input file
* Written by Kevin Makinson
* Oregon State University
* ====================================
* Data analysis parameters
* ====================================
* BandWidth for MSA (BW)
101 1
* Path for R library files to be downloaded into (libloc)
102 /nfs/stak/students/m/makinske/lib
* Website for downloading R files (Rrepos)
103 http://cran.r-project.org
* ======================================
* RAPSS parameters
* ======================================
* requested number of threads (requestTh)
201 8
* Working for storing files (inDir)
202 /nfs/stak/students/m/makinske/RAPSS-EOC
* Real time speed up multiplier
203 480
* ======================================
* Plume Program Parameters
* ======================================
* Grid Resolution (gridResolution)
301 100
* Max Y value (grid distance in x and y direction) (maxY)
302 10000
* Stack Height (hE) (meters)
303 20
* Height above ground (z)
304 0
* Release rate (Q) (Bq/s)
305 1e30
* dt (in seconds) (adjust if running into memory issues for long times)
306 60
* timestep advancements for looking ahead (runAheadTime) (hr)
307 3
* What time of day did the plume start? (plumeStartTime)  (e.g., 1 AM) (note: not 1:00
AM)
308 3 AM
* What time of day did the simulation start? (simulationStartTime)  (e.g., 1 AM) (note:
not 1:00 AM)
```

309 5 AM
* Name of wind observation data file from http://www.raws.dri.edu/index.html
310 JuniperDunesWind.txt
* Name of windrose data file from: http://www.raws.dri.edu/index.html
311 JuniperDunesWindObs.txt
*end of file

D. Appendix D: RAPPS-EOC Source Code Explanation

Some sections are very similar to RAPSS-STA. The following explanations primarily pertain to the differences between RAPSS-STA and RAPSS-EOC. Please see Appendix B for explanations of functions used in both RAPSS-EOC and RAPSS-STA.

D.1. Pmain.cpp Source Code Explanation

Pmain.cpp (see Appendix C.1) is the file that "runs" RAPSS-EOC. Lines 12-29 are variable definitions which have been extracted to the top level in order to allow the user to edit these variables via the RAPSS-EOC input file. Line 075 reads the input file defined in RAPSSinputFile() (lines 106-225). Lines 080-085 change 12-hour time to 24-hour time. Finally, lines 93-95 call cyclePlumeProgram(), the function that cycles RAPSS-EOC.

D.2. CyclePlume.h Code Explanation

CyclePlume.h (see Appendix C.2) contains one function definition, cyclePlumeProgram(), which acts as the primary control mechanism for RAPSS-EOC. This file can be considered the second layer below Pmain.cpp. After the local variable definition section (lines 16-46), the user in entered into a while-loop that exists for the rest of the function. This depends on the string variable, *answer*, being "yes," or "no," signifying if the user wishes to perform more cycles in the while-loop. Lines 052-060 copy Alert.gif and tswtabs.css into the same directory that the display will eventually live. Alert.gif is the animated flashing "ALERT" picture in the display. Tswtabs.css is the script for the buttons at the bottom of the display (see Section 9.4). Lines 062-073 ask the user if he or she wishes to run the program, and changes *answer* accordingly. The user is then asked how many cycles he or she wishes to run (lines 085-092). This causes

RAPSS-EOC to cycle in the while loop for a user defined number of cycles until asking if he or she wishes to continue. Real time data are sampled at line 109.

If it is not the first cycle, RAPSS-EOC loads the data from the previous cycle (lines 120-126). Lines 127-164 create an estimate of the current state by simulating one hour blocks of time with the wind speed and direction loaded from the real time data. These state estimates are log-scaled and output to the file, *currentStateOut*, which is the current state, plus restart number, then .csv (e.g., currentState2.csv). The parallel section of RAPSS-EOC lives in lines 170-255. After setting the number of threads to the user defined value, *requestTh*, (lines 171-191), the truly parallel section begins as line 196. Shared memory are *nthreads*, the number of threads; *flagVec*, the array that signals if the city is in danger or not; and *transientExplanation*, which is a vector of explanations about where the wind is coming from and at what speed. Lines 209-216 create directories one thread at a time (omp critical structure) for the thread data. Lines 220 and 221 sample wind speed and direction from the windrose data file. Lines 227-234 run the plume program on each thread, and communicate what each thread is doing to *transientExplanation*. The data are log scaled and added to the estimate of the current state (Lines 238-244). Lines 245-254 load *flagVec,* with the information described above. Lines 258-264 run the grid organizer script, perform mean shift analysis, and generate the plots. Finally, line 266 calls htmlDisplayWriter() to create the display for the recently completed cycle.

D.3. FunctionsEOC.h Code Explanation

As suggested by the title of this header file, this file contains the majority of the functions used in RAPSS-EOC (see Appendix C.3). This file can be considered the third layer below Pmain.cpp. The functions in FunctionsEOC.h will be briefly explained in this section. Many of the functions in this section are taken verbatim from BloodAndGuts.h from RAPSS-STA. Only the functions that are different will be described in this section. Lines 018-040 list the functions that will be defined in later line numbers.

- LoadGridData() (lines 076-102) returns a 2D vector of doubles that corresponds to the grid of concentrations that will be eventually passed to R for display. loadGridData() expects a file name of a csv file of log-scaled concentration data. It then raises each concentration to the power of 10 (antilog), to allow for easier addition of concentrations later.
- realTimeSimulator() (lines104-125) outputs to csv file as well as returns a vector of strings. This is meant to read wind history data and output it in real time to simulate not having acess to all of the wind history data at once.
- windDirTranslator() (lines 128-164) translates from the 16 normal directions on a compass, (e.g., N, NNE, NE, etc…) to polar coordinates. West is given a value of zero radians, WNW of $\pi/8$, and so on around the unit circle. This outputs a double corresponding to the value of the direction in radians.
- sigYFinder() (lines 166-190) determines the value of σ_y using the methodology described in Section 2.6.3. It is determined by the stability class, effective stack height and the position downwind of interest.
- sigZFinder() (lines 192-214) determines the value of σ_z based on the methodology described in Section 2.6.3. It is determined by the stability class, effective stack height and the position downwind of interest. It is worth noting that for most

experiments, z was set to zero (ground level concentration), so this function was rarely used.

- PrintGrid() (lines 245-258) writes to a .csv file, outputFileName the 2D vector of concentrations, *grid*, given in the input.
- initR() (lines 260-288) initializes R. This makes directories for the R files, and writes initR.r. See Appendix D.7 for further details.
- updateRwindex() (lines 296-305) updates pertinent information in R. This writes updateRwindex.R. See Appendix D.8 for further details.
- loadWindData() (lines 348-382) loads wind rose data from the RAWS weather data archive. The user downloads a wind history file from http://www.raws.dri.edu/index.html for data, and passes the name of the file to loadWindData(). The function reads it (lines 361-380), and returns a vector of strings that corresponds to the wind rose data for that the desired timeframe.
- windRoseNumTranslator() (lines 386-442) translates from the standard way directions are expressed in a compass setting, into a mathematical form. This expects integer that corresponds to a given wind direction (e.g., 0 is N, 1 is NNE, 2 is NE, etc…), and returns the value of that direction in radians, expressed as West is given a value of zero radians, WNW of $\pi/8$, and so on around the unit circle.
- *sampleWind() (lines 444-519) is one of the more complex functions, so it will be described in sections. Lines 445-465 are simply local variable definitions. U1 and U2 are random numbers, created at run time, that correspond to a uniform distribution between zero and one. Lines 467-471 load the probabilities that wind

is blowing in each direction. Lines 472-489 load the probabilities that the wind is blowing a certain speed given the direction. A normalized cumulative distribution function (CDF) is then created (490-505) across these wind speeds and directions. Lines 507-514 then use random numbers created at the beginning, U1 and U2, to sample wind speed and direction from the CDF. This function returns a dynamic array with the members being direction and speed, respectfully.

- loadWindObsData() (lines 520-568) is a function that expects a wind rose data file downloaded from http://www.raws.dri.edu/index.html. This function reads the file and returns a vector of strings that are correspond to wind direction, speed, and time.
- scenClustTranslator() (lines 570-580) returns the cluster number that a scenario belongs to. This expects the scenario number and the clusterMembers array. It searches through the array until it finds the scenario number and returns which cluster it belongs to.
- firstClustMemeber() (lines 582-590) is a Boolean function that only returns true if the scenario is the first member of a cluster. This is important in the html display engine, and is used to avoid duplicate displays of the same cluster.
- radianDirTranslator() (lines 592-644) translates from the direction expressed in radians to the direction expressed in normal compass directions. It has been written for directions in radians from zero to 4π, to compensate for the translations involved in the plume program.
- htmlDisplayWriter() (lines 646-887) is the function that creates the html display. Lines 660-676 count how many times the red and yellow trips have happened in

the scenarios to use in creating the red and yellow boxes in the display (see Section 9.4). Lines 678-697 name and create the files to be used, DISPLAY#.html, green#.html, unstable#.html, and misc#.html. Where the # symbol is used to represent the cycle number. DISPLAY#.html is created in lines 700-800. Green#html is the file that shows all clusters that did not trip, in other words, any cluster where the city is safe from the plume, written on lines 802-842. Misc#.html is written on lines 844-870 and contains the cluster information about the scenarios. Unsable#.html is mostly an artifact from the html interface with RAPSS-STA. Current it is set to always output “No model instabilities.”

D.4. PlumeProgram.h Code Explanation

This header file acts as the simulation software for RAPSS-EOC. The function PlumeProgram() returns a 2D vector of doubles, which corresponds to the grid of concentrations. Lines 12-23 are local variable definitions. Lines 25-61 compose of the structure for determining the concentration for a given space in the grid. Lines 31 and 32 rotate the X and Y axis along the direction of the wind, according to Equation (2.14). Lines 39-43 determine the concentration using Equation (2.5) for a given square of the grid. If the loop calculates the concentration in an area that is not in the path of the plume, it is assigned a concentration of 0.0001, instead of 0.0 (line 34). This makes it possible to log scale the whole grid without returning undefined numbers. If the concentration is below 0.001, it is assigned a concentration of 0.0001. This assures that areas with “low” concentrations don’t appear lower than areas with zero concentration.

D.5. GridOrganizer.r Code Explanation

This script organizes the grid data similar to Table 9.2. Lines 19 and 20 read the concentration grids in the form of .csv files. Lines 17-30 actually do the organizing. The organized data are put into an array, MeanShiftReady (line 31), which is output as meanShiftReady#.csv (lines 42-43), where the # symbol corresponds to the cycle number.

D.6. PlumeDisplay.r Code Explanation

This script displays a color coded concentration grid with an overlaid compass and a dot for the city of interest. Lines 018-023 read the prediction data clusters and lines 025-035 reorganize it into normal grid coordinates. The current state is read at line 38, and the prediction clusters are added to the 3D array in lines 040-046. Lines 049-055 address an issue with the indexing between C++ and R. In short, due to the method of indexing the values read by R, the grid appears reflected across the Y-axis without the routine executed in lines 049-055. The color pallet for concentrations is defined in lines 057-060. The loop in lines 069-151 generates plots in the form of two-page PDFs, where the first page contains a recreation of the current plume, and the second is the prediction from one of the clusters of the future plume. Lines 076-079 plot the color-coded concentrations. Lines 091-097 add contour lines. This has been disabled because for the plume data it often appears “messy” looking. Lines 100-135 overlay the unit circle with the standard 16 directions. Lines 133-144 display scale and labels on the y-axis. Finally, the color scale is created in lines 146-147.

D.7. initR.r Code Explanation

This simple script is executed only once at the beginning of the cycle. It loads pertinent information that does not change with each cycle into the R environment.

runAheadTime (line 14) is the number of hours ahead for the plume program to predict. GridResolution (line 15) is the size of the squares of the grid, in meters. MaxY (line 16) is the size in meters, of the grid in one direction. ReleaseAmt (line 17) is the quantity of radioactive material, in Becquerels, released from the facility.

D.8. updateRwindex.R Code Explanation

In RAPSS-EOC updateRwindex does not pass very much information. It simply updates the cycle number (line 02), and the current time (line 03).

D.9. Sample RAPSS-EOC Input File Explanation

The input file has three sections, data analysis parameters, RAPSS parameters, and plume program parameters.

Data Analysis Parameters:

- Card 101: MSA bandwidth. This is the bandwidth used in the mean shift algorithm (see Sections 2.5 and 6.6). It controls the cluster size, and membership. Smaller bandwidths yield more clusters with fewer members. Larger bandwidths yield fewer clusters with more members per cluster.
- Card 102: R library file path. This is the file path to the desired location for storing R libraries. If the location does not already exist, it will be created. The path should start, but not end with a forward-slash (“/”).
- Card 103: R website. This is the website RAPSS will access to download R libraries. Suggested address: http://cran.r-project.org.

RAPSS Parameters:

- Card 201 is the requested number of threads. If this number is below two, two threads will be used. If this number is greater than the maximum threads on the computer, the maximum threads will be used instead.
- Card 202 directory for storing files (inDir). This is the directory that will eventually contain the RAPSS-EOC data.
- Card 203 real time speed up multiplier. This number determines how much faster than real time the weather history data are read. A value of one will correspond to true real time. For interesting results a value of around 480 is suggested.

Plume Program Parameters

- Card 301 is the grid resolution. It is the size of the squares, in meters, of the squares of the concentration grid.
- Card 302 is the max Y-value. It is the size in meters, of the grid in one direction.
- Card 303 is the effective stack height, in meters.
- Card 304 is the height above ground that the plume concentrations are displayed for the generation of the grid. For demonstration purposes, this is set to zero.
- Card 305 is the release rate in Becquerels per second.
- Card 306 is the dt in seconds, used for the plume program. Increase this value to increase the speed of the program. Decrease the value for greater resolution.
- Card 307 is the amount of time to predict ahead from the current time, in hours.
- Card 308 is the time of day the plume starts. This expects a format similar to 3 AM, i.e., one number followed by an AM or PM.

- Card 309 is the time of day that the simulation begins at. This expects a format similar to 3 AM, i.e., one number followed by an AM or PM.
- Card 310 is the name of the wind observation data file from http://www.raws.dri.edu/index.html
- Card 311 is the name of the wind rose data file from: http://www.raws.dri.edu/index.html195