Parametric Testing of Launch Vehicle FDDR Models

Johann Schumann*
SGT, Inc., NASA Ames, Moffett Field, CA, 94035, USA

Anupa Bajwa†
NASA Ames Research Center, Moffett Field, CA, 94035, USA

Peter Berg†
SGT, Inc., NASA ARC, Moffett Field, CA

Rajkumar Thirumalainambi
RIAEX Inc. Sunnyvale, CA, 94087

For the safe operation of a complex system like a (manned) launch vehicle, real-time information about the state of the system and potential faults is extremely important. The on-board FDDR (Failure Detection, Diagnostics, and Response) system is a software system to detect and identify failures, provide real-time diagnostics, and to initiate fault recovery and mitigation. The ERIS (Evaluation of Rocket Integrated Subsystems) failure simulation is a unified Matlab/Simulink model of the Ares I Launch Vehicle with modular, hierarchical subsystems and components. With this model, the nominal flight performance characteristics can be studied. Additionally, failures can be injected to see their effects on vehicle state and on vehicle behavior. A comprehensive test and analysis of such a complicated model is virtually impossible. In this paper, we will describe, how *parametric testing* (PT) can be used to support testing and analysis of the ERIS failure simulation. PT uses a combination of Monte Carlo techniques with n-factor combinatorial exploration to generate a small, yet comprehensive set of parameters for the test runs. For the analysis of the high-dimensional simulation data, we are using multivariate clustering to automatically find structure in this high-dimensional data space. Our tools can generate detailed HTML reports that facilitate the analysis.

I. Introduction

MANNED launch vehicles are extremely complex systems, the life of the astronauts depend on the safe and reliable operation of the entire vehicle. Therefore, real-time information about the state of the system and potential faults is extremely important. The on-board FDDR (Failure Detection, Diagnostics, and Response) system is a software system to detect and identify failures, provide real-time diagnostics, and to initiate fault recovery and mitigation. In order to test the FDDR model and software, the ERIS failure simulation model has been created for the Ares I launch vehicle. The ERIS model contains models for the external vehicle dynamics and controls as well as internal vehicle dynamics and response, at various levels of detail. Overall, the model has more than 900 physical and design parameters; more than 170 variables are recorded during each simulation run. This ERIS failure simulation is a collaborative effort from a few groups across NASA, providing their individual physical models, which are are integrated into one large Matlab/Simulink model. With this model, the nominal flight characteristics can be studied and failures can be injected.

A careful test of the ERIS model is important to ensure its quality for accreditation as a Critical Math Model. However, a comprehensive test and analysis of such a model is virtually impossible because of its size and complexity. The failures, which are injected for testing can occur at various times during the ascent (e.g., during the burn of the First Stage or the Upper Stage) and they can occur in isolation (single failure) or as multiple failures. Since the failures can affect the model behavior, these interdependencies create a

*This work was in part funded by NASA OSMA's SARP program.
†Program Funding through Ares Vehicle Integration Office

huge space which must be processed for testing and analysis. In order to demonstrate robustness of the ERIS model, a sensitivity analysis over model parameters is necessary. This is in particular important since the sub-models have been developed by different groups, at different levels of fidelity, where parameters values can be the "engineer's best guess".

In this paper, we will describe, how *parametric testing* (PT) has been used to support testing and analysis of the ERIS failure simulation. PT uses a combination of Monte Carlo techniques with n-factor combinatorial exploration to generate a small, yet comprehensive, set of parameters for the test runs. N-factor exploration assumes that any problem is caused by the interaction of at most n specific parameter values (usually n=2 or n=3). With that assumption, which has been observed in many software systems, the number of required test runs can be reduced drastically to usually several hundreds to a few thousands. We will describe this technology and discuss, how the ERIS model has been tested by variations of key parameters as well as systematic variations of the injected failures.

Even a limited number of test runs creates a vast amount of data, as hundreds of model variables are recorded over time. We are using multivariate clustering algorithms (which have been generated with the AutoBayes[11] tool) to automatically find structure in this high-dimensional data set and to locate and estimate safety margins. Our tool set, which has been implemented in Matlab also generates detailed HTML reports, which documents the results of testing and analysis and allows the model designer to interactively navigate the data. Our testing and analysis environment is highly flexible and can be adapted easily toward other Simulink models[1] as well as other simulation environments (e.g., Trick[2]) or software algorithms.[3]

Of course, our tool is not the only testing environment suitable for the analysis of large Simulink models. Mathworks' System-Test[a] is a tool to automatically test a Simulink model. However, it only provides random Monte Carlo testing or full combinatorial exploration, which, in most cases would result in a prohibitively large number of test cases. The tool generates elaborate testing reports, but does not perform any advanced statistical analysis on the test results. Other tools, like Symbolic PathFinder,[4] T-VEC[b], or MathWorks' Design verifier[c] can automatically generate a minimal set of test cases that completely covers all elements of the model. However, these tools are not tailored toward parametric variations or robustness analysis, and they usually do not scale to large models.

The rest of this paper is structured as follows: in the next section, we describe the basic structure of the ERIS Simulink model and it characteristics. Section III the discusses the analysis requirements that we address by parametric testing, which is presented in detail in Section IV. Section V discusses, how the results of the simulation runs are presented in autogenerated HTML reports, how multivariate clustering is used to find structure in this usually huge data set and a statistical approach toward error propagation and parameter sensitivity analysis. Finally Section VI discusses future work and concludes.

II. ERIS Failure Models and Simulation

II.A. The Ares I Rocket

NASA's Ares I is an in-line, two-stage rocket topped by the Orion Crew Exploration Vehicle (CEV) and its Launch Abort System. In addition to its primary mission to carry astronauts to Earth orbit, Ares I could also deliver supplies to the International Space Station (ISS), or could "park" payloads in orbit for future retrieval. During ascent, the First Stage boosts the vehicle towards low Earth orbit. About two minutes after launch, the spent booster separates, and the Upper Stage Engine ignites to power the vehicle into a circular orbit. It can be structured into the categories: The First Stage is a single, five-segment reusable Solid Rocket Booster (SRB) similar to the Space Shuttle's solid rocket motor. The Upper Stage uses a J-2X engine which generates thrust by combusting its cryogenic liquid propellants – oxygen and hydrogen. Ares Elements are shown in Figure 1.

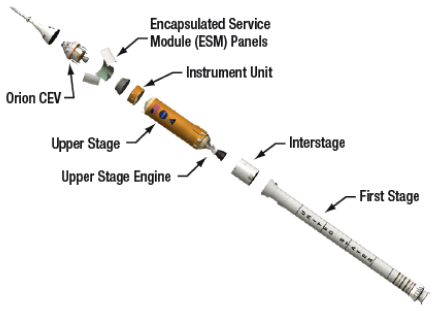

Figure 1. The Ares I rocket and its elements

[a] http://www.mathworks.com
[b] http://www.t-vec.com
[c] http://www.mathworks.com

II.B. The ERIS Failure Simulation

The ERIS Failure Simulation is a unified, modular, hierarchical representation of the Ares I Launch Vehicle, using quantitative equations. The physical models of various subsystems are integrated using the Matlab/Simulink environment for an *integrated* ERIS model. With this model, the nominal flight performance characteristics can be studied. Additionally, failures can be injected to see their effects on vehicle state and on vehicle behavior. In this model, failures are injected to simulate the overall behavior of virtual model of Ares I. These failures do not include vehicle structural failures and catastrophic events (e.g., break-up). ERIS Failure Simulation outputs time-stamped vehicle state variables to create scenario data sets, for specific injected failures. As a baseline simulation, nominal flight characteristics are established for the ERIS model. ERIS, which stands for Evaluation of Rocket Integrated Subsystems, is an integrated model of all major subsystems such as Avionics, Ground and Control, RCS, TVC, MPS, and J2-X, at various levels of detail (Figure 2). The ERIS model, developed in Matlab/Simulink, is hierarchical in nature. It started as a generic launch vehicle model and then Ares-specific details to the subsystems were added.

It can be structured into the categories: (a) the external vehicle dynamics and controls (with 6 subsystems) and (b) the internal vehicle dynamics and response, which is comprised of 10 subsystems. Apart from these blocks, multiple pieces of avionics systems along with special controls are added to the integrated system. More specifically, the model accounts for Ares I geometry (in Constellation coordinate systems), subsystem and component functions as equations, aerodynamic properties, flight dynamics, mass properties, Earth's atmosphere, and gravity. Most subsystems, end effectors and sensors (real measurements, as well as computed values) are modeled. ERIS provides an easy way to inject faults, component failures, command failures, and sensor noise to study their effects on integrated vehicle performance. It can simulate failure modes such as loss of vehicle and/or loss of mission due to major emergency conditions. The model generates time-stamped sensor readings on Ares I for a nominal flight, and for off-nominal cases, spanning the ascent phase of the mission timeline. The ERIS Failure Simulation is a very large Simulink model (containing approximately 16 top-level blocks, more than 900 inputs, parameters, and gains, and more than 170 outputs) but it easily runs on a single-processor desktop.

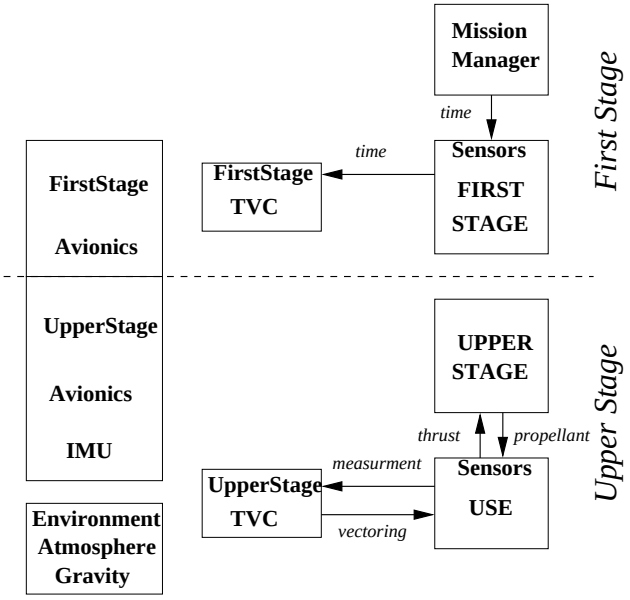

Figure 2. Major ERIS subsystems for first and upper stage

It produces repeatable results and is being tested in multiple ways: through parametric testing (as discussed in this paper), with test case generation tools, and with integrated integrated vehicle model comparisons with outputs of other models such as MAVERIC from MSFC. ERIS failure analysis adopts the approach of parametric testing in finite input-output parameter spaces along with constants and gains of the model.

III. Analysis Requirements

The ERIS Failure Simulation is a critical part of the Ares 1 on-board software development. Therefore, it must be tested thoroughly for accreditation. As discussed above, the system must not only be exercised using different failure scenarios, but also a variation of important model parameters (out of the more than 900 parameters) must be used for robustness analysis. In this paper, an injected failure is always attached to a Simulink signal, has a time of occurrence (seconds into ascent), and an indication if this is a First Stage or Upper Stage failure. Obviously, an exhaustive exploration is prohibitively expensive. Thus, we aimed to perform the following set of experiments

1. systematic excitation of First Stage failures and variations of failure times (single-failure mode),

2. systematic excitation of Upper Stage failures and variations of failure times (single-failure mode),

3. simultaneous injection of one First Stage failure and one Upper Stage failure at various times (systematic excitation), and

4. parametric analysis of important continuous performance parameters, like efficiency of fuel pumps or amount of loaded propellants in the baseline configuration (no failures) and in the failure case

The test objective is to select a reasonable subset of variables, and their ranges, which can generate sufficiently distinct trajectories for Ares I to define an expected envelope of trajectories. Thus improbable, or extremely unlikely, scenarios are not considered such as the "no fuel loaded" case.

Because of the size of the model and rapid succession of model versions, we have developed a testing harness, which can execute all runs automatically, and which only requires minimum changes to the model. The definition of which parameters and variables are to be used for each experiments can be easily done by the user using an Excel spreadsheet. After the execution of all runs, data analysis is performed and the tool generates several HTML reports, which will be presented in more detail below. These reports present the data run-by-run, or variable-by-variable. Failure information and correlations between recorded variables are also presented in HTML reports. The main goal of these reports is to enable the analyst and modeler to easily navigate the test results and obtain comparisons between the nominal and failure cases.

IV. Parametric Testing

IV.A. The Testing Process

The overall testing consists of several steps, which have to be prepared and carried out. Figure 3 gives a sketch of this process. Based upon the desired test goals (testing for failure cases, parametric analysis, or a combination thereof), appropriate sets of test cases are generated. These are provided to the ERIS simulation model and executed. Our test harness can automatically instrument the Simulink model so that only minimal user interactions are required. All simulation runs need to be monitored to catch potential problems in the simulator. Finally, the recorded data from the simulation runs are analyzed and HTML reports are generated.

IV.B. Generation of Test Cases

As the full combinatorial exploration of all possible values of input and model parameter reaches infeasible numbers, we experimented with a combination of combinatorial exploration of selected failure modes and failure time, n-factor combinatorial exploration, and Monte Carlo random generation. As input for the testcase generation an Excel spreadsheet provides the following information:

1. For each model parameter to be varied, the nominal, minimal, and maximal value is given. The tool then generates test cases with values within the range. The domain specialist should provide appropriate values for the minimal and maximal values.

2. For each possible failure mode, selected from the auto-generated spread-sheet, the following information must be provided by the domain expert: can the failure mode be selected at all? if yes, at which points in time (simulation time) will the failure be injected (e.g., at t=0 or t=110s or t=150s)? The user must furthermore specify if the failure mode belongs to First Stage or Upper Stage, and must select if a single failure is activated or a combination of 2 failures is used.

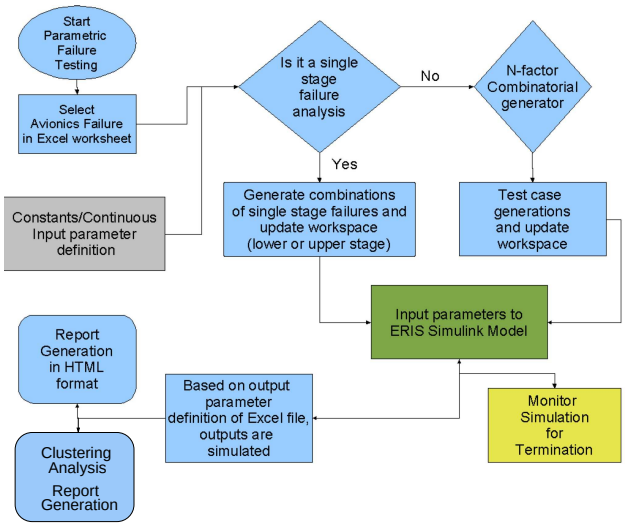

Figure 3. Process Flow Chart for Parametric Analysis

In our experiments, we generated a full combinatorial exploration for the failure modes, as the number of possibilities for the single failure and the 2-failures were reasonably limited. For the model parameters (and optionally the failure modes as well), the tool can switch between Monte-Carlo and n-factor.

The Monte Carlo (MC) testcase generation treats each input variable as a statistical variable with a given probability density function (in our case Gaussian or uniform), from which values for the test cases are randomly drawn. Although MC test cases can be generated very easily, there is no guarantee whatsoever regarding uniqueness and coverage of the input space, resulting again in the need to generate a huge number of test cases.

In real systems, most failures are caused by a specific, single value of one input variable or parameter. The case that a fault is triggered by a specific combination of two variables is much less likely. Even more unlikely is the case that 3 input variables must have specific values in order to trigger the failure; the involvement of 4 or more variables can be, for most purposes, ignored. This observation[5–7] has been used to develop the n-factor combinatorial testcase generation. Here, the generated cases completely cover all combinations of variables up to n.

The literature describes a number of efficient algorithms[8] including the IPO algorithm.[9] For our experiments, we used a tool developed at JPL,[1] which extends the IPO algorithm. Table 1 shows the number of test cases generated for different settings. For each continuous variable (e.g., tank pressure), the entire variable range is subdivided into k equidistant bins. If a bin is selected, a random number from this bin is drawn and comprises the actual parameter value. For our experiments, we considered three sets of variables: a small set of 5 continuous parameters, a set of 12 parameters, and a combination of these 12 parameters with 8 discrete failure mode variables. Table 1 lists the different combinations. For the continuous variables, we chose 5 or 10 bins; the discrete variables had different numbers of values each (last row). A full combinatorial exploration (comb) generates a very large numbers of test cases, compared to n-factor. The table lists the number for various values of n and the run time to generate those (on an intel MacBook Pro). Run times were limited to 10 minutes.

No Vars	No Bins	comb	1-fact	2-fact	3-fact	4-fact
5	5	$3 \cdot 10^3$	10	68 [<1s]	360 [<1s]	1,534 [2s]
12	5	$244 \cdot 10^6$	10	99 [<1s]	690 [2s]	4,355 [85s]
5	10	$100 \cdot 10^3$	20	255 [<1s]	2,768 [2s]	24,338 [100s]
12	10	$1 \cdot 10^{12}$	20	361 [2s]	5,317 [66s]	–
20	var	$87 \cdot 10^{18}$	34	556 [2s]	9,640 [610s]	–

Table 1. Number of test cases generated with full exploration (comb) and n-factor ($n = 1, 2, 3, 4$) exploration.

IV.C. Running the Tool

As described above, the entire configuration for the tool is stored in an Excel speed sheet. A default spread sheet with all possible variables can be conveniently generated directly from the Simulink model using our tool set. From this input, the tool automatically generates all required test cases. The Simulink model is then automatically instrumented such that it uses the specific input values. When each test case is executed, selected model variables are stored as time series data (arrays with time stamps). These data form the basis for subsequent data analysis. For efficiency purposes, all simulations were executed using Simulink's "rapid acceleration mode", where the model is first compiled into C and then run as a separate executable. For each iteration, the Simulink model runs for a pre-defined simulation time spanning the burn phase for First Stage, separation, and the main burn for Upper Stage.

As discussed earlier, the ERIS model is a continuous model. Exercising it with parameters, which deviate (on purpose) substantially from nominal values can cause a non-termination of the Simulink model. In most cases, such parameter settings cause oscillations, which require an extremely small step size. Unfortunately, Simulink does not provide any functionality to abort a simulation after a certain amount of *CPU* time. An additional Simulink subsystem, which polls the CPU time in regular intervals and stops the simulation after the limit has been reached did not work for our purposes, because in several cases, the simulation did not terminate in the inner loop. To avoid such problems, an external Unix process was started, which monitored

the simulation process and killed it after it reached a CPU time limit. Only with this kind of external safe-guards it has been possible to automatically run thousands of test cases reliably.

V. Data Analysis and Results

For each run, all values for selected model variables are stored as arrays with time. Although most experiments yield large data sets, we chose to keep all raw data in order to be as flexible as possible for the subsequent data analysis phase. This is in contrast to many other testing harnesses, where a specific "pass/fail" property has to be provided up-front and any change of the property requires a time-consuming re-execution of the test cases (as in e.g., Mathworks' System-Test). Using these simulation data, our tool set provides several visualization and analysis modes, which can be triggered by Matlab commands or which can be accessed through a simple graphical user interface.

V.A. Generation of Reports

For each experiment, the tool automatically generates two standardized HTML reports. Because this tool is being used repeatedly with different versions of the models, all relevant information about the model and the experiment set-up must be provided in one single document. We chose HTML, because it does not require a special viewer, is easy to generate, and provides all necessary methods for simple navigation. The current version of the tool, however, does not provide any traceability of the experiment and its results to elements of the Simulink model or requirements documents.

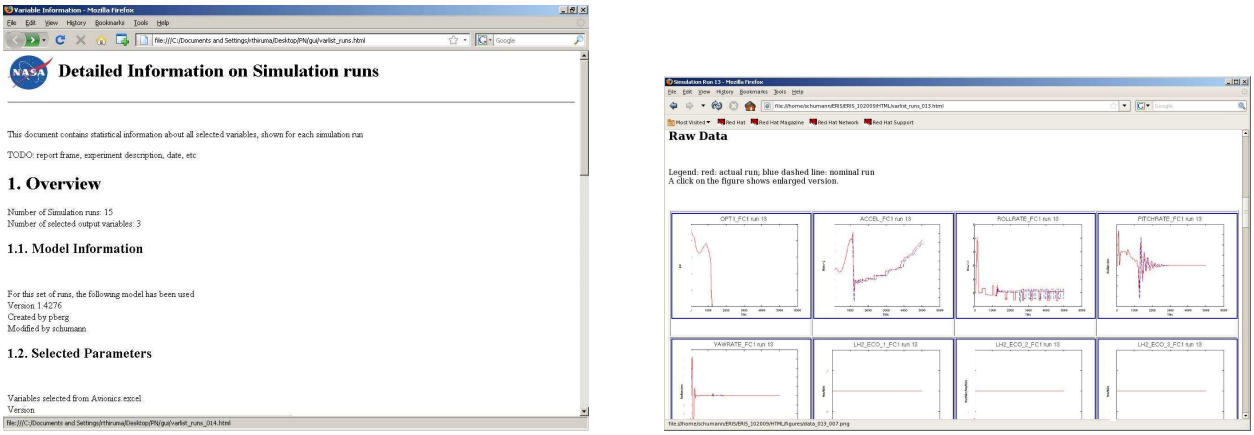

Figure 4. Autogenerated HTML report (excerpts)

Figures 4 and 5 show selected screen shots from the report, which presents the data run-by-run. The first figure displays the top level portion of this report. A standardized (modeled in close accordance with relevant NASA standards) header contains all important information about the model and the experiment and lists, which parameters have been varied and which failure variables have been set. Then, a tabular overview of all runs is presented, together with a summary information about each run. Of particular interest here is the information if the simulation terminated early or ran through the entire specified simulation time. Each entry is hyperlinked into pages, which show all details of each individual runs (Figure 4(right) is an example). Here, thumb nail plots for each recorded variable and a comparison with the nominal run (in blue) give a quick overview on what happened during the simulation. Clicking on a thumb nail opens up a full-size plot in a separate window (Figure 5). This specific figure shows the acceleration related to Flight computer 1 (FC1). The pages for the individual runs also contain a list of error messages that occurred during the run as well as the actual setting for each variable in tabular format (not shown).

A simple graphical user interface, depicted in Figure 6 allows the user to visualize individual simulation variables for a selection of simulation runs, in our case, relative values of the roll-rates are shown.

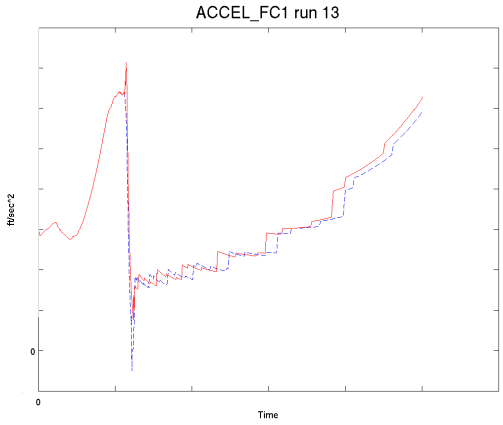

Figure 5. Plot of individual variable shown with comparison to nominal scenario (blue).

A second HTML report produces plots for each recorded model variable ("variable-by-variable report"). Here, the traces for all runs are superimposed. Figure 7 shows a plot where the acceleration is shown over time. Different parameter settings usually cause minor variations with respect to the nominal case. A big deviation from the nominal behavior is caused by the effects of an injected failure that becomes active at around the index of 200. The top group of curves shows that a substantial spread in acceleration can be a result of this failure. The parameter settings of the simulation run depicted in blue curve caused a big negative acceleration around separation, indicating a potential problem. The horizontal red line shows the default value, which is to be reached after separation of the stages. The purpose of this variable-by-variable report is to give the analyst a quick overview on how the parameter variations in the experiment influenced the development of the recorded variables. Drastic changes indicate that this model variable is sensitive with respect to the experimental settings and invites further investigation. Variables that do not change over the experiment (or only change very little) can be ignored for further data analysis or can indicate a problem with the model.

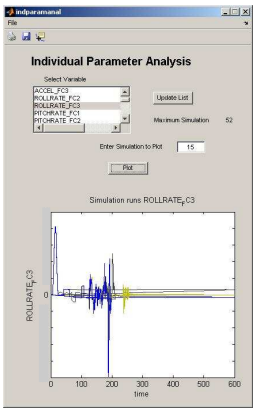

Figure 6. User Interface for Individual Parameter Analysis

Finally, a failure correlation report provides basic information on how parameter settings influence the outcome of each simulation run. This report produces scatter plots over each of the changed input parameters. Each run produces a single dot in each of the plots. These dots are colored according to the result of the simulation runs. In our example (Figure 8), dots are colored according to the abortion time of the simulation run. Blue dots correspond to successful simulations, magenta dots mark simulations, which failed mission goals, and red dots mark early simulation termination. This plot shows some obvious observations, for example that without a sufficient fuel load or highly inefficient pumps, the launch cannot be successful. This type of visualization can also provide more information, in particular if the analysis is combined additional statistical data analysis, which

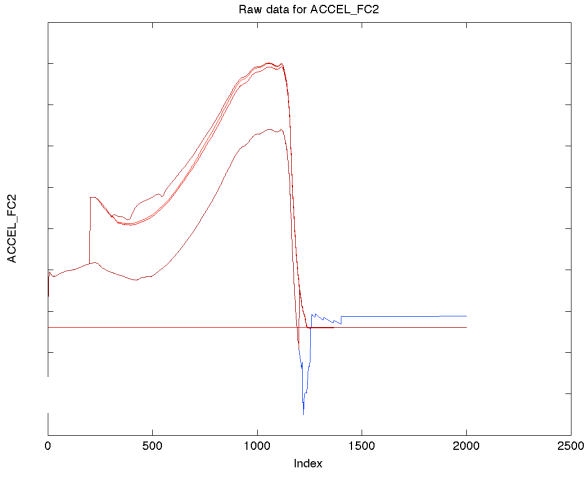

Figure 7. An example plot from the variable-by-variable report

we will discuss in the next section. There, the coloring of the data points is determined by more advanced analyses.

V.B. Clustering Analysis

The result of the simulation runs with injected failures and parametric variations result a very large, high-dimensional set of traces over time. Except for the analysis of individual runs or checking for a specific property, manual analysis is infeasible. In our work, we use *multivariate clustering*, a machine learning technique to automatically detect structure in a large, high-dimensional data set. All simulation runs are sorted into a number of individual groups according to their relative similarity. Then, this information can be used to guide visualization and further analysis.

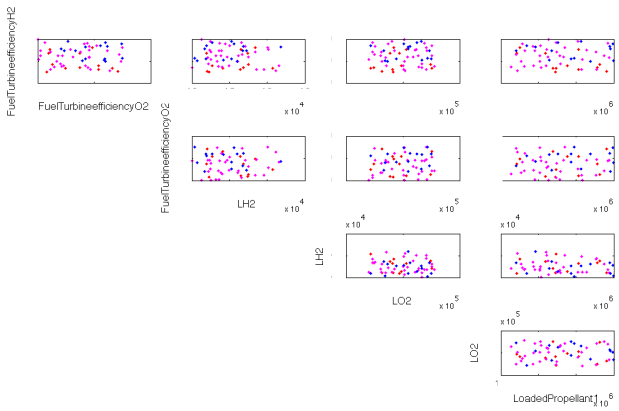

Figure 8. Failure scatter plots over a subset of varied parameters. (red = simulation failure, blue = mission success, magenta = mission objectives failed)

In a first step, we reduce the size and complexity of the data set by manually selecting a set of *features*. A feature compresses one variable trace into a single scalar value for each simulation run. Typical examples include "maximal value" (e.g., maximal roll acceleration during ascent), "value-at-end" (e.g., left-over propellant), or "sum", which provides an approximation of the integral, e.g., to measure the overall amount of exercised control. Other features yield discrete values, e.g., "unstable", "fault", or "separation occurred". These features can be generic or specific to the ERIS model. We have developed a simple graphical user interface (Figure 9), which allows the analyst to easily select features and customize the clustering analysis. This graphical user interface allows the user to select variables of interest (top left of Figure 9) and features as well as statistical properties for each selected variable (pop-up window on the lower right corner). After extraction of the data, clustering can be started; control parameters for clustering and other analysis methods (currently not used) and plots about the quality of the clustering are parts of the interface. Finally, HTML reports can be generated directly from this graphical user interface.

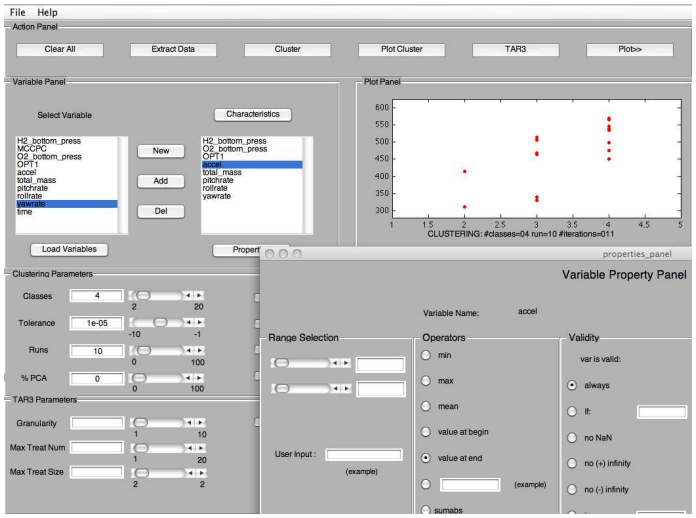

Figure 9. AutoBayes Clustering User Interface

For the actual clustering, we use C code that has been generated by the AutoBayes tool. AutoBayes[10,11] (developed at NASA Ames) is an automatic system that generates customized data analysis algorithms from high-level statistical specifications. The selected features in general have different statistical properties and probability density functions (PDF). Some features are normal (Gaussian) distributed, whereas others require specific PDFs (e.g., for angular values). Thus, the use of library functions (which usually only handle normal distributions) is not possible. Since each data analysis job requires a different customized algorithm, the AutoBayes tool is ideally suited for this task.

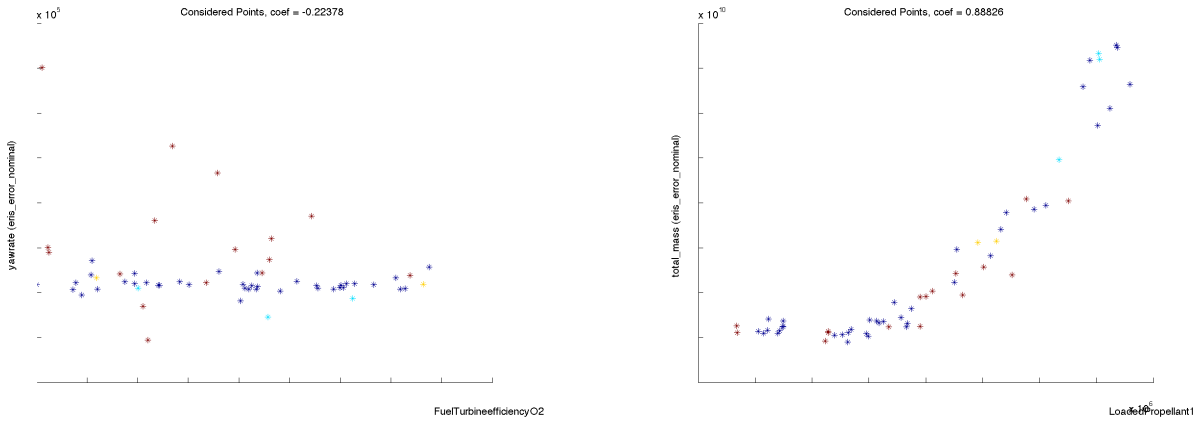

Figure 10. Clustering Result

Once the data have been clustered into a user-selectable number of groups (usually 3 to 8), another HTML report is generated. This report is similarly structured to the ones discussed above. For pairs of features that have a strong correlation with each other, correlation plots are generated where each data point is colored according to which group it belongs. An example is shown in Figure 10. Both panels show the correlation between a varied input parameter (pump efficiency, loaded propellant) and the selected feature of a recorded simulation variable (deviation from nominal case in yaw rate and total mass, respectively). Each dot corresponds to a single simulation run. Here, the dark blue class depicts nominal or close to nominal behavior; brown dots show substantial deviation in the yaw rate. Yellow and light blue classes do not show a deviation in this variable, but the light blue runs obviously only occur with large amounts of loaded propellant, as indicated on the right panel. The right panel also reveals that the deviation in the yaw rate is obviously independent of the amount of loaded propellant over large ranges, because there is no clear separation of the brown and dark blue points in this plot.

V.C. Parametric Estimation

A detailed analysis of the uncertainties involved in the ERIS model and parameters is essential to understand interactions within the integrated model. We have been studying these issues using parametric testing described above. The sensitivity and error propagation of the simulated data set has strong similarities with design of experiments. Sensitivity analysis looks at the effect of varying the inputs and parameters of a mathematical model on the output of the model itself. Global and local sensitivity analysis uses a least square approach with respect to the data set and model as well by varying input and model parameters. This error analysis can be helpful to understand the behavior of the model and the parameter interactions in the model.

Our parametric estimation analysis of variables during the ascent phase of the mission is specifically focused on temperature, pressure, and propellant. The error analysis provides results regarding variable sensitivity and impact on the model output. For this experiment, 200 noisy vectors were used as input data for the model. The nominal model output is considered as the baseline and a sum of square errors between nominal and simulated data is calculated. These numbers and cumulative distribution function of the error provides insight into parameter sensitivity. Optimization techniques like smoothing or regularization can be applied to these results, so that the envelope of simulated outputs can be estimated. The steps for this analysis are shown in Figure 11(left). The right panel of this figure shows results of 200 runs, 83 of the runs were above the average of sum of square error. The plot shows the pitch over time with a three sigma

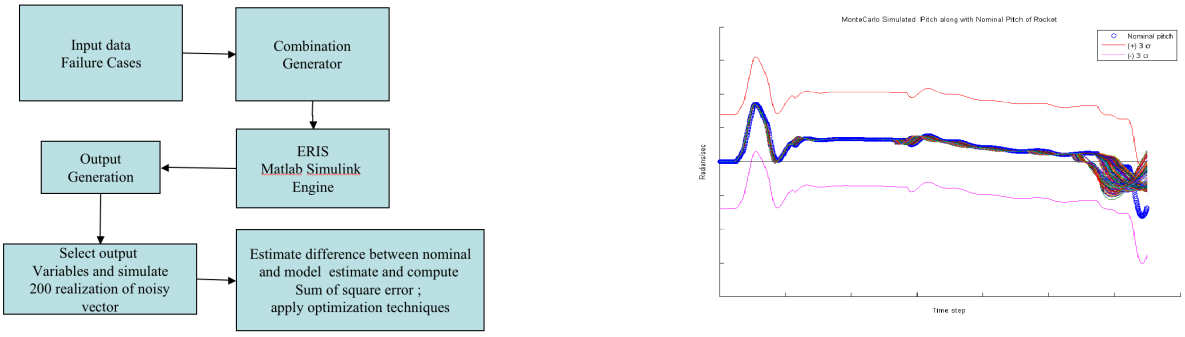

Figure 11. Steps for calculation of parameter sensitivity and pitch rate with a 3σ envelope (right).

envelope that can be considered as a safe range to estimate and propagate the error. Results of this analysis allow engineers and designers to re-evaluate model structure and input parameter ranges.

VI. Conclusions

In this paper, we have presented, how parametric testing can be applied to the ERIS failure simulation. With a set of tools, which have been developed at NASA Ames we have been able to test the large ERIS Simulink model on autogenerated sets of test cases, perform data analyses of the recorded simulation data, and generate HTML reports. Because of the size and complexity of the model, our sophisticated way of generating parameter variations and failure scenarios by using a combination of full exploration, n-factor combinatorial, and Monte Carlo techniques proved invaluable to keep the number of test scenarios within a manageable range without loosing coverage.

Manual analysis of the results of hundreds of simulation runs with more than 170 recorded variables is impossible. With our tools, we can generate standardized HTML documents, which present the data in various views and thus allow the analyst to easily browse and navigate the data. The high-dimensional data set is automatically structured using feature selection and multivariate clustering. Future work will include advanced data analysis using the NASA Ames margins tool. With this tool, safety margins and root cause analyses in multiple variables can be performed.

References

[1]Schumann, J., Gundy-Burlet, K., Pasareanu, C., Menzies, T., and Barrett, T., "Software V&V Support by Parametric Analysis of Large Software Simulation Systems," *Proc. IEEE Aerospace*, IEEE Press, 2009.

[2]Gundy-Burlet, K., Schumann, J., Menzies, T., and Barrett, T., "Parametric Analysis of ANTARES Re-entry Guidance Algorithms Using Advanced Test Generation and Data Analysis," *Proc. iSAIRAS 2008 (9th International Symposium on Artifical Intelligence, Robotics and Automation in Space)*, 2008.

[3]Giannakopoulou, D., Bushnell, D., and Schumann, J., "Formal Testing for Separation Assurance," *accepted AMAI (special issue Formal Methods in Aerospace)*, 2010.

[4]Pasareanu, C. S., Mehlitz, P. C., Bushnell, D. H., Gundy-Burlet, K., Lowry, M. R., Person, S., and Pape, M., "Combining unit-level symbolic execution and system-level concrete execution for testing NASA software," *ISSTA*, 2008, pp. 15–26.

[5]Cohen, D., Dalal, S., Parelius, J., and Patton, G., "The combinatorial design approach to automatic test generation," *Software, IEEE*, Vol. 13, No. 5, Sep 1996, pp. 83–88.

[6]Dunietz, I. S., Ehrlich, W. K., Szablak, B. D., Mallows, C. L., and Iannino, A., "Applying design of experiments to software testing: experience report," *ICSE '97: Proceedings of the 19th international conference on Software engineering*, 1997, pp. 205–215.

[7]Wallace, D. R. and Kuhn, D. R., "Failure Modes in Medical Device Software: an Analysis of 15 Years of Recall Data," *International Journal of Reliability, Quality and Safety Engineering*, Vol. 8, No. 4, 2001.

[8]Grindal, M., Offutt J., and Andler, S., "Combination testing strategies: a survey," *Software Testing, Verification and Reliability*, Vol. 15, No. 3, 2005, pp. 167–199.

[9]Tai, K. and Lie, Y., "A Test Generation Strategy for Pairwise Testing," *IEEE Transactions on Software Engineering*, Vol. 28, No. 1, 2002, pp. 109–111.

[10]Fischer, B. and Schumann, J., "AutoBayes: A System for Generating Data Analysis Programs from Statistical Models," *J. Functional Programming*, Vol. 13, No. 3, May 2003, pp. 483–508.

[11]Schumann, J., Jafari, H., Pressburger, T., Denney, E., Buntine, W., and Fischer, B., "AutoBayes Program Synthesis System Users Manual," Tech. Rep. NASA/TM-2008-215366, NASA, 2008.