Assessment of a Space Shuttle trajectory evaluation system (DOLILU II)

Diwakar Chakravarthy
West Virginia University

Assessment of A Space Shuttle Trajectory Evaluation System (DOLILU II)

Diwakar Chakravarthy

Thesis submitted to the College of Engineering and Mineral Resources at the West Virginia University

in partial fulfillment of the requirements for the degree of

Master of Science

In

Computer Science

Department of Computer Science and Electrical Engineering

Morgantown, West Virginia
2000

Keywords: Software Reliability, Bayesian Statistics, High Assurance Systems

Assessment of a Space Shuttle Trajectory Evaluation System (DOLILU II)

by
Diwakar Chakravarthy

Abstract

DOLILU II is a ground control system that generates space shuttle's launch trajectories, first stage guidance commands and verifies whether the generated trajectories are safe for the flight. It is a safety critical system and a high degree of confidence in its safety and reliability must be gained through assessment. We addressed three issues related to its safety and reliability assessment. We developed a reliability assessment framework for DOLILU II system. We proposed techniques to speed up test case execution and designed methodologies for the generation of input conditions needed to test the system.

We used a Bayesian statistical framework for reliability assessment. Bayesian statistics uses knowledge about the system to be incorporated into the reliability model before testing. DOLILU II has been operational for nearly five years. We use this information when developing the reliability model. This information is introduced in the form of prior beliefs.

DOLILU II system requires an average time of 30 minutes for each test run. This translates into a large time period required for testing to demonstrate that DOLILU II exhibits the required failure rate. Vertical slicing, a semantic transformation technique, is used to prove the possibility of parallel execution and enhance each test case execution.

DOLILU II is an on-demand system. Many test trajectories are needed for its assessment. Regression methods were used to develop models for the generation of input data.

Acknowledgements

I would like to sincerely thank my advisor Dr. Bojan Cukic for his interminable patience and able guidance without which this document would have never been written. I would like to thank Dr. Jack Callahan (CSEE department, WVU) for his timely advice and information regarding DOLILU II system.

The work presented in the thesis was partially supported by the research grant from Averstar, Inc. I would like to thank Mr. D. McCaugherty of Averstar, for providing the necessary documentation for the system. He was very helpful in sharing vivid descriptions of testing strategies previously used to test the system.

I would like to thank my committee members Dr. Jim Mooney and Dr. Franz Hiergeist for their kind cooperation.

I would like to thank Dr. Iskander from IMSE department, WVU, for his help with statistics and regression methodologies used in this thesis.

I would also like to thank my friends for their efforts in perusing this document and making valuable suggestions.

Table of Contents

List of Tables

List of Figures

1.0 Chapter 1: Introduction and Overview

1.1 Introduction

From Blaise Pascal's modest invention of the first adding machine, to present day fast computing machines; computers have undergone a significant metamorphosis. Starting with the big bulky computers (**UNIVAC, ENIAC**) that cost over a million dollars to fast desktop machines available today, they have progressed slowly from solving problems of the scientific community, to permeate all occupations.

With computers becoming increasingly indispensable, there exists growing awareness about the *"untrustworthiness"* of one of man's greatest inventions. Though computer systems firmware consists of both hardware and software, there has been more emphasis on how much one's reliance on software can be trusted. There are several factors that abet this cause. The chief one that plagues software engineering unlike other major engineering disciplines, including even computer hardware, is that they have a more disciplined approach to both designing and addressing problems that persist in their fields. There is generally a predictable life span for a given hardware product. It can be modeled as a decaying rate of efficiency at which the product will deliver service. Software, on the other hand, cannot be attributed with any of these factors. Most disciplines are characterized by having a strong mathematical backbone they can rely on, a simplistic design process and a more gradual change attaching more reliability to the products thus evolved.

Design of software based systems is characterized by inherent "*complexity*" of software involved. Idiosyncratic nature of software have been exhibited by one and all computer systems. Despite several fallacies, there is still a widespread acceptance of software. The tolerance towards software failures is inversely proportional to the cost a person is willing to pay for the software.

Failures in certain systems, however, are unacceptable. We may classify them into two categories

- Money critical systems: systems that fall into this category are transaction processing like banking software, telecommunications like networking systems deployed for stock exchanges. Erroneous data processing could lead to bankruptcy of a firm!
- Safety Critical Systems: software systems controlling/monitoring process control systems like nuclear power plants, chemical plants, satellite software, flight control. These systems are characterized by an enormous capital investment in creating them and a failure that could endanger human life. In this thesis, we deal with systems belonging to this category.

1.2 Software Usage And Experiences

Bev Littlewood and Lorenzo Stringini expressed their views regarding the deployment of software in "systems where software is critical for safety". In their article[3], they delineated their growing concern, emphasizing the shortfall to guarantee software correctness. They state "*an appropriate level of safety can only be granted if the applicability of software manning critical process is limited*". Despite several problems that cripple software, it is still been used in several areas inclusive of process control environments. The main advantage is that it allows a great degree of flexibility when subsequent changes needs to be done to the system. Furthermore, automation of equipment in environments like nuclear power plants decrease the risk of the human user controlling the process from exposure to hazardous radiation.

There is an adage that goes “one learns from one’s own experiences”. The best way to know how software behaves when actually put to use is gathered only from field experiences [3]. To attest to this fact, we may quote several examples where, software failure has not only wasted enormous capital investment but also endangered and killed human life. The following subsections relate these very instances.

1.2.1 Ariane 5

The maiden flight of *Ariane5* (European Space Agency), launcher ended in a failure on 4 June 1996. The launcher veered off its intended flight course, broke up and exploded. The board documented the proceedings of the investigation [4]. The Flight Control Systems on Ariane5 measured the altitude movements of the spacecraft in space by an Inertial Reference System (SRI). The data from the SRI was conveyed through a databus to the On-board Computer (OBC). The OBC executes the flight program and controls the nozzles of the solid boosters and cryogenic engine through hydraulic actuators and servovalves. In order to improve reliability, there were two SRI's operating in parallel with identical hardware and software. The design of Ariane5 SRI was practically the same as on its predecessor and hence the software was reused.

After about 40 seconds into flight, due to an angle of attack of more than 20 degrees high aerodynamic loads were caused. This in turn led to the separation of the boosters from the main stage, triggering the self-destruct system.

The angle of attack was caused by full nozzle deflections of the solid boosters and Vulcain main engine. This was commanded by the OBC on data that was received from the SRI system. This was because of a bit pattern from SRI 2 that was misconstrued to be the flight data.

An internal software exception was generated on SRI 1 when a data conversion from a 64-bit floating point to a 16-bit integer was made. SRI 1 shut down transferring control to SRI 2. SRI 2 also detected the same fault but could not switch back to the SRI 1. SRI 2 hence conveyed incorrect flight data to OBC. The error occurred in the part of the software that only performs alignment strap-down inertial platform. This software provides meaningful information only before the lift off. This function was, by mistake, operational for 40 seconds into flight. This time sequence was a requirement in Ariane4 but not in Ariane5. The operand error resulted in a high value of Horizontal Bias (BH), related to horizontal velocity. The value of BH was higher than expected for Ariane5 as the early part of the trajectory was different from Ariane4. This cascade led to the destruction of the launcher.

1.2.2 Mars Climate Orbiter

On December 11, 1998, NASA launched the Mars Climate Orbiter (MCO) with the objective to observe the planets surface, profile the structure of the atmosphere, and detect surface reservoirs. This was a part of the Mars Surveyor Program started in 1993.[5]

Nine and half months into the flight, on September 1999, MCO was to fire its main engine to achieve an elliptical orbit around mars. A technique called "*aerobreaking*" was being used to maneuver the flight through Mars' upper atmosphere, to reduce velocity and move into circular orbit. On September 23 1999, MCO mission was lost when it entered the atmosphere on lower trajectory than expected.

The board investigated the mishap and proposed the following reasons for the failure. On September 8 1999, the interplanetary Trajectory Correction Maneuver-4 (TCM-4) was planned. This maneuver was expected to adjust the trajectory such that after the orbital insertion burn the point closest to the planet would be at a distance of 226 km. The data was up-linked to the orbiter in metadata files (AMD files). The on-board computer computed the orbital insertion distance. TCM-4 was executed as planned on September 15, 1999. Mars orbit insertion (MOI) was planned for September 23. During this period, orbit determination processing by the navigation team received data indicating that the first periapse distance had decreased to the range of 150-170km. 24 hours before MOI, MCO began to feel strong effects of Mars' gravitational field. Before one hour of MOI, first periapse altitude was 110 km.

The MOI was started and all systems performed nominally until Mars' occultation loss of signal. Signal was to be reacquired after the 21-minute interval predicted for the occultation period. There was no retrieval.

On September 27, 1999, the operations navigation team discussed navigation discrepancies regarding velocity change modeling issues. After two days it was found that the small forces, ΔV's, used in orbit determination on the orbiter before the initialization burn (MOI) was low by a factor of 4.45 (1 pound force = 4.45 Newton). The impulse bit data contained in the AMD file was delivered in lb.-sec instead of expected units Newton-sec.

After navigation estimates, using available data through loss of signal, with corrected values the initial periapsis of 57 km was calculated which was too low for the spacecraft survival. The estimated minimum altitude for survival was 80km.

1.3 Motivation

Safety in today's technological terms seems akin to the notion of "risk". R.N. Charette [9] defined risk as an action/event having the following characteristics

- Having a loss associated with it
- Where uncertainty or chance is involved
- Some choice is also involved

Safety based on the above observations may be phrased as " freedom from exposure to danger, or the exemption from hurt or loss" [11].

Several safety-critical systems follow this description. These systems require a very high degree of confidence in their functioning. With so much at stake, a rigorous framework of development is deemed essential when dealing with such high consequence systems. Assurance of specifications was the first step taken to ensure the correctness of requirements for the system. Languages like Z [20] have been successfully used to formally specify the requirements thereby minimizing design errors that might creep in due to an incorrect requirement definition document. Furthermore, to achieve greater reliability by masking faults introduced at the design phase of the project, *design diversity* [26] was adopted leading to fault tolerant and fault preventive systems [6]. Consequently, these systems seldom exhibit failures. The question now arises, are all the above factors sufficient to attest that the software will function without failures? When such a high degree of dependability is a requirement, when subversion of the system could cause a loss of life, it becomes apparent that one needs to endorse that the system indeed satisfies the reliability requirement.

.

Our work involved in developing assessment methods for NASA's Day of Launch I-Load Update (**DOLILU II**) system. The system evaluates trajectory parameters against the I-Loads (Initialization Loads) that dictate the guidance commands for the space shuttle. Wrongly approved trajectories and guidance commands could lead to the destruction of the space shuttle. This system was a high consequence system.

Rigorous development methodologies were adopted to develop the system. During each phase in the software life cycle, stringent and scrupulous techniques were followed to ensure proper design and development (the requirements and design phase went through 8 levels before final approval). During the verification and validation phase, however, methodologies only stress tested the system with no effort to quantify the reliability of the system. Furthermore, the system has been operational for the past 5 years. This gave us an opportunity to analyze the system and also develop a framework to assess its reliability.

We looked into several theories that have been proposed over the years on estimating the reliability of software in safety critical systems. They could be classified into three main categories

- Formal methods
- Exhaustive testing
- Statistical testing

Formal methods have their genesis in years of mathematical formulation. The underlying paradigm is that a program meets the "correctness" requirement if it satisfies its intended purpose. In other words, it is said to be functionally correct if it behaves according to the specifications. Program correctness is an absolute measure; the program under verification is either valid or invalid. There is no quantification of reliability; it is either zero or one.

The drawback with this method is the assumption that the specifications themselves are correct in nature. A correct implementation of a specification may still lead to unreliable execution of program due to an imperfect specification [10] . It's tedious to provide proofs for complex systems. Hence it was difficult to formally verify the entire DOLILU II system.

The more traditional way to estimate reliability of software is testing. Huang et al. [12] stated in his paper that exhaustive testing is impossible as it is difficult to cover the entire input domain for complex systems, to ensure complete execution of all paths in a program. With a complex system like DOLILU II, this methodology was ruled out.

In this thesis we lay emphasis on the third form of software assessment based on statistical testing techniques. These models perform repeated executions of software to provide a certain level of confidence that the required degree of reliability has been successfully achieved.

There were several challenges that we were posed with applying statistical methods to DOLILU II system.

- Our discussions with the NASA IV&V personnel revealed that DOLILU system required a demonstration of probability of failure less than 10^{-5}. With a statistical assessment framework this would translate into a large number of test cases. However, we had information regarding the development process, the testing that had already been done and the fact that the system has been in operational use for several years. We needed an assessment framework, which would incorporate this information into our assessment model to determine an acceptable time frame to assess the system.

- After developing a framework for assessment, we now needed to have sufficient inputs generated to test the system.
- As the execution time for DOLILU II was in the order of minutes, we needed to decrease the time required for each test case execution

1.4 Thesis Overview

The remaining thesis is organized as follows. Chapter 2 overviews existing reliability assessment techniques. Chapter 3 analyzes the different systems that comprise the **DOLILU II** system. Emphasis is laid on **DIVDT** (*Day of launch I-Load Verification Table*). This chapter also enumerates the different techniques that have been used to evaluate the working of **DIVIDT,** inclusive of test plans and their executions.

Chapter 4 introduces the notion of reliability as applicable to software. A comparison is made between classical sampling theory and Bayesian inference, delineating the merits of the Bayesian approach and its implications on assessing ultrahigh reliability for software based systems.

Chapter 5 describes the use of slicing techniques to split the system into subparts, and demonstrate the accelerated test case execution of the **DIVIDT** subsystem.

Chapter 6 address issues for auto generation of test cases. Here we explore regression methodologies that could be adopted for generation of input conditions. It also addresses issues regarding oracles.

Chapter 7 summarizes the thesis and suggests further work and other approaches for evaluating the system.

2.0 Chapter 2: Related Work

Historically, software creation has been viewed as a two-stage process. The first is the translation of requirements from informal description into formal specifications. Second is the translation of specifications into executable code.

When viewing this two-tier hierarchy one realizes that, there are at least two processes where faults can be introduced. The main problem with achieving such high reliability is the probability that subtle design faults always exist [13]. Design faults arise due to incorrect understanding of requirements or due to omission in the specification stage. Implementation faults occur during the second phase due to insufficient testing, verification and validation.

This leads to three schools of thought for ensuring the reliability of critical software. One way is verifying the correctness of specifications, indicating that all desirable properties were captured. The second group aims at masking the effect of faults. Techniques belonging to this category are fault tolerance and fault prevention. The third group of methods, applicable at the tail end of software life cycle, includes assessment techniques that instill the confidence that the requirements of reliability are met [6].

The remaining part of this chapter illustrates reliability assessment techniques, mainly statistical reliability assessment methods.

2.1 Reliability Assessment Methods

Techniques, like assurance of specifications, vouch for the correctness of the specifications. They reflect that all the properties were captured. It improves the removal of faults introduced during the requirement specification stage. Nevertheless, the question remains does it truly reflect the reliability of the system. One cannot assure that the implementation and design were devoid of faults. One might however reduce any ambiguity in understanding the specifications.

Fault Tolerant and Fault Preventive systems attempt at nullifying subtle design errors that might encroach. They adopt *design diversity*, to decrease the faults in the system. Fault tolerant systems appeals to the robustness of software. These types of systems increase the reliability by making the software execute in an acceptable manner for spurious inputs. This is achieved by having different versions of the software. Fault Prevention strives to achieve the same by making comparisons of different designs and then taking the union of them. These systems enhance the reliability, but they do not provide any information that justifies the reliability level they claim to have achieved

There are two approaches to assess the reliability of software:

- Static Assessment Analysis.
- Dynamic Assessment Analysis.

Static analysis requires no execution of the program that is being analyzed. Program verification uses mathematical logic to prove program correctness. It proves beyond the element of doubt that the program functions as specified under all possible input conditions, implying all execution conditions.

Dynamic assessment analysis adheres to the more conventional technique of executing the program to check for conformance with specifications. The program undergoing evaluation is subjected to different inputs. The outputs of the program are then compared with the expected results for the tested inputs.

2.1.1 Static Assessment Techniques

The main technique used for static assessment is formal verification. It is based on reasoning whether a program will work in accordance to specifications using mathematical proof checking. There are several theories that have matured over the years and culminated in proper proving techniques. A few of these are Hoare's axiomatic method [34], Floyd's inductive method [35] and structural induction [35][36]. Several textbooks explain these methods in detail [35].

One of the chief drawbacks with formal verification is the lack of mechanized proof verification tools [6]. To prove a program functionally correct, one manually performs the proof. When dealing with high consequence systems, the complexity of the software may be high. Consequently, this leads to two main problems. The proof turns out to be tedious and, abiding by the adage "*to err is human*", erroneous proofs can result.

Another aspect is transcribing the program into mathematical models. With increased complexity of systems and increased complexity of languages, it seems difficult to apply formal techniques to assess the correct functioning.

In principle, reliability of software can be quantified either by formal verification or by statistical testing. The requirements specification of DOLILU II was written in plain English and no attempt was made to formalize it with any form of mathematical notation. Furthermore, the size and complexity of the specification documents make formal program verification virtually impossible. Therefore the assessment of the DOLILU system was done by program testing.

2.1.2 Dynamic Assessment Techniques

Several techniques have been proposed over the years that fall into this category. Some of the notable ones are, fault based assessments and statistical based methods. Voas, Miller and others [38] proposed fault injection techniques to assess reliability. In this method, one physically injects faults into the program. The process is termed *software fault injection*. The software is then tested to determine whether faults can be detected. Intuitively, this form of testing gives us insight into the test coverage obtained from test cases. For more information on this type of testing one might refer to [6][38][37]. When dealing with safety critical systems, which have a tendency almost never to fail, fault-based methods do not provide sufficient knowledge of the reliability of software.

In this thesis, we shall deal with what is termed *black box testing*. In this type of testing we treat the software as a black box and test it statistically until it satisfies the reliability requirements.

Musa proposed, that, to quantify the reliability of the software we need to have a proper, well-defined, approach towards testing. He suggested the use of *operational profiles* to dictate testing [19]. Operational profiles portray the likely field use of the software. He even illustrated how one may develop the operational profile by doing a case study. Exhaustive testing is infeasible and testing alone does not guarantee the absence of faults. The rationale behind Musa's work is, if testing was directed by the operational usage (operations that take place most often) of the software, the likelihood of detecting faults is enhanced and thus, the reliability estimate becomes more realistic.

The major drawback is that it is difficult to determine the operational profile for much software and it is difficult to predict the changes in the estimate of reliability, for changes in the input profile. Interestingly, if one wanted to develop the operational profile for DOLILU II system, it is noted that all the functions of the system need to be exercised. The rules for evaluation may be divided into single point evaluation and a range evaluation (for different parts of the trajectory). We shall now discuss the statistical reliability assessment method.

2.1.2.1 Statistical Assessment Methods

In this thesis, we have used methods for evaluation of reliability of **DOLILU II**. In this method of analysis, there are two approaches, *black box approach* and *white box approach*. In the *black box* approach, the program under test is visualized as a function f that maps all points in the input space into the corresponding points in the output space. No knowledge of the implementation of the program is required. There is another school of thought that uses the knowledge of the implementation (the code) to direct tests. This methodology is termed *white box* testing. Here test cases are derived from the input based on execution path coverage, or specifications. Both these methods divide the input domain is into two portions, failure causing and non-failure causing inputs. We shall now introduce the notion of reliability.

Intuitively, reliability is tied with failures, the more the number of failures the lesser the reliability. If a total of n test runs were conducted and, of these, if n_f inputs led to incorrect results, according to the Nelson model, which derives the relationship between reliability and the number of test cases executed, the estimated reliability R (which is the proportion of right executions) is

$$R = \frac{n - n_f}{n} \quad \text{or}$$

$$R = 1 - \frac{n_f}{n}$$

If number of test runs is large, n->∞, the fraction n_f / n approaches zero, when n_f is significantly smaller than n. The above equation approaches unity. According to this model, the more the test cases that are executed and the fewer failures one observes; the higher is the reliability of the software.

Safety critical systems are characterized by a very high degree of dependability. Development is rigorous in nature and the possibility of faults in design process is reduced. Reliability assessment is done during the validation phase of the software life cycle. Due to the nature of the software, it is unlikely that the software would exhibit failures during final testing. Consider the case when we have executed 1000 test cases and observed no failures ($n_f = 0$), then, according to the Nelson model the estimate of reliability is 1. The fraction n_f / n is the failure probability. Is this a true estimate? The question now arises, how do we predict the failure probability when we do not see any failures?

Statistical assessment is done by selecting inputs randomly from the input domain and testing the software. The foundation for this form of assessment lies in statistical sampling theory [27]. Testing is done by randomly picking out inputs from the input space. From statistical point of view this may be represented as a Bernoulli trial [6]. Reliability assessment assumes sampling with replacement for easier and economical implementation of software testing. Since the input space is astronomical in size, the probability of selecting the same test case twice is minimal.

Classical theory helps predict the probability of failure using the Laplace rule of succession [31]. If one samples the input space which has an unknown number of failure causing inputs and after running t tests on the software, if no faults are detected then the probability of failure in a single run is given by

$$\frac{1}{t+2}.$$

This indicates that failure probability is inversely proportional to the number of test cases executed. This is intuitive as the greater the number of correctly executed test cases the more is one's confidence in the software being tested.

To establish a failure rate of less than 10^{-9} failures per hours we need to test the program for 10^{9} hours. The alternative solution is to make prior assumptions about the quality of software [28][13][3][6]. Chapter 4 in the thesis deals with the Bayesian framework for reliability assessment, which uses prior subjective beliefs to be incorporated into the model.

It is not sufficient if one could predict the failure probability θ, of the program. We must gain confidence that the predicted value of θ does in fact depict the realistic estimate of θ.

Input domain models are plagued by the following shortcomings

- large number of test cases
- The assumption of a test oracle: oracles decide whether the result from a test run is correct or incorrect unequivocally. Building test oracles is difficult. In this thesis, we define premises on how to create a test oracle.
- Reliability estimation depends upon the ability to closely approximate/predict the operational profile of the field use.[41]

Despite these potential problems, a strong theoretical background of sampling theory is the basis for choosing the input domain approach to assessing **DOLILU II**.

There are several techniques suggested in literature for effective testing of the software. They all address the method of test data selection. The most popularly known are code coverage techniques (branch testing and path testing), specification based strategies [16][21], data flow criteria [17][18], and the domain strategy [22]. All these methods share a common characteristic: the program's input domain is divided into subsets called *subdomains*. One or more inputs from each of these subdomains are selected as representatives of the subdomain. This approach is called *partition-testing* [24].

Studies by Duran and Naftos[29] and Hamlet and Taylor[30] showed that there is only a marginal difference in finding bugs between partition testing and random testing.

In the remaining part of the thesis, chapter 4 and 5, we shall discuss the applicability of these methods for the assessment of **DOLILU II** system. We discuss the relative merits and their problems in trying to assess the reliability of the trajectory evaluation system.

3.0 Chapter 3: Day of Launch I-Load Update System (DOLILU II)

The **DOLILU II** system for the Space Shuttle program has been developed for the generation of launch trajectories and to allow modification of the shuttle's first stage guidance commands based on wind conditions (atmospheric conditions) measured hours preceding the launch. The system consists of trajectory software required to generate trajectories and verify guidance commands to recommend decisions on whether to fly or not to fly. It is clear that **DOLILU** is a high consequence system i.e., there is very high cost and a risk of life associated with the eventual occurrence of a failure.

The decision making process may me viewed as consisting of four independent stages:

- Tracking and Wind Profiling stage: forms the first stage in the process. This process assimilates data the from the launch balloons (tracking system), uses interpolation and extrapolation techniques for the generation of wind, temperature and pressure profiles based on altitude and time (wind profiling). This is shown as Tracking and Generate Wind Profile in figure 3.1.
- Trajectory and Guidance Commands stage: there are two processes involved in this stage. They are *Day-of-launch Ascent Design System* (DADS) and *Space Vehicle Dynamic Simulation* (SVDS). DADS, generates the initialization loads (first stage guidance commands) and the launch trajectories for the day-of-launch (DOL) atmospheric conditions. SVDS generates a trajectory for mean monthly atmospheric conditions surrounding DOL with DADS generated guidance commands for these conditions and one for DOL conditions.

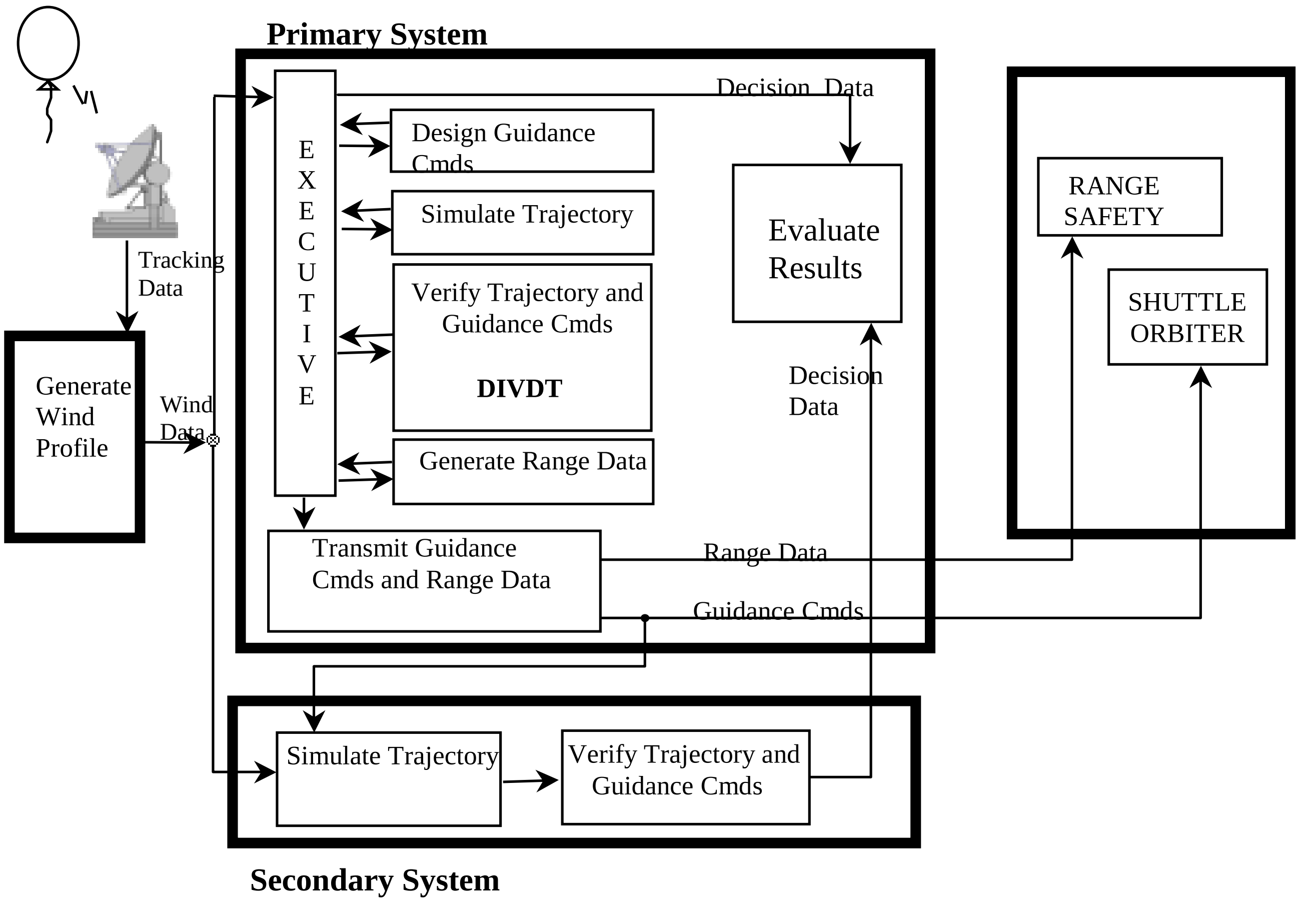

Figure 3.1: Integrated Day-of-Launch I-Load Update (DOLILU II) System Diagram

- Verification stage: this stage forms the penultimate stage in the decision process. This process evaluates wind and trajectory conditions against the guidance commands to ensure the launch would not be fatal. We call this stage the Trajectory Evaluation Stage.
- Experts Team: after the evaluation process is completed by DOLILU II system, a team of experts analyze the results obtained from the system and finally decide whether to make the flight or not.

In order to increase the reliability of the system, NASA adopted *design diversity*, to make the system more fault tolerant. There are two separate computational lanes indicated as *primary* and *secondary* systems in figure 3.1. Guidance commands and flight trajectories generated are deemed valid only if the outputs of both the lanes agree in their analysis. In this thesis, we are interested in the assessment methodologies for the primary system, especially the Trajectory Evaluation System.

3.1 DOLILU II – Primary System

The tracking systems receive the wind and atmospheric data from the balloons released a few hours before the scheduled time of launch. The data is fed into the wind profile generation processor. As the name suggests the primary function of this processor is to generate the profiles in the required data format understandable to the remaining part of the system.

The **TLAMS** (Trajectory and Load Analysis Management System, shown in the diagram as the *executive*) initiates the next sequence of events for trajectory and guidance commands generation and their validation. It invokes the *Day-of-launch Ascent Design System* (**DADS**). DADS processor generates the guidance commands and simulates the trajectories for the day of launch conditions. Guidance commands (I-Loads) and measured winds are passed on to the *Space Vehicle Dynamic Simulation* (**SVDS**) processor. SVDS produces two trajectory files, one is called the reference trajectory (REFTRAJ), and this is simulated for mean monthly wind conditions. It also creates a second trajectory file (DOLITRAJ) which uses the updated I-loads and present wind conditions.

Near real-time verification is the most critical function of the DOLILU system. Successfully simulated trajectories and their corresponding I-loads (guidance commands) are verified for conformance with safety related rules, called envelopes. The envelopes have been derived from previous experience (*experience envelopes*) and known system constraints (*system envelopes*). If any of the system constraint rules are violated then the violation must be reported. A violation, when plotted, would be clearly seen to exceed the system and experience envelopes. Such a flight trajectory is generally deemed invalid and must be dismissed. TLAMS executive invokes the Day of launch I-Load Verification Table (DIVDT) processor, which performs trajectory verification. DIVDT verifies trajectories based on a predefined set of rules[40].

The DIVDT should detect all potentially unsafe flight conditions, verifying the outputs from all the other processors of DOLILU system. Therefore, the reliability quantification of the DIVDT processor is highly desirable.

3.2 Day-of-launch I-Load Verification Data Table (DIVDT)

DIVDT evaluates the trajectory conditions against the guidance commands (I-loads) that are generated. The process of evaluation is done based on rules defined in the Quality Assurance Document [40]. Rules are classified as (in their increasing order of importance)

- System Constraint Rules (S-rules): Any violation is considered being a failure to make the flight. These rules affect the integrity of the system.
- Experience Constraint Rules (E-rules): Design engineers based on their experience with the system determine these rules. They are similar to S-rules but provide some more latitude in their evaluation. They help the experts' team in their final decision E.g. if the percentage of exceedance is not large for a given S-rule and is contained well within limits of the corresponding E-rule then the team may pass the rule.

- Processor Rules (P-rules): these sets of rules are defined for the processor integrity. They deal with the interfaces between the various processors. They check whether the data is conveyed properly between the various processors.
- Abort Region Determinator Rules: special rules that are defined for the safety of the flight.

DIVDT has two modes of execution

- Normal Mode
- QA mode

In the normal mode, it verifies that I-Loads generated by DADS is acceptable for flight by evaluating trajectory parameters in accordance with the DOLILU II Quality Assurance Rules. [40] The QA mode provides rapid quality assurance of a trajectory without a preceding DADS I-Loads generation. The trajectory is generated close to launch using the latest day-of-launch (DOL) environment, generated at an earlier balloon release time. All functions of the QA mode remain the same as normal mode except for the following

- Omission of the DADS-generated trajectory files
- Omission of DIVDT “P” and “A” rules evaluation.
- Omissions of outputs for “P” and “A” rules in the DIVDT output summary.[1]

DIVDT requires several inputs for it proper operation. They are as follows

I. SVDS simulated trajectories

There are two trajectory files that are generated by SVDS

[1] P = Processor rules, A = Abort Region Determinator Rules

- *svds_15_ref*: a reference trajectory, which could be any of the previously created SVDS trajectory files. This helps in the definition of experience envelope. Typically it contains mean monthly environments and pre-launch predicted mass properties. The I-Loads used are DADS generated I-Loads designed for the launch month's mean environments.
- *svds_15*: is SVDS trajectory file that is usually generated with the DADS updated I-Loads and measured winds (*normal mode*).

II. DADS generated trajectories and I-Loads

- *dads_15*: is a trajectory that is internally created by DADS. This is required only in the *normal mode* of execution.
- *dads_iloads*: is required only in the normal mode of operation. Contains the DADS generated I-Loads (guidance commands) and the *ARD (abort region determination)* engine modeling data.
- *ard_divdt*: is created by DADS and contains ARD parameters to be verified by DIVDT. Required only in the *normal mode.*

III. Limits input files

- DIVDTINPUT: limits input file used to input the variable limit values for each rule identified in [40].
- VENTDATA: contains limit arrays for the venting carpet constraint.

During the entire execution of the program there is no user intervention. DIVDT evaluates the rules and provides a summary of the results in ASCII (text summary files) as well as binary data files (plot data files). The evaluation takes place in three stages, first stage of flight, staging (this refers to pre-orbital insertion stage) and orbiting stage

(MECO). Plot files created by DIVDT are passed to the DIVPLT program, which generates plot outputs. The output files of DIVDT are described below

- *Divdt_control*: file containing the flight number, atmosphere balloon release time, wind balloon release time, and up to ten of the highest DIVDT rule violations.
- *Divdt_stdplt*: a file containing plot data for all the rules.
- *Divdt_rsplot:* containing plot data for Range Safety Rule, a system constraint rule.
- *Divdt_thrplt*: containing plot data for Altitude Constraint Rule.
- *Divdt_trjsum*: a file containing trajectory summary and evaluation of parameters at critical points in the trajectory (i.e. first stage, staging, MECO).
- *Divdt_topten*: a top-ten summary of the discrete rules with the highest percentage of exceedance. A summary of the DOLILU I-Loads is also given.
- *Divdt_detail*: a detailed summary of all the rules evaluated with their percentage of exceedance and a verdict of PASS or FAIL for each rule.
- *Divdt_summary*: a summary of the list of rules arranged in the descending order by their percentage of exceedance.

Further information about the quality assurance rules and DIVDT functioning could be obtained from the following documents [39][40].

3.3 Assessment Methodologies for DIVDT

As mentioned earlier, DOLILU II is a high consequence system. Prior to this work there has been no attempt to determine the reliability assessment of the software. Acceptance test cases identified in [43] represent the minimum testing required to evaluate the completeness and accuracy of the software. Testing criteria specifically addresses verification of logic paths, data handling, and design constraints that may be encountered during the normal operation of DIVDT. Tests are performed to verify compliance with functional requirements. After system testing, one of the following decisions must be made

- Acceptance of software: following the designated testing period the formal software configuration will be completed.
- Conditional Acceptance of software: contains conditions that had errata in them. It furnishes information of testing that needs to be done after the errors have been corrected. It is observed that only the corrected regions of the software are re-tested, but from our experience we know that changes made in the software could affect other regions in the program. Hence we need to perform tests on the complete software to ensure that the changes did not create new errors.
- Rejection of software: this is accompanied by the details of the requirements that were not met. Rejected software needs to be re-tested with all tests before formal acceptance.

The test philosophy ensures that all the rules are being exercised. The tests were designed for external interfaces, design, and program logic. Some of the techniques used are enumerated in the following subsections.

3.3.1 Inspection

Inspection is a manual verification technique in which the program (code, data) is examined to discover discrepancies with requirements. Code walk-through and code inspections were some of the techniques adopted here.

3.3.2 Analysis

Analysis involved formal verification methods, to analyze the functions. As these methods required rigorous mathematics, only a subset of the functions was analyzed this way.

3.3.3 Testing

Testing is performed with a specified set of steps, which will result in some expected results. The expected results are also documented along with the test results.

3.3.4 Demonstration

Typically this method is used to gain confidence that software will not abort under certain conditions. Incorrect input file format or a missing file is typical conditions that require demonstration

3.3.5 Comparison Against Results from Previous Version

Regression testing of the software is done to ensure that changes have not adversely altered the function form the previous version. Identical data sets are used to test both the versions and the results are compared.

Software testing is based on the criticality of the part during the evaluation process. Tests were classified into 5 sections,

- External interfaces: deal with the P-rules, external processors communicating with DIVDT. They tested for conformance of data.
- Internal interfaces: deals with testing DIVDT for correct rules evaluation. A combination of the aforementioned techniques was adopted.
- Limit testing: deals with limit based checking

- Top ten testing: manually manipulated trajectory data for a selected 10 rules and checked for confirmation from the DIVDT output summary.
- Worst Value testing.

Further description of strategies may be obtained from [43].

The techniques described were not used to assess the reliability of DOLILU II system. They only stress tested the system for conformance with the specifications. This gave us an opportunity to devise new techniques to assess the reliability of the software.

The next chapter deals with the Bayesian Inference Framework for reliability assessment. In the framework we try to incorporate our belief in the system using the knowledge that it has already been in use and tested. We do this by incorporating subjective probability into our model.

4.0 Chapter 4: Assessment Methodologies

Reliability assessment of systems must be based on precisely defined concepts in order to make comparisons between systems possible and to provide a logical basis to improve the system's reliability [28].

In most life critical applications, one has to establish that software reliability is indeed high. Butler and Finelli [51] classified software systems into three types namely,

• Ultrahigh reliability:	$< 10^{-7}$ failures/hour,
• Moderate Reliability:	$10^{-3} – 10^{-7}$ failures/hour,
• Low Reliability:	$> 10^{-3}$ failures/hour.

Unlike hardware, where reliability is associated with physical faults, software programs have faults induced in them from the beginning (requirements documents) to through out the life cycle. Nevertheless, when we view the system as an entity, subjecting it to inputs and observing the outputs, the system either produces a correct or an incorrect result. This may be viewed as a stochastic process, where the software produces errors in a stochastic manner [51]. Based on this observation we may now define software reliability:

Software reliability is the probability of failure free executions of the software, over a given period of time and within a specified environment.

Consider the following simplistic model for modeling software reliability. The software is subjected to external inputs. The program is viewed as a black box function f that maps the inputs to the corresponding points in the output domain. The same software is repeatedly executed for each of these inputs. There exist only two possible outcomes, either the software executes correctly and produces the right output, or the software executes incorrectly and produces an incorrect result. If we assume a constant failure rate (per input), say θ, then testing can be modeled as a binomial process. The number of failures F after n inputs is then given by

$$P(F = r) \quad = \quad {}^{n}C_{r}\ \theta^{r}\ (1-\theta)^{n-r} \tag{4.1}$$

where r represents the total number of failures observed for n inputs. The term ${}^{n}C_{r}$ is an alternative form for representing combinations, computed as $\begin{pmatrix} n \\ r \end{pmatrix}$.

We need to determine the probability of system failure for n inputs, failures can occur for all F>0. Therefore, using (4.1),

$$\begin{aligned} P(n) &= P(F > 0) = 1 - P(F = 0) \\ P(n) &= 1-(1-\theta)^{n} \end{aligned} \tag{4.2}$$

With our assumption of constant failure rate over time, we may represent n in terms of time; $n = kt$ where k = number of inputs/unit time then

$$P(t) \quad = \quad 1-(1-\theta)^{kt} \tag{4.3}$$

With kt being large in comparison with θ, $(1-\theta)^{kt}$ may be approximated as $e^{-\theta kt}$. This is obtained from the Poisson approximation to binomial discrete distribution. Hence some researchers assume that the time to failure distribution is exponential.

Traditionally, the method of software creation involves a cycle, where the software is created, then sufficiently tested. When an error is found, the bug is fixed and the software is tested again. This forms the basis for “Reliability growth models” (RGM). The goal for these models is to fit mathematical models to predict what would be the estimated reliability of the final version of the software, based on inter-failure time observation. There are numerous RGMs, the reader may refer to [6] for a concise description of these.

There are several problems in potentially applying reliability growth models for assessment of DIVDT (DOLILU II). When considering reliability growth models, the mean time to failure increases, (shown to exponential by Musa's model, others used log-linear distribution [51]) and consequently the failure probability decreases as more and more bugs are fixed. However, what should be an acceptable time frame before the model can achieve ultrahigh reliability? Miller and Keiller [52] stated that the time frame would be prohibitively large. Another problem is when the software itself doesn’t exhibit any failures. This is typical of safety-critical software. How could one fit a distribution with no observed failures during acceptance testing? We require a model that overcomes these difficulties.

4.1 Bayesian Inference Framework

Rev. Thomas Bayes’ paper first published in 1763 provides the basis for “Bayesian Statistical Inference”. Due to its fundamental importance, the paper was re-published in 1958 [28,45]. Several factors have contributed towards the recent resurgence and wide scale acceptance of the theory.

The cornerstone of Bayesian inference is the notion of subjective probability. Such a notion contrasts with the well-perceived notion of frequency for probability estimation. The axiom of probability states that the probability of an event has to be estimated by determining the value of success ratio. To progress towards this empirical estimation, one has to conduct trials repeatedly, in which the event occurs.

Subjective probability deals not only with the events but with propositions as well. A proposition is considered as a collection of events that contribute towards the estimation based on previous events, observed behavior or the reflection of one’s belief in the system. In statistical terms, we *hypothesize* that the event does occur with the estimated probability. As evidence increases relevant to the hypothesis, we then change our *degree of belief* in the hypothesis. Interestingly, some argue that subjective probabilities assigned to a particular hypothesis may indeed be quite individualistic [28]. In other words, the probabilities assigned by different individuals would reflect different beliefs yielding different results. Bayesian inference theory circumvents this in the posterior analysis where our degree of belief changes with the observations made. However, egregious probability assumptions are definitely not permissible.

4.1.1 Classical Probability Theory versus Bayesian Inference

There are distinctive differences between the classical theory and Bayesian methods of inference. In classical theory approach, the unknown parameter to be estimated (θ) is assumed to be a fixed constant. A point estimator, which is a function of the data set (observed) is chosen according to some principle such as minimum variance, least squares or the method of moments. Classical theory inferences are then made using inductive reasoning. The Classical method of inference is depicted in Figure 4.1. The process begins with the postulating of a sampling model. Inductive reasoning is used in conjunction with the sample observations to produce inferences about the unknown parameters.

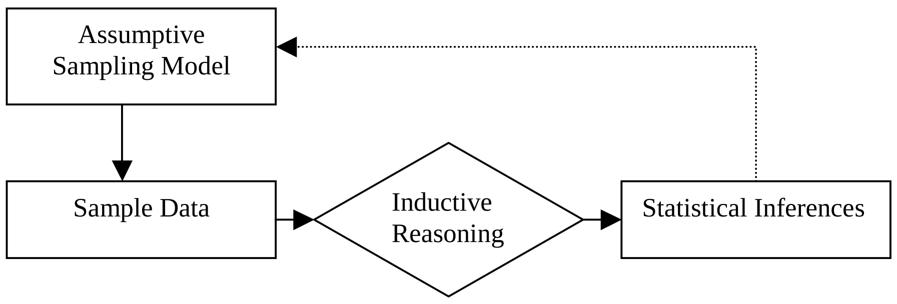

Figure 4.1: Classical Model for Inferences

Bayesian method of reasoning is deductive. The parameter of interest (θ) is assumed to be a random variable with *a priori* distribution *g(θ)*. This distribution expresses the assessor's state of knowledge or ignorance about θ before the sample data, say *y*, is analyzed. Given the prior distribution, and the data set *y*, Bayes' theorem is used to calculate the *posterior distribution g(θ|y)*.

The prior distribution in a Bayesian analysis usually embodies a subjective notion of probability. It is the distribution of degree of belief about θ before the observational data (*y*) is obtained. A distinctive feature of Bayesian inference is that it takes explicit account of prior information in the analysis. This contrasts with the classical approach of sampling theory. Figure 4.2 depicts the Bayesian method of inference. The process begins with a postulated sampling model. A prior probability distribution is also assumed. The sample data and the prior distribution are combined by the use of Bayes' theorem (explained in section 4.1.3). Deductive reasoning is then used in conjunction with the resulting posterior distribution to produce the desired inferences about the parameters of the assumed sampling model.

There are two further distinctive differences between sampling theory and Bayesian procedures. Bayes' use of relevant past experience, which is quantified by the

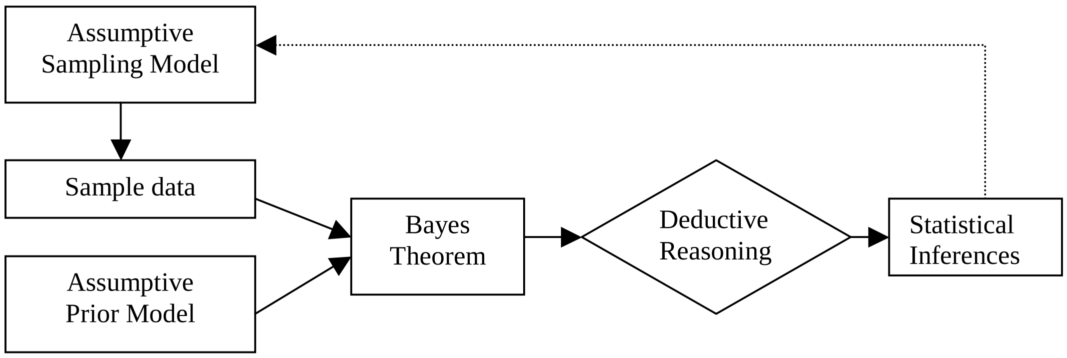

Figure 4.2: Bayesian Inference Model

prior distribution, produces inferences that are more informative. The second distinction is that the Bayesian method usually requires less sample data to achieve the same quality of inferences than methods based on sampling theory.

4.1.2 Advantages of Bayesian Inference in Software Reliability

Software reliability estimation methods based on sampling theory have been found useful for a variety of problems. There are, however, many instances in which the classical methods have been found to be less than satisfactory. There has been an ever-increasing demand for cost-effectiveness in reliability assessments of systems, typically safety critical systems. Sampling methods for software reliability estimation cause problems when based on scarce failure data. When observing the failure rates of software for commercial nuclear power plants [28], it was observed that the mean time to failures is $t = 7.9 \times 10^6$ h. As a consequence it is not possible to determine a two-sided confidence interval estimate on a constant failure rate, assuming an exponential failure time model. The point estimate would be zero, an overly optimistic estimate.

As another example, consider estimating the failure rate based on sample data consisting of zero observed failures in many reactor years of commercial operation. We are faced with the same situation as in our preceding example. These examples are reflective of problems in estimating the reliability of DIVDT (DOLILU II) software since no failures have been observed during test executions. We do know that the software has been successfully operating over the past five years. How does one incorporate this information into the estimation model? In situations like these, the methods based on sampling theory are frequently replaced in favor of more useful methods, like the Bayesian approach.

As mentioned earlier there are two important practical benefits in using Bayesian analysis. One is the increased quality of inferences, provided the prior information incorporated in the model reflects the true variation in the parameter(s). The other is the reduction in testing requirements. There is yet another advantage. Inferences that are inaccurate arise from incorrect assumptions and not from inadequacies of the method used to provide them.

Bayesian methods provide a satisfactory way of explicitly introducing and organizing assumptions regarding prior knowledge or ignorance. These assumptions lead, via Bayes' theorem, to posterior inferences about the reliability of parameter(s) of interest.

4.1.3 Bayes' Theorem with Subjective Probabilities

Bayes' theorem is the fundamental tool used to arrive at Bayesian inferences. Let θ denote the parameter of interest that we would like to estimate. The prior model represents the subjective information available about θ before the observation of the sample data x.

$$\begin{aligned}&x = obsevations\ on\ the\ sample\ data.\\&g(\theta) = the\ prior\ probability\ distribution.\\&f(x\,|\,\theta) = the\ conditional\ probability\ distrbution\ of\ x\ given\ \theta.\\&g(\theta\,|\,x) = the\ posterior\ distribution\ of\ \theta\ given\ x.\end{aligned}$$

The posterior model tells us what is known about θ given the knowledge of the data x. It is intuitive that the posterior model should represent an updated version of our prior knowledge of θ. If the data supports our belief, there should be an increased confidence in the subjective notions. On the other hand if the sample data does not support the subjective information, the posterior model should give a weighted consideration of both assessments, the sample data and prior. This is achieved by using Bayes' theorem. The posterior distribution is given by

$$g(\theta|x) = \frac{f(x|\theta)g(\theta)}{f(x)},$$

where *f(x)* represents the marginal distribution and may be obtained by

$$f(x) = \int f(x|\theta)g(\theta)d\theta.$$

4.2 Bayesian Reliability Assessment of DIVDT (DOLILU II)

The primary reason in adopting the Bayesian approach is to incorporate in our reliability assessment model the knowledge that the system has already been in use and, prior to that, rigorously tested. Furthermore, the techniques adopted for the creation of the software shows that the system was indeed developed using proper software engineering practices.

4.2.1 Choice of Prior Distribution

There are several papers in literature [13,46] and books [28,47] that have provided guidelines in choosing a proper prior distribution. For our framework, we chose beta distribution to accurately reflect prior beliefs. There are two primary reasons in choosing this distribution.

i. By proper choice of the parameters, it is possible to depict any type of distribution that is actually exhibited by the system.

ii. The distribution forms a conjugate family. The conjugate family has the property that both the prior and posterior distributions will be members of the same parametric family of distributions [46,47]. Intuitively this represents a kind of homogeneity in the way in which our beliefs are represented, and how they change as we receive extra information [13].

The intention here is to devise a framework that attests that if the system executes *n* demands without failure then it is deemed to have achieved the required confidence in reliability estimate. Within the Bayesian framework we represent our *prior* knowledge about the parameter of interest, here the probability of failure on demand denoted as θ, by the prior distribution. The prior distribution from the conjugate family is the beta distribution

$$f(\theta) = \frac{\theta^{p-1}(1-\theta)^{q-1}}{B(p,q)} \tag{4.4}$$

where *B(p,q)* is the beta function with *p>0, q>0*, *p* and *q* represent our prior belief in θ for the software under test. Assuming *ignorance prior* implies that it is equally likely to

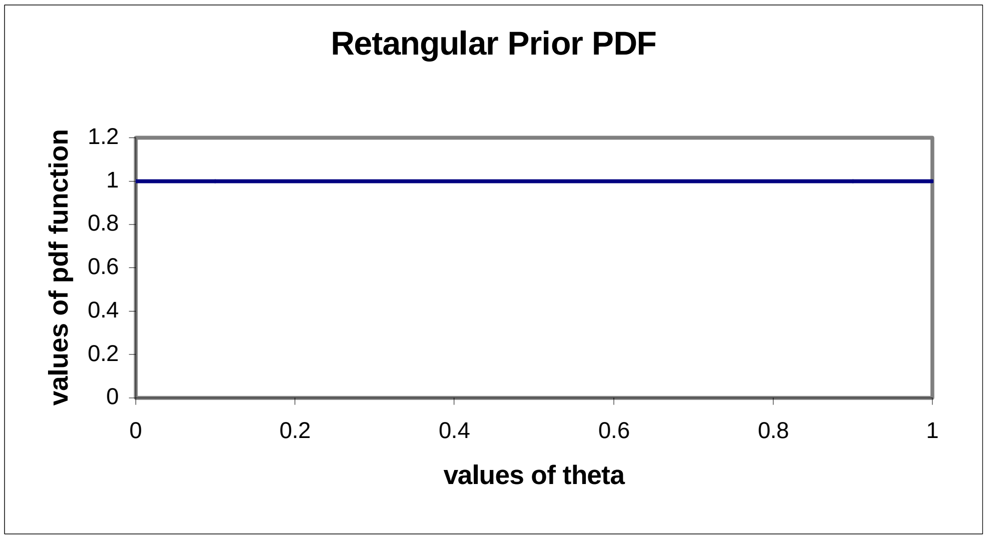

Figure 4.3: Graph for *beta(p,q)* with p=q=1, equally likely distribution

have any value of θ in the range 0-1. If we set the values of *p* and *q* as *p*=*q*=1 and substitute in equation (1) we obtain a *f(θ) = 1,* rectangular probability distribution as in Figure 4.3.

The goal of reliability assessment is not just to estimate the failure probability but to gain statistical confidence that the estimate is indeed realistic. In practice the required failure rate θ and the confidence level C are usually predefined. The question is how much testing needs to be done?

Let T be a random variable denoting the total number of test cases that need to be executed until the first failure is detected. To achieve a required confidence, an unknown number of test cases U needs to be executed such that

$$\Pr ob(T \leq U) = C. \tag{4.5}$$

Considering the classical sampling theory for estimating, the distribution of T is assumed geometric and the probability that T assumes a particular value t is given by

$$\Pr ob(T = t) = \theta(1-\theta)^{t-1}. \tag{4.6}$$

Combining equation (4.5) and (4.6), we get the equation relating U and C.

$$\Pr ob(T \leq U) = \sum_{t=1}^{U} \theta(1-\theta)^{t-1}. \tag{4.7}$$

$$\sum_{t=1}^{U} \theta(1-\theta)^{t-1} = C. \tag{4.8}$$

The left-hand side of the equation (4.8) is a geometric series and can be computed as

$$\theta \sum_{t=0}^{U-1} (1-\theta)^{t} = \theta \left(\frac{1-(1-\theta)^{U}}{1-(1-\theta)} \right). \tag{4.9}$$

Substituting (4.9) in (4.8) we get

$$(1-(1-\theta)^{U}) = C \tag{4.10}$$

Solving equation (4.10) for U we get

$$(1-\theta)^{U} = 1 - C\,. \tag{4.11}$$

Taking $\log_e$ on both sides, the required number of test cases U, is therefore

$$U = \frac{\ln(1-C)}{\ln(1-\theta)}. \tag{4.12}$$

In the Bayesian framework of assessment if the system has executed *n* demands and we have seen *r* failures, we get posterior distribution of *f(θ)* to be [46,47].

$$f(\theta | n, r, p, q) = \frac{\theta^{p+r-1}(1-\theta)^{q+n-r-1}}{B(p+r, q+n-r)}. \tag{4.13}$$

Assuming ignorance prior, we get p=q=1, therefore

$$f(\theta | n, r, 1, 1) = \frac{\theta^{r}(1-\theta)^{n-r}}{B(1+r, 1+n-r)}. \tag{4.14}$$

If we require *U* demands before detecting the first failure then *n=U, r=0,* therefore

$$f(\theta | U, 0, 1, 1) = \frac{(1-\theta)^{U}}{B(1, 1+U)} \tag{4.15}$$

*B(*1,1+*U)* is the complete beta function. The generalized form is

$$B(a,b) = \int_0^1 \theta^{a-1}(1-\theta)^{b-1} \tag{4.16}$$

Therefore *B(1,1+U)* is given by

$$B(1, 1+U) = \int_0^1 (1-\theta)^{U} d\theta = \frac{1}{1+U} \tag{4.17}$$

It is required that the failure rate θ should be less than the pre-mediated value p_0 with confidence level *C*. Now *Prob(θ<p₀)* is the cumulative density function given by

$$\Pr ob(\theta \le p_0) = \int_0^{p_0} f(\theta | U, 0, 1, 1) d\theta. \tag{4.18}$$

Mathematically we represent the above statement as

$$\Pr ob(\theta \leq p_0) \geq C. \tag{4.19}$$

Substituting equation (4.17) in (4.18) we get

$$\int_{o}^{p_0} f(\theta | U,0,1,1) \geq C. \tag{4.20}$$

Substituting for *f(θ|U,0,1,1)* from equation (4.14) we get

$$\int_{0}^{p_0} \frac{(1-\theta)^U}{B(1,1+U)} \, d\theta \geq C. \tag{4.21}$$

Combining equations (4.20) and (4.16) and integrating we get

$$\int_{0}^{p_0} (1+U)(1-\theta)^U \, d\theta \;\; = C. \tag{4.22}$$

The solution for the above equation yields the result in the following form

$$(1+U)\left[\frac{(1-\theta)^{1+U}}{(1+U)(-1)}\right]_0^{p_0} \;= C. \tag{4.23}$$

Simplifying equation (4.22) we get

$$1-(1-p_0)^{1+U} = C. \tag{4.24}$$

Taking log$_e$ on both sides and solving equation (4.23) for *U*, we get the total number of test cases required to achieve with defined confidence C that the reliability requirement p$_0$ is satisfied as

$$U = \frac{\ln(1-C)}{\ln(1-p_0)} - 1. \tag{4.25}$$

Comparing equation (4.24) and (4.12) we realize that they are the same. It is clear that since *p*=*q*=1 provide no knowledge, they do not influence the interpretation of the test results. Hence, both Classical and Bayesian theory require almost the same number of cases in the absence of prior knowledge of failure distribution.

Now we shall introduce our prior belief into the framework. Before we do this we need to justify our claim in the prior distribution. Two main reasons encourage our belief

- The software has been in use for the past couple of years. As already discussed in chapter 3, several methodologies, partial correctness proofs, inspection, code walk-through were already adopted in testing the software piecewise. However we would like to reiterate that there has been no attempt in assessing the reliability of the software.
- A rigorous development process was adopted and two different versions of the software were developed for added redundancy adding to the fault tolerance of the system.

Considering the above factors, especially the fact that the software has been operational failure free for more than five years, rigorously tested and stringently developed, we could safely assume that the software at least exhibited a failure probability of 10^{-3} failures/h. According to [13,28,47] we could assign the values to *p* and *q* to reflect this belief. We get *p*=8 and *q*=9850 (see Appendix B Table B.1) Now our complete beta function is (taking *a=p* and *b=q+U*)

$$B(p, q + U) = \int_0^1 \theta^{p-1}(1 - \theta)^{q+U-1} \tag{4.26}$$

Repeated integration by parts and simplification yields (see Appendix B Solve B.1)

$$B(p, q + U) = \frac{(p - 1)!}{(q + U)(q + U + 1)\dots(q + U + p - 1)} \tag{4.27}$$

Taking *n=U*, *r=0* (no failures observed), equation (4.12) reduces to

$$f(\theta | U, 0, p, q) = \frac{\theta^{p-1}(1 - \theta)^{q+U-1}}{B(p, q + U)}. \tag{4.28}$$

In equation (4.18) we substitute (4.27), we get

$$\int_0^{p_0} f(\theta | U, 0, p, q) \geq C \tag{4.29}$$

$$\frac{(q+U)(q+U+1)\ldots\ldots(q+U+p-1)}{(p-1)!}\int_o^{p_0}\theta^{p-1}(1-\theta)^{q+U-1}d\theta = C \tag{4.30}$$

Integrating equation (4.27) by parts and simplifying we get

$$1-\frac{(q+U+p-1)(q+U+p-2)\ldots.(q+U+1)}{(p-1)!}\theta^{p-1}(1-\theta)^{q+U}-\frac{(q+U+p-1)\ldots..(q+U+2)}{(p-2)!}\theta^{p-1}(1-\theta)^{q+U+1}-$$
$$\frac{(q+U+p-1)(q+U+p-2)\ldots(q+U+3)}{(p-3)!}\theta^{p-3}(1-\theta)^{q+U+2}-\frac{(q+U+p-1)(q+U+p-2)..(q+U+4)}{(p-4)!}\theta^{p-4}(1-\theta)^{q+U+3}-\ldots\ldots.=C$$

We used MathCAD to solve equation (4.28). We substituted the values for p_0 = 10^{-3}, 10^{-4}...10^{-7} with C = 0.99. Numerical solutions obtained for the above equation is given in table 4.2.

A comparison between the total number of test cases required by random sampling and the equivalent number of test cases required using the Bayesian framework is shown in tables 4.1 and 4.2 and figure 4.7 and 4.8

Table 4.1: Number of test cases from Random Sampling (C=0.99)

Random Sampling	
Values of θ	Number of test cases
10^{-3}	4,604
10^{-4}	46,052
10^{-5}	460,516
10^{-6}	4,605,168
10^{-7}	46,051,700

Table 4.2: Number of Test Cases using Bayesian Framework (C=0.99), assuming θ=10-5

Bayesian Method	
Values of θ	Number of test cases
10^{-3}	950
10^{-4}	22,052
10^{-5}	260,780
10^{-6}	4,295,000
10^{-7}	41,541,171

In [13] Bev Littlewood states that in order to assess the reliability in a Bayesian Framework, we first need to believe that the system indeed exhibits the proposed failure rate before testing. The posterior analysis would then endorse this belief in our system from the testing results.

Table 4.3: Number of Test Cases using Bayesian Framework (C=0.99)

Bayesian Method	
Values of θ	Number of test cases
10^{-3}	950
10^{-4}	2678
10^{-5}	9436
10^{-6}	20796
10^{-7}	51987

When testing reveals failures it translates into our posterior model by decreasing our belief that the probability of failure has been achieved. Consider equation 4.14, when testing reveals failures, the value of r in the equation increases. This decreases the power for the factor $(1-\theta)$ and increases the power for the factor θ. This shows that the new distribution now decreases our initial belief. If no errors were detected, then the distribution changes with an increase our original belief (as the power of $(1-\theta)$ now increases).

In table 4.3, we compute the total number of test cases required before the first failure occurs to ensure with a confidence of 0.99 (having a prior belief obtained from Table B.2 in Appendix B for each θ) that the software did indeed exhibit the proposed failure probability.

It is clear from the tables, that in order to obtain a reasonable time frame to test a system, we need to believe that the system did indeed achieve the proposed degree of failure rate before testing. The purpose for testing is to endorse this belief. If the system did not achieve the proposed reliability, then this is reflected in our posterior analysis.

The cornerstone for Bayesian Inference is the subjective knowledge of the system that is incorporated into the assessment model. One has to understand how one might provide values for the priors p and q.

Consider Table B.3 is Appendix B. This shows the variation in the number of test cases when we try to decrease the amount of variation in our prior model. As one traverses the table downwards, this is indicative of decreased variance in our prediction and the consequent increase in the number of test cases.

Hence, if we need a greater degree of confidence in our prior beliefs itself we need to execute more number of test cases.

4.2.2 Extended testing with occurrence of failure

The above subsection predicts the total number of test cases that needs to be executed correctly before we deem that the software has indeed achieved the required failure probability at the required confidence. How should one proceed in the eventuality of a failure? When using the Bayesian Framework in reliability assessment, we could use the information that testing has yielded until the first failure was observed [46]. To predict how many future test cases need to be done after the failure has been revealed we could use the posterior distribution obtained until the first failure occurred. Let us assume that a failure occurred after the execution of s demands ($s < n$ the estimated number of demands). The posterior for θ immediately following the failure on the sth demand

$$f(\theta | s,1,p,q) = \frac{\theta^{p}(1-\theta)^{s-2}}{B(p+1, s+q-1)} \tag{4.31}$$

This forms the prior distribution for θ for further testing that needs to be conducted. We now need to compute U_1 the total number of failure-free executions required for the software to exhibit the required probability density function; this is

$$f(\theta | U_1 + s, 1, p, q) = \frac{\theta^p (1-\theta)^{U_1+s+q-2}}{B(p+1, U_1 + s + q - 1)} \tag{4.32}$$

Notice that this is simply the posterior distribution after seeing both *(s-1)* failure free executions followed by a failure, and then seeing U_1 further failure free demands. This posterior distribution will be the same whenever the single failure occurred among $s+U_1$ demands: it depends only upon the total number of demands, and the number of failures. Now we may compute U_1 for which

$$\int_0^{p_0} \frac{\theta^p (1-\theta)^{U_1+s+q-2}}{B(p+1, U_1 + s + q - 1)} \geq C \tag{4.33}$$

In general, if we have seen in the r^{th} failure and the failures occurred on the s_1th, $(s_1+s_2)^{\text{th}}$, …….,$(s_1+s_2+s_3+\ldots..+s_r)^{\text{th}}$ demands we should require an additional U_r demands executed failure-free, such that

$$\int_0^{p_0} \frac{\theta^{p+r} (1-\theta)^{\sum_0^r s_a + U_1 + q - r}}{B(p+r, U_1 + \sum_0^r s_a + q - r)} \geq C \tag{4.34}$$

Having developed a framework to assess DIVDT (DOLILU II), we need to address two important issues when estimating software. How does one accelerate the execution of software in order to achieve an agreeable period for testing the software? How does one generate the input conditions to test the system?

5.0 Chapter 5.0: Acceleration of Test Cases

After establishing a framework for reliability assessment of DIVDT (DOLILU II) using Bayesian inference, we were faced with another unique problem that required immediate attention. The time taken for the execution of each test run of the software is too long given the amount of testing required. Typically DIVDT (DOLILU II) takes on an average (assuming a normal mode of execution for Day of Launch I-Load Verification Data Table (DIVDT)) 20 to 25 minutes [43] for a single run. Despite the reduction in the number of test cases, around 200000 test case executions still need to be performed. This would amount to around 9-10 years of execution time to certify with 99% confidence that DIVDT (DOLILU II) indeed achieved ultrahigh reliability!! Consequently we required a reduction in the time for each test case execution.

5.1 Transformations for Accelerated Execution of Test Cases

Program transformations have been used for code modifications especially in code optimizations in order to enhance certain desirable properties such as performance or portability. The application of a transformation to a given program is a three-step process:

- Decision: which part of the program do we apply the transformation and what type of transformation are we going to use.
- Verification: ensure that the transformation doesn't change the meaning (semantics) of the program or if it changes it is done in a restricted manner that is acceptable.
- Actual transformation of the program.

Transformations can be effective only if it is possible to discern whether there are benefits in their application. We expect benefits in the speed of execution of the program.

Semantically equivalent transformations preserve the exact meaning of the program under transformation. In other words, the transformed program does operations in exactly the same sequence as the original program. It is important to note and quoted from [50], that, “semantic transformation is the property of the program execution and not the program”. By that, we mean, if the transformed program executes correctly for some input we can conclude the original program also executes correctly. If the transformed program executes incorrectly for some other input, we can justifiably claim the same for the original program.

The above transformations are based on source code. One may even speed up testing by changing other parameters of influence. E.g. environment conditions can be changed, like using a faster processor, or opting for centralized parallel processing to distributed computing. These types of transformations are termed configuration transformations.

In this thesis, whenever we refer to a transformed program, it may either refer to a source-to-source transformation with unchanged environment conditions, or only configuration transformations or a combination of both.

5.2 Source Transformations

Source-to-source transformations are based on making changes to the software to increase the speed of operation of the program. We shall now discuss the different techniques that we tried to adopt to decrease the execution time of the software.

5.2.1 Vertical Slicing

Process control programs are usually very large complex programs constructed by composing smaller components like procedures; user defined data types and others. They usually compute several output parameters. When developers/users attempt to understand and manipulate programs, they achieve this by decomposing the program. This general observation led to the concept of program slices first introduced by Weiser [48]. It is a straightforward method of decomposing a program into different data flow blocks, by analyzing control and data flow. The original algorithm generated static program slices from data flow graphs. A static program slice extracted all the statements that affect a variable or a subset of variables in the program. A static slice groups statements that directly or indirectly affect the value of a given output variable or output variables. The program slice constitutes a separately executable program, which preserves a specified projection of the original program's behavior, viz., it computes a subset of the original program's output variables. This form of slicing is referred to as "vertical slicing" as it decomposes programs into data blocks in the direction of data flow and control [6][50].

The interesting feature of this slicing technique is that, each program slice is capable of executing and reproducing the exact behavior of the original program within the specified sub-domain in which the slice is defined.

From the point of view of speed-up gain we have three major consequences by using the slicing technique

- Potential reductions in the number of program statements in each individual slice, leading to smaller, faster executable programs.

- Each slice is independently executable and consequently this makes parallel execution of all the slices feasible. Furthermore, when run concurrently the union of the outputs of the all slices forms the output of the original program.
- Consider when some of the vertical slices could be formally verified. The execution of these is deemed unnecessary. Hence, we could reduce the number of vertical slices that needs to be executed.

5.2.2 Reducing numerical precision

This monotonic transformation technique is based on the fact that, double precision arithmetic is more CPU intensive than single precision arithmetic [50]. The same applies between single precision and integer computations.

When programs are decomposed using vertical slicing, each slice can be further transformed to hasten the computation process. We can transform all double declarations to floats. This transformation is applicable only if the slice itself isn't computationally sensitive. The definition of slice functions that are not sensitive to computational precision is borrowed from [50].

> *A program function slice is not sensitive to computational precision if and only if small fluctuations of the input variables produce a small fluctuation in the output variables.*

During the execution of the program slice, if a disparity occurs, the original slice is run to compare whether a fault did occur or not. Program slices with sensitive slice functions are tested with the original slices.

When it is known that there is a requirement for only a fixed number of significant digits, we may manipulate the calculations to be integer computations. We replace the decimal computations with integer computations by multiplying by a factor of 10^n. After the computations are done, we can determine the final value by dividing the final result by 10^n. However fractional computations may lead to erroneous results.

5.3 Environment Changes

Changing environment conditions in which the program is being tested can further accelerate testing. One method would be by running on a faster processor. It is important to note that the processors used should be "binary compatible" (representation of data types and executables should be alike).

5.4 Transformations applied for DIVDT (DOLILU II)

We shall now apply the aforementioned methods to decrease the total execution time of DIVDT.

5.4.1 Applying Vertical Slicing to DIVDT (DOLILU II)

DIVDT evaluates trajectory conditions with the generated I-Loads (initial guidance commands) based on pre-defined rules [40]. When decomposing the software, it became apparent that we should make the decomposition rules based. This is further reassured when one observes output from DIVDT. Every rule is evaluated based on trajectory parameters and a verdict is given as a PASS or FAIL for each rule. A list of all the rules evaluated as adapted from [40][43][42] is given in Table 5.1.

Table 5.1: DOLILU II Quality Assurance Rules

Rule No	Title	E6	Roll, Pitch and Yaw Actual Accelerations
S1	Pitch and Yaw I-Load within SAIL Envelope	E7	Roll, Pitch and Yaw Actual Body Rates
S2	Staging Dynamic Pressure	E8	SSME Pitch and Yaw Gimbal Commands
S3	Staging Angle of Attack	E9	SRB Rock and Tilt Gimbal Cmds
S4	Staging Angle of Sideslip	E10	Pitch and Yaw I-Loads Within

			Parameter Experience
S5	Staging Roll, Yaw and Pitch Rates	E11	Elevon Hinge Moment Experience
S6	SRB Apogee Constraint	E12	(deleted)
S7	Elevon Hinge Moment System	E13	(deleted)
S8	Venting Carpet Constraint	E14	Staging Velocity
S9	Heating Carpet Constraint	E15	Altitude Rate at SRB Separation
S10	(deleted)		
S11	(deleted)	P1	(deleted)
S12	(deleted)	P2	(deleted)
S13	Throttle-Altitude Constraint	P3	(deleted)
S14	Range Safety	P4	(deleted)
S15	Margin	P5	(deleted)
S16	Roll, Pitch and Yaw Actual Body Rates (Post SAR)	P6	(deleted)
S17	(deleted)	P7	(deleted)
S18	Flight Control System	P8	DADS Convergence Check
S19	DADS Wind I-Loads Within SAIL Envelope	P9	(deleted)
S20	TREF Within SAIL Envelopes	P10	SVDS/DADS I-Loads Comparison
S21	Throttle I-Loads within SAIL and Certification Envelopes	P11	I-Loads File Validation
S22	AGT Occurrence	P12	SVDS/DADS Weights @ 150 Seconds Comparison
E1	Staging Gamma		
E2	Staging Altitude	A1	ARD DELT
E3	Staging Azimuth	A2	ARD AGT
E4	Roll, Pitch and Yaw Guidance Altitude Errors		
E5	Roll, Pitch and Yaw Commanded Body Rates		

Before we proceed to apply vertical slicing techniques for speed enhancement, we shall define a few terms that would aid in our explanation of the data flow proofs provided in Appendix A. A program slice is defined as a set of all program statements for a relevant computation. A slicing criterion specifies the slice (computation) for a variable, *v*, at statement *n*. Program slices for a given criterion are obtained by deleting zero or more statements from the given program *P* but still computing the same value for *v* at statement *n*.

An important pre-condition for applying vertical slicing is to provide data flow proofs for each slice establishing complete independence and semantic integrity of each individual slice. Data-flow proofs are given for each rule in Appendix A.

There is a great degree of redundancy in all the vertical slices. This is because even before the execution of rules for evaluation one does checks for input parameter consistency and a variety of checks for correct formats.

Having established the independence of all the slices our model of execution is as shown in figure 5.1. We need to replicate the data across all the processes. Certain processes like evaluation of P8, P9, P11 may be eliminated as they only check for convergence checks. These can be formally verified.

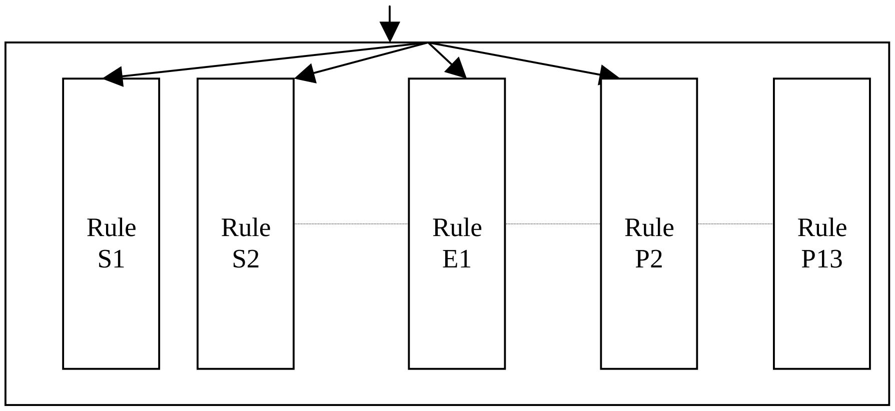

Figure 5.1: Parallel Execution of DIVDT

5.4.2 Applying Numerical Precision Reduction for DIVDT

After consulting with NASA personnel and as documented in [43] for testing purposes it was sufficient if the precision was considered upto 5 significant digits. We transformed all double declarations to floats. As mentioned earlier this method is applicable for slices that are not sensitive computationally. As most of the rules evaluated are based on flow dynamics small changes in the input does not produce large changes in the outputs.

As the number of significant digits were fixed we could use the transformation to integer computations by multiplication by 10^n and division later by 10^n. This could further help the computation process.

5.4.3 Environment Changes

Now that the possibility of parallel execution has been established, we could opt for distributed computing or parallel machine. When changing environments we need to ensure binary compatibility. Hence we need to choose binary compatible computers during distributed computing. Hence we could chose a parallel machines as this reduces latency delays in networks and all the processors are binary compatible.

In the next chapter we shall discuss the generation of input conditions for automated testing of DIVDT.

6.0 Chapter 6: Automated Test Case Generation

Several papers have been published on automated test case generation for testing software. They are usually based on criteria like specifications [16] or data flow graphs [17][18]. Data flow methods usually give a comprehensive logical path coverage.

In this thesis we develop an automated test strategy for testing the Trajectory Evaluation Stage (DIVDT) of DOLILU II system. DIVDT does the evaluation by exercising one function per rule to determine if the input files are indeed valid flight trajectories. Conceptually, there are only two types of execution:

- Either an incorrect data in any of the input files terminates the evaluation process and DIVDT displays an appropriate message, or
- The evaluation of all the rules is successfully completed.

To automate generation of test cases, we need to generate input conditions. Even though our primary aim was to test DIVDT, we could not simulate the required trajectory files generated by DADS and SVDS (If we could simulate it then we could replace the functionality of DADS and SVDS!!). Hence we needed to generate inputs to the system, namely atmospheric conditions.

6.1 Basic Philosophy

The idea behind this methodology is simple. We first analyze the data (the observations) and then attempt to fit models to predict/interpolate data. In order to achieve this we needed to first establish interactions between the factors that comprise the system. We chose one of the factors to be the independent variable and determine others based on changes to this variable.

The algorithm is as follows

- Assimilate data
- By the rule of the thumb, chose 80% of the data to fit the model and the remaining 20% to evaluate the model.
- Determine first, the possibility of liner relationships between factors, considering two factors at a time and then three (line on plane).
- Remove any outliers that might exist. These produce erroneous models.
- Fit models, first starting with linear models. We should get a fairly good idea in previously mentioned step about linearity between factors.
- Determine goodness fit for the model chosen
- Fit “mutators”; mutators change the independent variable and then predict other values based on this change to simulate a new input suite.

In our model we have four factors of interest, namely, pressure, temperature, wind and direction of wind. We chose pressure as the independent variable as it had a near linear relationship with altitude for the first 2km of the atmosphere and later on a near exponential curve. We then establish the models to predict temperature and wind speeds based on changes in pressure.

6.2 Regression Models, Surface Models Analysis

We applied regression methods for establishing predictor functions between the various factors. In this section we will introduce regression models that we analyzed and applied to automate generation of test cases. We first analyze linear models, both one factor and multiple factor parameters and then address models that fit polynomial curves, and log linear curves for the data.

6.2.1 Linear Regression models

Regression is a statistical methodology that determines the relationship between two or more factors and utilizes it to predict one from the others. When we address relationships there are two types that generally come to mind: a pure functional relationship and statistical relationship. The two differ in their models and predictions. Functional relationships define precise mathematical formulae between two or more factors. The observations for functional relationship all fall on the curve. Statistical models on the other hand are not perfect. Statistical models determine best-fit curves for the data observed and hence observations do not fall directly on the curve. This is made clear from graphs given below. Figure 6.1 shows a perfect linear fit depicting functional

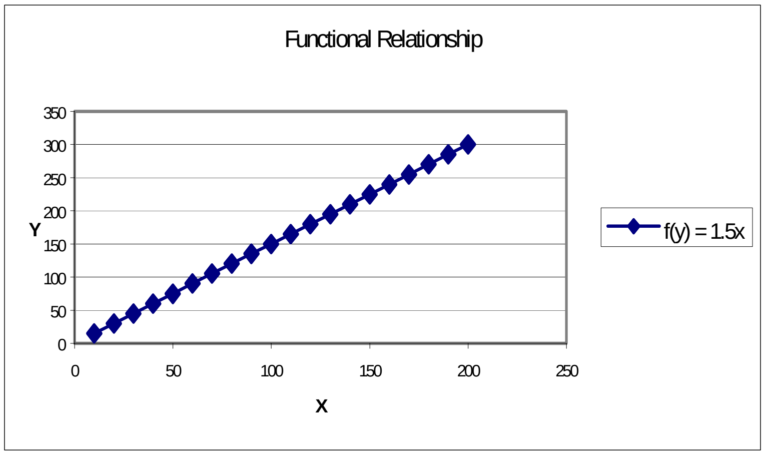

Figure 6.1: Functional Relationship *f(y) = 1.5x*

relationship that exists. Consider the statistical model fit in the graph shown in Figure 6.2. It is clear that the fit model is not linear. However regression models are statistical guesses as to the best-fit possible for the given data. Clearly, a linear fit does not accommodate the data given. Further on in this chapter we shall discuss transformation techniques to rectify this situation and explain the goodness-fit for the given regression model.

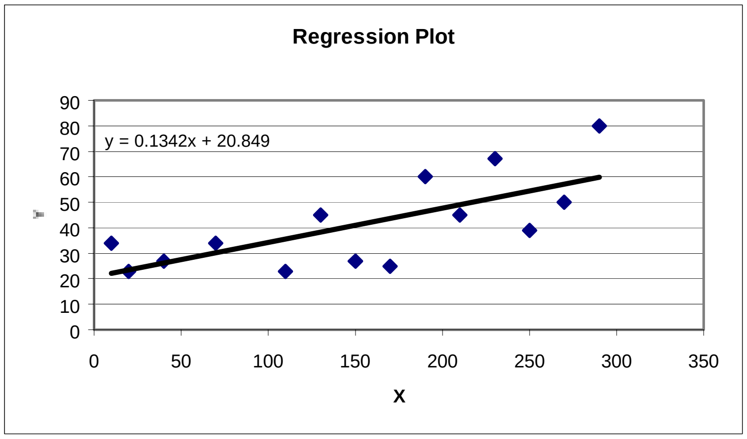

Figure 6.2: Regression Plot *f(y) = 0.1342x + 20.84*

The linear regression model is given by

$$y_i = \beta_1 x_i + \beta_2 \tag{6.1}$$

The parameters β_1 and β_2 are called the regression coefficients. β_1 is the slope of the regression line. It indicates the change in the mean of the probability distribution of *y* per

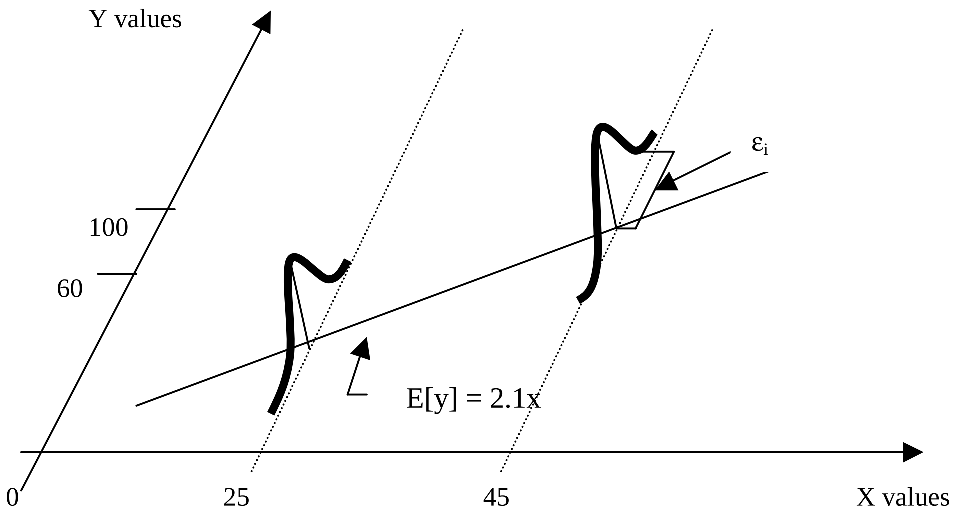

Figure 6.3: Simple Linear Regression Model

unit increase in *x*. β_2 is the y-intercept of the regression line. We know that regression models do have errors in their prediction. Errors are assumed normally distributed around the predicted point. The bell shaped curves at every value of y show this in the diagram. The mean value of this distribution (the expected value) is our predicted value. Hence, we talk about the *"mean probability distribution"* of y. The equation of a linear model with errors is

$$y_i = \beta_1 x_i + \beta_2 + \varepsilon_i \qquad \ldots(6.2)$$

The alternative model is to use the predictor variable deviation $X_i - \overline{X}$ rather than X_i. To leave the model unchanged we alter equation (6.2) as follows

$$y_i = \beta_1(x_i - x_{mean}) + \beta_1 x_{mean} + \beta_2 + \varepsilon_i \qquad \ldots(6.3)$$

This can be reduced to

$$y_i = \beta_1(x_i - x_{mean}) + \beta_0^* + \varepsilon_i \qquad \ldots(6.4)$$

where $\beta_0^* = \beta_1 x_{mean} + \beta_2$.

6.2.2 Estimation of the Regression Model

The observational data used for predicting the parameters in a regression function consists of observations on the predictor variable *X* and the corresponding observations of the response variable *Y*. In general, we group each trial as a pair (X_i , Y_i), where i denotes the trial number. There are two main methods for estimation

- Method of Least Squares.
- Method of Maximum Likelihood.

6.2.2.1 Method of Least Squares

To find "good" estimates for β_1 and β_2, we employ the method of least squares. Method of least squares is based on minimizing the error between the estimated value for the response variable and the actual value for the response variable. In essence, method of least squares requires that the sum of squared deviation be minimum. The criterion may be mathematically formulated as

Minimize $Q = \sum_{i=1}^{n}[(y_i - (\beta_1 x_i + \beta_2)]^2$ …(6.5)

We need to determine the values of β_1 and β_2 for which the fitted regression model would have minimum errors in the estimates. We determine the values $\beta_1 = b_1$ and $\beta_2 = b_2$ for which the criterion is satisfied. The values are derived by partially differentiating equation (6.5) with respect to β_1 and β_2, we get

$$\frac{\partial Q}{\partial \beta_1} = -2\sum_{i=1}^{n}(y_i - \beta_1 x_i - \beta_2)x_i \qquad (6.6)$$

and $$\frac{\partial Q}{\partial \beta_2} = -2\sum_{i=1}^{n}(y_i - \beta_1 x_i - \beta_2) \qquad (6.7)$$

Equating equations (6.6) and (6.7) to zero for minimization we get

$$\sum_{i=1}^{n}(y_i - \beta_1 x_i - \beta_2)x_i = 0 \qquad (6.8)$$

and $$\sum_{i=1}^{n}(y_i - \beta_1 x_i - \beta_2) = 0 \qquad (6.9)$$

Solving the two above equation we get

$$b_1 = \frac{\sum_{1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sum_{1}^{n}(x_i - \bar{x})^2} \qquad (6.10)$$

and $$b_2 = \frac{1}{n}\left(\sum_{1}^{n} y_i - b_1 \sum_{1}^{n} x_i\right) = \bar{y} - b_1 \bar{x} \qquad (6.11)$$

6.2.2.2 Method of Maximum Likelihood

No matter what may be the form of the distribution of the error terms ε_i, the least squares method provides unbiased estimators of β_0 and β_1 that have minimum variance among all unbiased linear estimators. The normal error regression model is as follows:

$$Y_i = \beta_0 + \beta_1 X_i + \varepsilon_i \qquad \text{….(6.12)}$$

Where:

Y_i is the observed response in the i^{th} trial,

β_0 and β_1 are parameters,

X_i is a known constant, the level of the predictor variable in the i^{th} trial,

ε_i are independent $N(0, \sigma^2)$,

i = 1……, n.

The regression model implies that the Y_i are independent normal random variables, with mean $E(Y_i) = \beta_0 + \beta_1 X_i$ and variance σ^2. The normality assumption for the error terms is justifiable in many situations because the error terms frequently represent the effects of factors omitted from the model that effect the response to some extent and vary at random without reference to the variable X. When the functional form of the probability distribution of the error terms is specified, estimators of the parameters β_0 , β_1 and σ^2 can be obtained by the *method of maximum likelihood*.

Essentially, the method of maximum likelihood chooses as estimates those values of the parameters that are most consistent with the sample data. The concepts presented for maximum likelihood estimation of a population mean carry over directly to the estimation of the parameters of normal error regression model. For this model, each Y_i observation is normally distributed with mean $\beta_0 + \beta_1 X_i$ and standard deviation σ.

In general, the density of a function Y_I for the normal error regression model is as follows utilizing the fact that $E(Y_i) = \beta_0 + \beta_1 X_i$ and $\sigma^2(Y_i) = \sigma^2$:

$$f_i = \frac{1}{\sqrt{2\pi\sigma}} \exp\left[-\frac{1}{2}\left(\frac{Y_i - \beta_0 - \beta_1 X_i}{\sigma} \right)^2 \right] \qquad (6.13)$$

The likelihood function for n observations Y_1, Y_2… ,Y_n is the product of the individual densities. Since the variance σ^2 of the error terms is usually unknown, the likelihood function is a function of three parameters, $\beta_{0,}$ $\beta_{1,}$ and σ^2.:

$$L(\beta_0, \beta_1, \sigma^2) = \prod_{i=1}^{n} \frac{1}{(2\pi\sigma^2)^{1/2}} \exp\left[-\frac{1}{2\sigma^2} (Y_i - \beta_0 - \beta_1 X_i)^2 \right] \qquad (6.14)$$

$$L(\beta_0, \beta_1, \sigma^2) = \frac{1}{(2\pi\sigma^2)^{n/2}} \exp\left[-\frac{1}{2\sigma^2} \sum_{i=1}^{n} (Y_i - \beta_0 - \beta_1 X_i)^2 \right] \qquad (6.15)$$

The values of $\beta_{0,}$ $\beta_{1,}$ and σ^2 that maximize this likelihood function are the maximum likelihood estimators and are denoted by $\hat{\beta}_0$, $\hat{\beta}_1$ and $\hat{\sigma}^2$ respectively. These estimators can be found analytically, and they are as follows:

Parameter	Maximum Likelihood Estimator
β_0	$\hat{\beta}_0 = b_0$
β_1	$\hat{\beta}_1 = b_1$
σ^2	$\hat{\sigma}^2 = \Sigma(Y_i - \overline{Y}_i)$ 1..n

Thus the maximum likelihood parameters of β_0 and β_1 are the same estimators as provided by the method of least squares. Here maximum likelihood estimator $\hat{\sigma}^2$ is biased, and ordinarily the unbiased estimator MSE is used. Note that the unbiased estimator MSE differs but slightly from the likelihood estimator $\hat{\sigma}^2$, especially if *n* is not small:

$$MSE = \left(\frac{n}{n-2}\right)\hat{\sigma}^2 \tag{6.16}$$

6.2.3 Multiple Regression Models – Surface Models

Every so often the linear regression model may not suffice to delineate the nature of the data that one is trying to model. The reason is two-fold,

- Either the response variable in the model does not relate to only one parameter but may be dependent on several factors in reality.
- There is also the plausible causality between two or more predictor variables. These are termed statistically as interactions between the factors.

When scenarios like these arise, we adopt multiple linear regression models also called surface models (the reason being that the fitted regression model forms a surface in the n-dimensional space). In general, we term all models as multiple linear regression models even if they contain quadratic factors or exponential terms. This is made clear further on in this chapter.

We may represent the multiple regression model as

$$Y_i = \beta_0 + \beta_1 X_{i1} + \beta_2 X_{i2} + \beta_3 X_{i3} + \ldots\ldots\ldots\ldots + \beta_{ip} X_{ip} + \varepsilon_i \tag{6.17}$$

where

Y_I is the predicted variable,

$X_{i1} – X_{ip}$: are the predictor variables,

β_1 - β_2 are the regression coefficients.

In order to simplify the explanation let us consider the case of bivariate regression. In this case Y_i is dependent on two variables represented as X_{i1} and X_{i2}. The regression model takes the form

$$Y_i = \beta_0 + \beta_1 X_{i1} + \beta_2 X_{i2} + \varepsilon_i \tag{6.18}$$

Here the coefficients β_1 and β_2 represent the variability of X_{i1} and X_{i2} with changes in Y_i. In other words the β1 denote the per unit change in the value Y_i for a small change in the value of X_{i1} provided X_{i2} is held constant. The vice versa is the definition for β2. ε_{ii} represents the error in the estimate for Y_i.

6.2.3.1 Estimating variability and multiple correlation coefficient

We define the multiple regression correlation coefficient R^2 as a measure of the prediction of Y obtained from the regression equation. If Y is perfectly predicted then R^2 = 1. If the multiple regression equation predicts no better than the equation $Y = \overline{Y}$, then R^2 = 0. The proportion of the variability of Y accounted for by regression on p predictor variables is given by

$$R^2_{Y.12....p} = \frac{SS_{reg}}{SS_y} \tag{6.19}$$

where $SS_{reg} = \sum (\hat{Y}_i - \overline{Y})^2$ is the amount of variability in Y accounted by regression.

The variability of Y defined as SS_y is partitioned into two main parts SS_{reg} and SS_{error}. The first is the variability by regression alone and the second is the error or residuals. SS_{reg} for multiple regression with p predictor variables with completely uncorrelated variables leads to non-overlapping error components associated with each of the predictors. Hence the total variability is the sum of individual variability for each predictor variable. However, if the variables are correlated then we need the regression coefficient to be adjusted to reflect the true variability and goodness fit estimate. For our purposes to determine adjusted R^2 we use

$$R_{adj}^2 = 1 - \frac{SS_{error} / (N-1-p)}{SS_y / (N-1)} \tag{6.20}$$

where N is the sampled population size and p is the number of predictor variables

6.2.3.2 Estimation of the regression coefficients

We may represent the regression equation in matrix terms as follows (assuming two predictor variables for illustration purposes)

$$\begin{bmatrix} Y_1 \\ Y_2 \\ Y_3 \\ . \\ Y_n \end{bmatrix} = \begin{bmatrix} 1 & X_{11} & X_{12} \\ 1 & X_{21} & X_{22} \\ 1 & X_{31} & X_{32} \\ . & . & . \\ 1 & .X_{n1} & .X_{n2} \end{bmatrix} \text{ X } \begin{bmatrix} \beta_1 \\ \beta_2 \\ \beta_3 \end{bmatrix} + \begin{bmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \varepsilon_3 \\ \varepsilon_4 \\ \varepsilon_5 \end{bmatrix} \tag{6.21}$$

This forms a set of linear equations that needs to be solved to determine the regression coefficients. The equation may be represented as

$$Y = BX + \varepsilon \tag{6.22}$$

Let X' represent the transpose of matrix X. Similarly β' and Y' represent the respective matrix transposes for β and Y. Therefore we have

$$Y' = \begin{bmatrix} Y_1 & Y_2 & Y_3 & . & Y_n \end{bmatrix} \tag{6.23}$$

$$\beta' = \begin{bmatrix} \beta_1 & \beta_2 & \beta_3 \end{bmatrix} \tag{6.24}$$

and X' as

$$X' = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ X_{21} & X_{22} & X_{23} & X_{24} & . & X_{2n} \\ X_{31} & X_{32} & X_{33} & X_{34} & . & X_{3n} \end{bmatrix} \tag{6.25}$$

We can then determine the coefficient matrix β to be given by

$$\beta = (X'X)^{-1}X'Y \tag{6.26}$$

The various statistics are then given by

$$ESS = \beta' X'Y - n\overline{Y}^2 \tag{6.27}$$

$$TSS = Y'Y - n\overline{Y}^2 \tag{6.28}$$

$$R^2 = \frac{ESS}{TSS} \tag{6.29}$$

$$F = \frac{R^2 / (k-1)}{(1-R^2)/(n-k)} \tag{6.30}$$

$$R^2_{adj} = 1 - (1-R^2)\frac{n-k}{k-1} \tag{6.31}$$

where n is the number of observations and k is the number of variables involved in the regression equation. The value of R^2 approaching 1 shows a very good fit for the data. To formally verify the goodness fit we determine the equivalent F-statistic. A high value for the F-statistic is indicative of a good fit and a low value closer to 1 is indicative of a bad regression fit for prediction.

6.3 Selecting the Best Regression Equation for Prediction

Sometimes the criterion of interest is predicted by developing a regression equation containing a subset of the potentially useful predictor variables that are available. A number of automated procedures have been developed to produce the best possible predictions with regression equations that contain relatively few predictors.

These procedures include

- Forward selection
- Backward elimination
- Stepwise regression.

With these procedures it is often possible to select a subset of the potential predictors that accounts for nearly as large a proportion of the variability in Y as does the entire pool of predictors.

6.3.1 Forward Selection

In this procedure, one variable at a time is used to build the regression equation. Initially the predictor with the highest correlation (positive or negative) is selected. If it fails to meet the criterion for inclusion, the procedure ends with no predictors in the equation, and the final equation is

$$Y_i = \overline{Y} \qquad ..(6.32)$$

If the first predictor meets the criterion, on the next step a second predictor is selected and tested to determine whether it should be entered into the equation. The predictor selected is the one that would result in the greatest increment in R^2 if added to the equation. If the second predictor does not meet the criterion for inclusion, the procedure terminates with only a single predictor in the equation. If it does meet the criterion, on the third step, a third predictor is selected and tested, and so on. At each step, a partial *F* test is performed on the selected variable, and the criterion for inclusion is stated in terms of the critical value or the significance level of the *F*. In forward selection and stepwise regression, a liberal criterion for entering variables into the equation is often employed. This will generally allow the investigation of more variables than would normally be used, so that a number of possible equations can be considered.

It should be noted that for procedures like forward selection, the usual significance levels obtained from the *F* distribution are not appropriate. This is because at each step a number of possible predictors are examined and only one- the one that produces the greatest increment in R^2 or, equivalently, the one that has the largest partial *F*- is tested. If only a single predictor variable is to be chosen from a pool of *m* possible predictors, the situation is analogous to choosing the largest member of a family of *m* contrasts and testing it for significance.

6.3.2 Backward Elimination

Backward elimination begins with all the predictors in the equation and removes them one by one until the final equation is obtained. At each step, the predictor in the equation that produces the smallest increment in R^2 is tested to determine whether it should be removed from the equation. Again the criterion for removal is generally stated in terms of the significance level of a partial *F* test. If the selected variable is removed, another predictor is selected and tested on the next step. The procedure terminates when a predictor that has been selected for testing is not removed from the equation; it and all other predictors remaining in the equation are included in the final regression equation.

6.3.3 Stepwise Regression

Stepwise regression is a combination of forward selection and backward elimination. The procedure is essentially the same as the forward selection with the exception that after each new predictor has been added to the regression equation, all the predictors already in the equation are reexamined to determine whether they should be removed. A partial *F* test is performed on the predictor already in the equation that produces the smallest increment in R^2. If the predictor no longer satisfies the criteria for inclusion, it is removed from the equation. Statistical packages allow the user to set the significance levels for entering or removing a variable. The F for entering the variables into the equation should be set at least as high as the F for removing them. Otherwise, variables may be cycled in and out of the equation.

It is not difficult to see why it is sometimes desirable to remove a predictor that had been entered early in the analysis. For example, suppose that X_5 is highly predictable from X_4 and X_9 but is more highly correlated with Y than either of them. Even though X_5 may enter the equation early because of its high correlation with *Y*, it will become superfluous after X_4 and X_9 are entered. That is even if X_5 contributes significantly to the predictability of *Y* by itself, it may not make a significant contribution over and above the predictability provided by the other two variables.

Again it is important to emphasize that when predictor variables entered into the equation are selected from a larger pool, the significance levels printed out by stepwise programs are not ‘real’ *p* values. Because many practitioners seem to be unaware of this fact, stepwise regression outputs are often misinterpreted.

Finally we again emphasize that the sole motivation for the automated procedures described in this section is to develop useful prediction equations that include subsets of the available variables.

6.4 Development of Model

We needed to develop vertical atmospheric profiles for wind, temperature and pressure. As outlined earlier on in this chapter our first stage in this process is to analyze the data. We first determine the variations of temperature, pressure and wind with increasing altitude. We plotted graphs to determine the variation. Figure 6.4-6.6 shows the individual variations of temperature, pressure and wind with height.

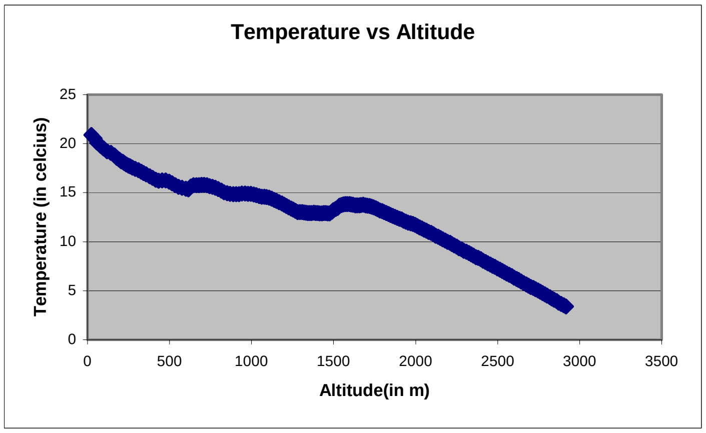

Figure 6.4 Graph showing variation in temperature with altitude

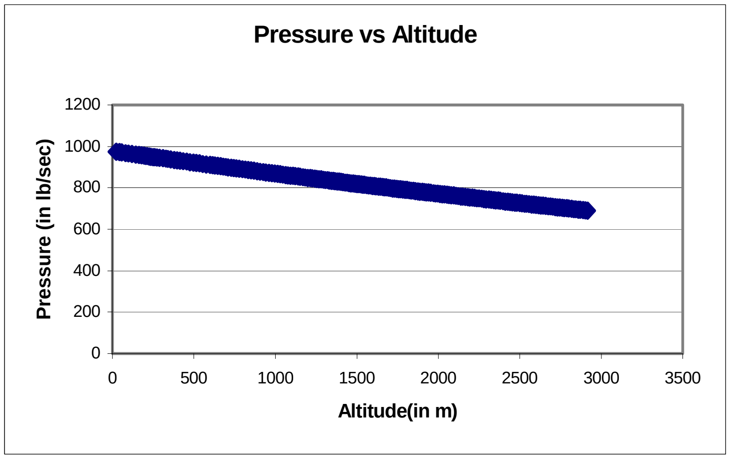

Figure 6.5 Variation of Pressure with Altitude

The graph for pressure (Figure 6.5) shows the greatest linearity with altitude. Hence we choose pressure as the independent variable and expressed all other parameters in terms of pressure. The basic philosophy is to fit regression models to predict pressure changes with altitude and then determine the corresponding values of temperature and wind based on generated values of pressure. We fit normal or Gaussian mutators to generate different but close values around the mean value of pressure in a given day at the ground level. The purpose of mutations is to generate a new set of test trajectories. We then predict the value sets as described above.

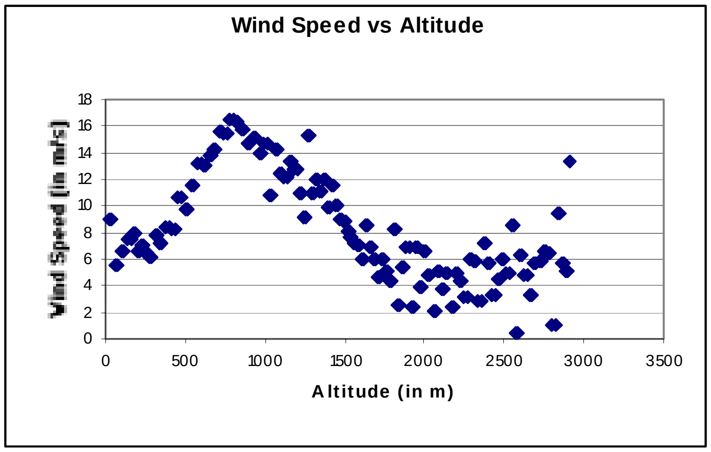

Figure 6.6 Variation of Wind Speed with Altitude

6.4.1 Pressure Model

The following fitted models presented in this section provide prediction for a single day, the chosen day is 28th May 1996. The subsections explain how we fit a model for predicting pressure, temperature and wind conditions.

We first ran a correlation between pressure and altitude. We obtained the index as $CORR_{pres,wind}$ = -0.99923. This is indicative of a possibly high probability of linearity between the two factors. Hence we choose a linear regression function to predict pressure based on altitude. The fitted regression equation may be represented as

$$pressure = b_1(alt) + c_1 \qquad ..(6.33)$$

Solving we get b_1 = -0.09795 and c_1 = 969.2212. Hence the equation is

$$pressure = -0.09795(alt) + 969.2212 \qquad ..(6.34)$$

In order to test to goodness fit we determine the correlation regression coefficient and the F-statistic, the values found were R^2 = 0.999 and *F-stat* = 124860. Both these values indicate a very good regression fit between the predicted (dependent) and predictor (independent) variables.

6.4.2 Temperature Model

To determine temperature variation with altitude, we ran a correlation with pressure and altitude. The respective correlation coefficients obtained were $CORR_{temp,press}$ = 0.95 and with altitude $CORR_{temp,\, alt}$ = -0.9574. We shall now adopt a forward regression methodology. Develop a model with only altitude as a parameter. The correlation regression coefficient R^2 = 0.9164. This is a good fit but we decided to introduce pressure in the equation too and find out if there is an improvement in prediction. Now the regression coefficient improved to R^2 = 0.9448. This is a significant improvement over the previous prediction.

We investigated whether temperature varies linearly with a transformed variable (like either log(alt) or $(alt)^2$). In fact the temperature had a better correlation with alt^2. We now did stepwise regression. We needed to determine if all the factors contributed towards the prediction. When adopting the stepwise regression methodology we determined that the combined contribution of pressure and height is better in predicting the temperature than alt^2. The equation is

$$temp = -0.21257(pressure) - 0.02547(alt) + 225.5906 \qquad ..(6.35)$$

The fitted regression plot function visualized as a surface forms a plane in the 3-D space. This is as shown in the figure below

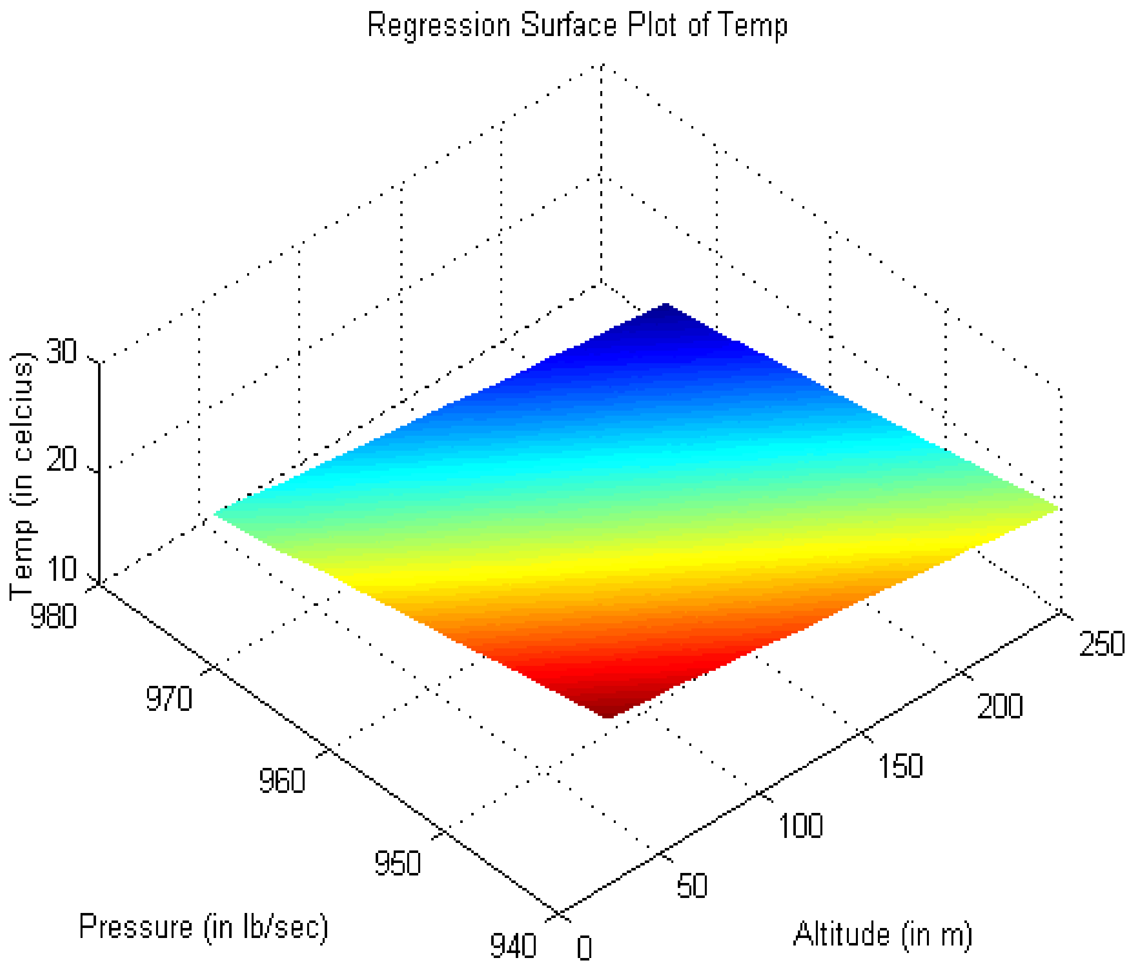

Figure 6.6: Surface Plot of Temperature with Pressure and Altitude

6.4.3 Wind Model

The challenge was to model wind values in terms of altitude, pressure and temperature. Wind's vicissitudes in values needed some form of transformation to be applied to the predictor and response values to try and conform it to as near linearity as possible. We adopted certain standard transformations like log, exponential, square root and others. There was a great degree of randomness in the values for wind. Stepwise regression was then performed. The fitted regression equation was $wind = 20383.21 + 21 * \log(temp) - 2045765 * (1/Pressure) - 3876 * \log(pressure)$.(6.36)

6.5 Generating Values

There are two approaches for developing models for predicting atmospheric conditions. One model is to develop atmospheric conditions similar to mean monthly wind and pressure. The second model is to have a new model generated for each day and predict for that particular day. Since the DOLILU II system required to have near real time situations to test, and as mean monthly atmospheric conditions do not capture the nuances in atmospheric conditions on a particular day, we decided on having a separate model for each day.

Having predicted the equations in the preceding subsections we shall now generate input conditions to form different test suites. One major advantage with the second approach is we generate closely related input trajectory parameters, which helps us in creating an oracle that unambiguously decides on the output generated.

We now fit a mutator, say a normal mutator, that randomly generates pressure values in range at the ground level. We then generate the corresponding values of for pressure, wind, and temperature based on the values generated.

7.0 Chapter 7.0: Conclusions and Further Work

In this thesis we have attempted to address three primary issues involved when assessing software,

- A framework of assessment. We used a Bayesian framework to incorporate our knowledge of the system into our assessment model. We needed to determine how we could assign priors to reflect the belief in our system. We investigated and determined that we may assign values for p, q based on mean probability of failure and the tolerance for variance.
- Enhancing the speed for test execution. Apart from vertical slicing, we attempted at alternative schemes for enhancing speed for execution. Monotonic transformations like changing numerical precision could enhance the speed. But we couldn't incorporate these changes permanently to the software. If we needed to do so, we required an oracle that decides whether the output was incorrect. With an occurrence of an error the oracle then runs original program segment to determine the correct output.
- Automated generation of test cases.

The first two address the economic feasibility for testing software, especially high assurance software. The last issue deals with automating generation of test cases for ensuring the reliability of the software when there is scarcity of previous data.

We developed a statistical framework for assessing software reliability of a high assurance system DOLILU II. We used a Bayesian Inference Framework, primarily because of the use of subjective prior knowledge in the reliability assessment model. Bayesian methods not only provide better interpretation of test results but also achieve it with fewer test executions. A comparison between Random Sampling and Bayesian methodologies clearly shows a superior framework for assessment when using Bayesian Statistics.

One of the important areas for further research would be how to translate different methodologies of quality assurance into prior beliefs for the software? For Example if the specification for the software was extensively exercised, how could this translate into a belief within the assessment model. How could bayesian statistics support analysis in an earlier stage in the software lifecycle?

It is clear that no single assessment method is capable of an accurate prediction of software quality. We could further research into marrying different methods like formal verification, testing coupled with bayesian framework to achieve a better framework which assesses high assurance software economically, and in a reasonable time frame.

We also discussed methods for enhancing the speed for each test execution. We focused primarily on program slices (semantic transformation), that are partial programs, which are capable of executing independently. The union of all the outputs produced by the individual slices forms the output of the original program. We also suggested changes for numerically intensive slices to further enhance the speed.

Research could be directed on how we may achieve this. We need to research into applicability of the transformation with their criterion of inclusion, in other words what should be an acceptable degree of latitude one may give to these transformed computations so that they would not adversely affect the output. Research could be done also in predicting the change in the output for the proposed changes.

One aspect that wasn’t addressed in this thesis is generation of a test oracle. An oracle unambiguously decides whether a given output is correct or incorrect. For DOLILU II system the generated outputs for any two input trajectories are expected to be close for near similar input conditions. An oracle could simply decide on an erroneous output by determining the difference in the two simulated outputs for closely related inputs. Research can be done further on generating an automated oracle on the aforementioned premise. One has to decide what statistical distance would qualify to define the degree of closeness for the inputs and the outputs.

There are ample opportunities were we could apply a different approach for assessing. We could also look into genetic algorithms, which may be used to generate test suites for a program. Genetic algorithms are learning based algorithms, which continually enrich themselves. We could research into the applicability of these algorithms for test case generation based on criterion like logical path coverage and others.

Bibliography

[1] T. Forester, P Morrison, Computer Ethics: Cautionary Tales and Ethical Dilemmas in Computing, (Second Edition), MIT Press, Cambridge, 1994.

[2] L. Hatton, A Roberts, "How accurate is Scientific Software", *IEEE Trans. Software Engineering* Vol. 20, Oct. 1994, pp. 785-797.

[3] Littlewood Bev, L. String00ini, "The Risks of Software", *Scientific America*, Nov. 1992, pp. 62-75.

[4] "Ariane 501- Report by the Inquiry Board", European Space Agency, available on http://www.esrin.esa.it/tidc/Press/Press96/ariane5rep.html.

[5] "Mars Climate Orbiter Mishap", Mars Climate Orbiter Failure Investigation Board, Phase I Report, Art Stephenson, Lia L. LaPiana, D. R. Mulville, Peter J. Rutledge, D. Folta, Greg Dukeman, R. Sackheim, Peter Norvig, Dec. 1999, JPL Pasadena, CA. available on ftp://ftp.hq.nasa.gov/pub/pao/reports/1999/MCO_report.pdf

[6] Cukic, Bojan "Transformational Approach to Software Reliability Assessment", Doctoral Thesis, Department of Computer Science, University of Houston, August 1997.

[7] "Software in Safety Related Systems", IEE/BCS Joint Study Report, A Special Report, B. A. Wichmann, John Wiley & Sons, Chichester, 1992, pp. 1-122.

[8] S. Bhattacharya, A. Onama, F. B. Bastani, "High Assurance Systems", Comm. of the ACM, Vol40, No.1, Jan. 1997, pp. 67.

[9] A.N. Charette, Application Strategies for Risk Analysis, Software Engineering Series, McGraw Hill, 1990.

[10] C. Ghezzi, M. Jazayeri, Programming Language Concepts, John Wiley & Sons, New York, 1982.

[11] Jonathan Bowen "*Formal Methods in Safety-Critical Standards*", *Proc. 1993 Software Engineering Standards Symposium (SESS'93)*, Brighton, UK, 30th Aug. - 3 September 1993, pp 168-177.

[12] J.C. Huang, "An Approach to Program Testing", *ACM Computing Surveys*, Vol. 8, No. 3, Sept. 1975, pp. 113-128.

[13] Bev Littlewood, Lorenzo Strigini, “Validation of Ultrahigh Dependability for Software-based Systems”, *Comm. of the ACM*, Vol. 36, No. 11, Nov. 1993, pp. 69-79.

[14] S. Gerhart, D. Craigen, T. Ralston, “Experience with Formal Methods in Critical Systems”, *IEEE Software*, Vol. 11,No. 1, Jan. 1994, pp. 21-28.

[15] C. Ghezzi, M. Jazayeri, D. Mandrioli, Fundamentals of Software Engineering, Prentice-Hall, (Third ed.) 1997.

[16] I.J. Hayes, “Specification directed module testing”, *IEEE Trans. Software Engineering*, vol. 12, No. 1, Jan. 1986, pp. 124-133.

[17] J.W. Laski, B. Korel, “A Data Flow oriented program testing strategy”, *IEEE Trans. Software Engineering*, vol. 20, No. 5, March 1985, pp. 72-87.

[18] S. Rapps and E.J. Weyuker, “Selecting software test data using data flow information”, *IEEE Trans. Software Engineering*, vol. 11, no. 4, April 1985, pp. 367-375.

[19] J.D. Musa, “Operational profiles in software reliability engineering”, *IEEE Software,* March 1993, pp. 14-32.

[20] J. Jacky, “Specifying a Safety Critical Control System in Z”, *IEEE Trans. in Software Engineering*, Vol. 21, No. 2, Feb. 1995, pp. 99-106.

[21] G. Luo, A. Das, G.V. Bochmann, “Software testing based on SDL specifications with save”, *IEEE Trans. Software Engineering*, vol.20, Jan. 1994, pp. 72-87.

[22] L.J. White, E.I. Cohen, “A domain strategy for computer program testing”, *IEEE Trans. Software Engineering*, vol. 6, no. 7, July 1991, pp. 247-257.

[23] L. Dalton, E. Collins, P. Perry, G. Pollac, C. Sicking, “A Review of Research and Methods for Producing High Consequence Software”, *Proc. 1995 IEEE Aerospace Applications Conference*, Vol. 1, Aspen, CO, 1995, pp. 197-245.

[24] T.Y. Chen, Y.T.Yu, “On the Expected Number of failures Detected by Sub-domain Testing and Random Testing”, *IEEE Trans. Software Engineering*, vol. 22, No. 2, Feb. 1996, pp. 109-119.

[25] P.G. Bishop, “The variation of Software Survival Tine for Different Operational Input Profiles, *Proc. IEEE Intl Symp. on Fault-Tolerant Computing FTCS-23*, Toulouse, France, June 1993, pp. 98-107.

[26] A. Avizienis, L. Chen, “On the Implementation N-Version Programming”, *Proc. Computer Software and Applications Conference*, 1977, pp. 149-155.

[27] F. B. Bastani, A. Pasquini, "Assessment of a Sampling Method for Measuring Safety-Critical Software Reliability", *Proc. of 5th Int'l. Symp. on Software Reliability Engineering(ISSRE '94)*, Montery, CA., Nov.1994.

[28] Harry F. Martz, Ray A. Waller, Bayesian Reliability Analysis, John Wiley and Sons Inc., New York, 1982.

[29] J.W. Duran, S.C. Ntafos, "An Evaluation of Random Testing", *IEEE Trans. Software Engineering*, vol. 10, no. 7, July 1984, pp. 438-444.

[30] R. Hamlet, R. Taylor, "Partition Testing does not inspire confidence", *IEEE Trans. Software Engineering*, vol. 16, no. 12, Dec. 1990, pp. 1402-1411.

[31] W. G. Cochran, Sampling Techniques, John Wiley and Sons Inc., New York, N.Y., 1977.

[34] C.A.R. Hoare, "An Axiomatic Basis for Computer Programming", *Comm. of the ACM*, vol. 12, No. 10, Oct. 1969, pp. 576-583.

[35] R. B. Anderson, Proving Programs Correct, John Wiley & Sons Inc., New York, N.Y., 1979.

[36] R.M. Burstall, P.J. Landin, "Programs and their proofs: An Algebraic Approach", *Machine Intelligence*, No. 4, Edinburgh University Press, 1969, pp. 17-43.

[37] R. DeMillo, R. Lipton, F. Sayad, "Hints on test data selection: help for the practicing programmer", *IEEE Computer*, vol. 11, no. 4, Apr. 1978, pp. 34-41.

[38] J.M. Voas, K.W. Miller, "Software testability: The new verification", *IEEE Software*, May 1995, pp. 17-28.

[39] Software Requirements Specification, Flight Design and Dynamics, Ascent Discipline, Ascent Subsystem, Day-of-Launch Function, DIVDT Program, Version 4.2P, STSOC-RQ-820754, Rockwell Space Operations Company, March 22, 1993.

[40] NSTS 08329, DOLILU II System Definition and Requirements Document, Volume VI, DOLILU II Quality Assurance Rules, Day-of-Launch Function, DIVDT Program, NASA, Feb 1992.

[41] B. Cukic, D. McCaugherty, D. Chakravarthy, "Reliability Prediction of a Trajectory Verification System". 1998 IEEE Workshop on Application-Specific System and Software Engineering, Richardson, TX, March 1998.

[42] Detailed Design Document: Flight Design and Dynamics, Ascent Discipline, Ascent Subsystem, Day-of-Launch Function, DIVDT Program, Version 4.4FP, SOC-SP-820949B, Rockwell Space Operations Company, May 22, 1993.

[43] Acceptance Test Procedures, Flight Design and Dynamics, Ascent Discipline, Ascent Subsystem, Day-of-Launch Function, DIVDT Program, Version 4.4FP, Rockwell Space Operations Company, December 15, 1994.

[44] N.G. Leveson, M. Heimdahl, H. Hildreth, J. Reese, “Requirements Specification for Process Control Systems”, *IEEE Trans. Software Engineering*, vol. 20, no. 9, September 1994.

[45] Bayes T., 1958 Essay Towards Solving a Problem in the Doctrine of Chances, *Biometrica*, Vol. 45, pp. 293-315.

[46] Bev Littlewood, David Wright, “Stopping Rules for the Operational Testing of Safety Critical Software”,*25th Conference on FTCS*, Pasadena, CA, June 1995, pp. 444-451.

[47] N.L. Johnston, S. Kotz, *Distributions in Statistics: Discrete Distributions Vol. 1 & 2*, John Wiley, New York, 1969.

[48] M. Weiser, “Program Slicing”, *IEEE Transactions on Software Engineering*, vol. 10, no. 7, July 1984, pp. 352-357.

[49] K. Ottenstein and L. Ottenstein, “The Program Dependence Graph in Software Development Environments”, *In Proceeding of the ACM SIGSOFT/SIGPLAN Software Engineering Symposium on Practical Software Development Environments*, May 1984, pp. 177-184.

[50] Anouar Jamoussi, Cukic B., Hilford V., Bastani F.B., “Accelerated Execution of Test Cases for Software Reliability Assessment”

[51] Butler R.W., Finelli G.B., “The Infeasibility of Quantifying the Reliability of Life-Critical Real-Time Software”, NASA Langley Research Center, Hampton, VA.

[52] Keiller P.A., Miller D.R., “On the use and the performance of software reliability growth models”, *Reliability Engineering and System Safety*, 1991, pp. 95-117.

Appendix A

We adopted a tabular form of representation for ease of explanation. Consider table A.2, which provides the proof for the evaluation of rule S1 (pitch and yaw I-load with Sail envelopes). The format is as follows, each procedure call forms a separate row in the table. The keyword procedure and a descriptive name delineating the purpose of the procedure precede them. Variables declared locally within the procedure are defined in the row immediately following the procedure. Variables indicated as global are either defined as common in FORTRAN (or passed by reference in C) to the function.

When DIVDT is invoked the first procedure executed is to set up input files for DIVDT's proper functioning. If any of the input files are missing, DIVDT terminates the evaluation process and an appropriate message is displayed or logged in a file. This procedure is shown in row 1 of table A.2.

The next procedure called checks for the format of the input trajectory and limit files. In order to maintain consistency in representing data for trajectories and input files, NASA adopted a standard method of representation. If any of the files deviate from their expected format or if there are missing values or an incorrect data type (e.g. a float value is expected in the file but textual string is found) DIVDT terminates the evaluation process and logs the error in a file. The file contains information on where the error occurred, the filename and the line at which the error occurred, the type of error that caused the termination. This is procedure in row 2 in table A.2.

The third row in table A.2 indicates reading in the input limits for each individual rule. Although the limits for all the rules are read in, when vertically slicing only variables that affect the evaluation of a specific rule are required. These variables are indicated in the succeeding row. Allocations are done on the heap so that the values may be passed on to evaluation procedure.

Rows 5 through 15 are executed in a loop. This is shown alongside the table with an arrow starting at row 15 and ending in row 5. This loop terminates with the end of the trajectory files. Trajectory files are large and cannot be stored in memory. Hence DIVDT loops through each point in the trajectory file. Every point in the trajectory file is stored in specified format referred to as a record. Every record in the file is preceded by a textual line, indicating which record type to use based on which stage (first stage, pre-orbital insertion or orbiting stage) in the flight trajectory is the evaluation taking place. The record names used are

- STDRCD:- this indicates the standard record format for the first stage conditions in the trajectory.
- SRBSTD:- indicates the standard record format for pre-orbital insertion stage in the trajectory.
- MECSTD:- indicates the standard record format for the orbiting stage.
- SRBSUP:- this is a supplementary record format required for certain rules during pre-orbital insertion stage.
- MECSUP:- supplementary record format required for certain rule evaluations during the orbiting stage.

Each record is an array and follows a common naming convention. Record names are of one of the following types STDRCD_XXX, SRBSTD_XXX, MECSTD_XXX, SRBSUP_XXX and MECSUP_XXX. Here _XXX = DOLIT to indicate SVDS simulated trajectory for day-of-launch conditions, or _XXX = DADS to indicate DADS simulated trajectory or _XXX = REFT indicating reference trajectory.

Row 5 in table A.2, refers to the procedure call that reads in SVDS trajectory record into the appropriate array, followed by DADS trajectory record (row 7 in table A.2) and finally Reference trajectory record (row 9 in table A.2).

Row 12 calls procedure read DADS I-Loads (guidance commands) which reads the corresponding DADS I-Loads from the dads_iloads file into ILRECD array.

Row 14 in table A.2 calls procedure evaluate ruleS1. This procedure evaluates the rule for the given point in the trajectory. It stores percentage exceedance and the reference values. Once the evaluation is completed for all points in the trajectory, the data is stored in files (ASCII for textual files like detailed summary and binary for plot files).

Row 16 in table A.2 calls DIVDTPLT to generate the plot files.

We can repeat the same procedure for the other rules.

Table A.2 Rule S1

1	procedure SETUP_INPUT_FILES		
2	Procedure CHECK_UNIT15 (format for all files)		
3	Procedure Read_Input_File_Limits		
4	ARV[51] PSI_LOW_LIM[51] PSI_HI_LIM[51] THET_HI_LIM[51] THET_LOW_LIM[51]		
5	Procedure Process_DOLITRAJ (read in record)		
6	STDRCD_DOLIT[250]	MECSTD_DOLIT[200] MECSUP_DOLIT[50]	SRBSTD_DOLIT[200] SRBSUP_DOLIT[50]
7	Procedure Process_DADSTRAJ (read in record)		
8	STDRCD_DADS[250]	MECSTD_DADS[200] MECSUP_DADS[50]	SRBSTD_DADS[200] SRBSUP_DADS[50]
9	Procedure Process_REFTRAJ (read in record)		
11	STDRCD_REFT[250]	MECSTD_REFT[200] MECSUP_REFT[50]	SRBSTD_REFT[200] SRBSUP_REFT[50]
12	Procedure Read DADS_Iloads		
13	ILRECD[200]		
14	Procedure Evaluate RuleS1		

15	PSI_ACTUAL , PEREXCEEDANCE [90] (global), REFVALUES[90](global) THET_ACTUAL, RESULT[90](global)		
16	Procedure DIVDTPLT		

Table A.3 Rule S2

1	Procedure SETUP_INPUT_FILES		
2	Procedure CHECK_UNIT15 (format for all files)		
3	Procedure Read_Input_File_Limits		
4	QBR_LIM		
5	Procedure Process_DOLITRAJ (read in record)		
6	STDRCD_DOLIT[250]	MECSTD_DOLIT[200] MECSUP_DOLIT[50]	SRBSTD_DOLIT[200] SRBSUP_DOLIT[50]
7	Procedure Process_DADSTRAJ (read in record)		
8	STDRCD_DADS[250]	MECSTD_DADS[200] MECSUP_DADS[50]	SRBSTD_DADS[200] SRBSUP_DADS[50]
9	Procedure Process_REFTRAJ (read in record)		
11	STDRCD_REFT[250]	MECSTD_REFT[200] MECSUP_REFT[50]	SRBSTD_REFT[200] SRBSUP_REFT[50]
12	Procedure Evaluate_RuleS2		
13	QBAR_ACTUAL , PERCENTEXCEEDANCE[90], REFVALUES[90] RESULT[90]		
14	Procedure DIVDTPLT		

Table A.4: Rule S3

1	procedure SETUP_INPUT_FILES		
2	Procedure CHECK_UNIT15 (format for all files)		
3	Procedure Read_Input_File_Limits		
4	ALD_LIM_LOW ALD_LIM_HI		
5	Procedure Process_DOLITRAJ (read in record)		
6	STDRCD_DOLIT[250]	MECSTD_DOLIT[200] MECSUP_DOLIT[50]	SRBSTD_DOLIT[200] SRBSUP_DOLIT[50]
7	Procedure Process_DADSTRAJ (read in record)		
8	STDRCD_DADS[250]	MECSTD_DADS[200] MECSUP_DADS[50]	SRBSTD_DADS[200] SRBSUP_DADS[50]
9	Procedure Process_REFTRAJ (read in record)		
10	STDRCD_REFT[250]	MECSTD_REFT[200] MECSUP_REFT[50]	SRBSTD_REFT[200] SRBSUP_REFT[50]
11	Procedure Evaluate RuleS3		
12	ALD_ACTUAL (iterative process evaluating the rule for each record in the i/p traj.) RESULT		
13	Procedure DIVDTPLT		

Table A.5: Rule S4

1	procedure SETUP_INPUT_FILES		
2	Procedure CHECK_UNIT15 (format for all files)		
3	Read Limits RuleS4		
4	BED_LIM_LOW BED_LIM_HI		
5	Procedure Process_DOLITRAJ (read in record)		
6	STDRCD_DOLIT[250]	MECSTD_DOLIT[200] MECSUP_DOLIT[50]	SRBSTD_DOLIT[200] SRBSUP_DOLIT[50]
7	Procedure Process_DADSTRAJ (read in record)		
8	STDRCD_DADS[250]	MECSTD_DADS[200] MECSUP_DADS[50]	SRBSTD_DADS[200] SRBSUP_DADS[50]
9	Procedure Process_REFTRAJ (read in record)		
10	STDRCD_REFT[250]	MECSTD_REFT[200] MECSUP_REFT[50]	SRBSTD_REFT[200] SRBSUP_REFT[50]
11	Evaluate RuleS4		
12	BED_ACTUAL (iterative process evaluating the rule for each record in the input trajectory) RESULT		
13	Procedure DIVDTPLT		

Table A.6: Rule S5

1	procedure SETUP_INPUT_FILES		
2	Procedure CHECK_UNIT15 (format for all files)		
3	Procedure Read Limits RuleS5		
4	PD_LIM_LOW PD_LIM_HI QD_LIM_LOW QD_LIM_HI RD_LIM_LOW RD_LIM_HI		
5	Procedure Process_DOLITRAJ (read in record)		
6	STDRCD_DOLIT[250]	MECSTD_DOLIT[200] MECSUP_DOLIT[50]	SRBSTD_DOLIT[200] SRBSUP_DOLIT[50]
7	Procedure Process_DADSTRAJ (read in record)		
8	STDRCD_DADS[250]	MECSTD_DADS[200] MECSUP_DADS[50]	SRBSTD_DADS[200] SRBSUP_DADS[50]
9	Procedure Process_REFTRAJ (read in record)		
10	STDRCD_REFT[250]	MECSTD_REFT[200] MECSUP_REFT[50]	SRBSTD_REFT[200] SRBSUP_REFT[50]
11	Procedure Evaluate RuleS5		
12	ROLL_RATE (iterative process evaluating the rule for each record in the i/p traj.) YAW_RATE PITCH_RATE RESULT		
13	Procedure DIVDTPLT		

Table A.7: Rule S6

1	procedure SETUP_INPUT_FILES		
2	Procedure CHECK_UNIT15 (format for all files)		
3	Procedure Read Limits RuleS6		
4	DELTA_WIND_MARGIN		
5	Procedure Process_DOLITRAJ (read in record)		
6	STDRCD_DOLIT[250]	MECSTD_DOLIT[200] MECSUP_DOLIT[50]	SRBSTD_DOLIT[200] SRBSUP_DOLIT[50]
7	Procedure Process_DADSTRAJ (read in record)		
8	STDRCD_DADS[250]	MECSTD_DADS[200] MECSUP_DADS[50]	SRBSTD_DADS[200] SRBSUP_DADS[50]
9	Procedure Process_REFTRAJ (read in record)		
10	STDRCD_REFT[250]	MECSTD_REFT[200] MECSUP_REFT[50]	SRBSTD_REFT[200] SRBSUP_REFT[50]
11	Procedure Evaluate RuleS6		
12	Wind_Margin RESULT		
13	Procedure DIVDTPLT		

Table A.8: Rule S7

1	procedure SETUP_INPUT_FILES		
2	Procedure CHECK_UNIT15 (format for all files)		
3	Procedure Read Limits RuleS7		
4	ARV[50] ELVHM_INB_SYST_LIM_LOW[50] ELVHM_OUTB_SYST_LIM_LOW[50]		
5	Procedure Process_DOLITRAJ (read in record)		
6	STDRCD_DOLIT[250]	MECSTD_DOLIT[200] MECSUP_DOLIT[50]	SRBSTD_DOLIT[200] SRBSUP_DOLIT[50]
7	Procedure Process_DADSTRAJ (read in record)		
8	STDRCD_DADS[250]	MECSTD_DADS[200] MECSUP_DADS[50]	SRBSTD_DADS[200] SRBSUP_DADS[50]
9	Procedure Process_REFTRAJ (read in record)		
10	STDRCD_REFT[250]	MECSTD_REFT[200] MECSUP_REFT[50]	SRBSTD_REFT[200] SRBSUP_REFT[50]
11	Procedure Evaluate RuleS7		
12	Array_Size Elvm_Left_Inboard Elvm_Right_Inboard Evlm_Left_Outboard Evlm_Right_Outboard RESULT		
13	Procedure DIVDTPLT		

Table A.9: Rule S8

1	procedure SETUP_INPUT_FILES
2	Procedure CHECK_UNIT15 (format for all files)
3	Procedure Read Limits RuleS8 (VENTDATA)

4	NMCH (number of Mach Values) MACH (MACH number) NBET (number of Batches) PTS (number of Points) CALD (Center Altitude) CBAR (Center Pressure) ALD (Altitude Coords.) QBAR (Pressure) DALD (Delta Altitude (change)) DBAR (Change in Pressure)		
5	Procedure Process_DOLITRAJ (read in record)		
6	STDRCD_DOLIT[250]	MECSTD_DOLIT[200] MECSUP_DOLIT[50]	SRBSTD_DOLIT[200] SRBSUP_DOLIT[50]
7	Procedure Process_DADSTRAJ (read in record)		
8	STDRCD_DADS[250]	MECSTD_DADS[200] MECSUP_DADS[50]	SRBSTD_DADS[200] SRBSUP_DADS[50]
9	Procedure Process_REFTRAJ (read in record)		
11	STDRCD_REFT[250]	MECSTD_REFT[200] MECSUP_REFT[50]	SRBSTD_REFT[200] SRBSUP_REFT[50]
14	Procedure Evaluate RuleS1		
15	ALPHA QBAR RESULT		
16	Procedure DIVDTPLT		

Table A.10: Rule S9

1	procedure SETUP_INPUT_FILES		
2	Procedure CHECK_UNIT15 (format for all files)		
3	Procedure Read Limits RuleS9		
4	ARRAY_SIZE MACH_REF[50] (Mach Reference) MIN_ALPHA_5[50] MIN_BETA_5[50] MAX_DENSITY_1[50]		
5	Procedure Process_DOLITRAJ (read in record)		
6	STDRCD_DOLIT[250]	MECSTD_DOLIT[200] MECSUP_DOLIT[50]	SRBSTD_DOLIT[200] SRBSUP_DOLIT[50]
7	Procedure Process_DADSTRAJ (read in record)		
8	STDRCD_DADS[250]	MECSTD_DADS[200] MECSUP_DADS[50]	SRBSTD_DADS[200] SRBSUP_DADS[50]
9	Procedure Process_REFTRAJ (read in record)		
10	STDRCD_REFT[250]	MECSTD_REFT[200] MECSUP_REFT[50]	SRBSTD_REFT[200] SRBSUP_REFT[50]
11	Procedure Evaluate RuleS9		
12	RESULT		
13	Procedure DIVDTPLT		

Table A.11: Rule S13

1	procedure SETUP_INPUT_FILES		
2	Procedure CHECK_UNIT15 (format for all files)		
3	Procedure Read Limits RuleS13		
4	ARRAY_SIZE ALT_LIM[50] THR_LIM[50] KMIN_ALT[50] DEL_ALT_TWO[50]		
5	Procedure Process_DOLITRAJ (read in record)		
6	STDRCD_DOLIT[250]	MECSTD_DOLIT[200] MECSUP_DOLIT[50]	SRBSTD_DOLIT[200] SRBSUP_DOLIT[50]
7	Procedure Process_DADSTRAJ (read in record)		
8	STDRCD_DADS[250]	MECSTD_DADS[200] MECSUP_DADS[50]	SRBSTD_DADS[200] SRBSUP_DADS[50]
9	Procedure Process_REFTRAJ (read in record)		
10	STDRCD_REFT[250]	MECSTD_REFT[200] MECSUP_REFT[50]	SRBSTD_REFT[200] SRBSUP_REFT[50]
11	Procedure Evaluate RuleS13		
12	QPOLY_3_SIGMA RESULT		
13	Procedure DIVDTPLT		

Table A.12: Rule S14

1	procedure SETUP_INPUT_FILES		
2	Procedure CHECK_UNIT15 (format for all files)		
3	Procedure Read Limits RuleS14		
4	VP_SIZE (array_sizes) ZVRT_REF[50] MIN_XVRT_ALLOWED[50] PR_SIZE[50] TIME_REF[50] MIN_PRANGE[50] PL_SIZE[50] PLONG_REF[50] MIN_PLAT_ALLOWED[50] MAX_PLAT_ALLOWED[50]		
5	Procedure Process_DOLITRAJ (read in record)		
6	STDRCD_DOLIT[250]	MECSTD_DOLIT[200] MECSUP_DOLIT[50]	SRBSTD_DOLIT[200] SRBSUP_DOLIT[50]
7	Procedure Process_DADSTRAJ (read in record)		
8	STDRCD_DADS[250]	MECSTD_DADS[200] MECSUP_DADS[50]	SRBSTD_DADS[200] SRBSUP_DADS[50]
9	Procedure Process_REFTRAJ (read in record)		
10	STDRCD_REFT[250]	MECSTD_REFT[200] MECSUP_REFT[50]	SRBSTD_REFT[200] SRBSUP_REFT[50]
11	Procedure Evaluate RuleS14		

12	XVRT ZVRT YVRT HPLN_PRANGE HPLN_PLAT RESULT
13	Procedure DIVDTPLT

Table A.13: Rule S15

1	procedure SETUP_INPUT_FILES		
2	Procedure CHECK_UNIT15 (format for all files)		
3	Procedure Read Limits RuleS15		
4	WTZMAR MARGIN_LIM		
5	Procedure Process_DOLITRAJ (read in record)		
6	STDRCD_DOLIT[250]	MECSTD_DOLIT[200] MECSUP_DOLIT[50]	SRBSTD_DOLIT[200] SRBSUP_DOLIT[50]
7	Procedure Process_DADSTRAJ (read in record)		
8	STDRCD_DADS[250]	MECSTD_DADS[200] MECSUP_DADS[50]	SRBSTD_DADS[200] SRBSUP_DADS[50]
9	Procedure Process_REFTRAJ (read in record)		
10	STDRCD_REFT[250]	MECSTD_REFT[200] MECSUP_REFT[50]	SRBSTD_REFT[200] SRBSUP_REFT[50]
11	Procedure Evaluate RuleS15		
12	WEIGHT_MARGIN RESULT		
13	Procedure DIVDTPLT		

Table A.14: Rule S16

1	procedure SETUP_INPUT_FILES		
2	Procedure CHECK_UNIT15 (format for all files)		
3	Procedure Read Limits RuleS16		
4	ADI_TIME_LIM PD_ADI_LIM_LOW PD_ADI_LIM_HI QD_ADI_LIM_LOW QD_ADI_LIM_HI RD_ADI_LIM_HI RD_ADI_LIM_LOW		
5	Procedure Process_DOLITRAJ (read in record)		
6	STDRCD_DOLIT[250]	MECSTD_DOLIT[200] MECSUP_DOLIT[50]	SRBSTD_DOLIT[200] SRBSUP_DOLIT[50]
7	Procedure Process_DADSTRAJ (read in record)		
8	STDRCD_DADS[250]	MECSTD_DADS[200] MECSUP_DADS[50]	SRBSTD_DADS[200] SRBSUP_DADS[50]
9	Procedure Process_REFTRAJ (read in record)		
10	STDRCD_REFT[250]	MECSTD_REFT[200] MECSUP_REFT[50]	SRBSTD_REFT[200] SRBSUP_REFT[50]
11	Procedure Evaluate RuleS16		

12	YAW_RATES PITCH_RATES ROLL_RATES RESULT
13	Procedure DIVDTPLT

Table A.15: Rule S19

1	procedure SETUP_INPUT_FILES		
2	Procedure CHECK_UNIT15 (format for all files)		
3	Procedure Read Limits RuleS19		
4	ARRAY_SIZE WIND_ALT[50] WNDNT_LIM_LOW[50] WNDNT_LIM_HI[50] WNDET_LIM_HI[50] WNDET_LIM_LOW[50]		
5	Procedure Process_DOLITRAJ (read in record)		
6	STDRCD_DOLIT[250]	MECSTD_DOLIT[200] MECSUP_DOLIT[50]	SRBSTD_DOLIT[200] SRBSUP_DOLIT[50]
7	Procedure Process_DADSTRAJ (read in record)		
8	STDRCD_DADS[250]	MECSTD_DADS[200] MECSUP_DADS[50]	SRBSTD_DADS[200] SRBSUP_DADS[50]
9	Procedure Process_REFTRAJ (read in record)		
10	STDRCD_REFT[250]	MECSTD_REFT[200] MECSUP_REFT[50]	SRBSTD_REFT[200] SRBSUP_REFT[50]
11	Procedure Read DADS_Iloads		
12	ILRECD[200]		
13	Procedure Evaluate RuleS19		
14	WND_EAST_CPMT WND_NORTH_CPMT RESULT		
15	Procedure DIVDTPLT		

Table A.16: Rule S20

1	procedure SETUP_INPUT_FILES		
2	Procedure CHECK_UNIT15 (format for all files)		
3	Procedure Read Limits RuleS20		
4	TREF_LIM_HI TREF_LIM_LOW		
5	Procedure Process_DOLITRAJ (read in record)		
6	STDRCD_DOLIT[250]	MECSTD_DOLIT[200] MECSUP_DOLIT[50]	SRBSTD_DOLIT[200] SRBSUP_DOLIT[50]
7	Procedure Process_DADSTRAJ (read in record)		
8	STDRCD_DADS[250]	MECSTD_DADS[200] MECSUP_DADS[50]	SRBSTD_DADS[200] SRBSUP_DADS[50]
9	Procedure Process_REFTRAJ (read in record)		
10	STDRCD_REFT[250]	MECSTD_REFT[200] MECSUP_REFT[50]	SRBSTD_REFT[200] SRBSUP_REFT[50]
11	Procedure Evaluate S20		
12	TIME_REF RESULT		

13	Procedure DIVDTPLT		

Table A.17: Rule S21

1	Procedure SETUP_INPUT_FILES		
2	Procedure CHECK_UNIT15 (format for all files)		
3	Read Limits RuleS21		
4	QPOLY_SYS_LOW QPOLY_SYS_HI QPOLY_SYS_LIM_LOW QPOLY_SYS_LIM_HI QPOLY1_SYS_LOW QPOLY1_SYS_HI QPOLY1_SYS_LIM_LOW QPOLY1_SYS_LIM_HI QPOLY2_SYS_LOW QPOLY2_SYS_HI QPOLY2_SYS_LIM_LOW QPOLY2_SYS_LIM_HI QPOLY3_SYS_LOW QPOLY3_SYS_HI	QPOLY3_SYS_LIM_LOW QPOLY3_SYS_LIM_HI QPOLY4_SYS_LOW QPOLY4_SYS_HI QPOLY4_SYS_LIM_LOW QPOLY4_SYS_LIM_HI THROT1_SYS_LOW THROT1_SYS_HI THROT1_SYS_LIM_LOW THROT1_SYS_LIM_HI THROT2_SYS_LOW THROT2_SYS_HI THROT2_SYS_LIM_LOW THROT2_SYS_LIM_HI	THROT3_SYS_LOW THROT3_SYS_HI THROT3_SYS_LIM_LOW THROT3_SYS_LIM_HI THROT4_SYS_LOW THROT4_SYS_HI THROT4_SYS_LIM_LOW THROT4_SYS_LIM_H
5	Procedure Process_DOLITRAJ (read in record)		
6	STDRCD_DOLIT[250]	MECSTD_DOLIT[200] MECSUP_DOLIT[50]	SRBSTD_DOLIT[200] SRBSUP_DOLIT[50]
7	Procedure Process_DADSTRAJ (read in record)		
8	STDRCD_DADS[250]	MECSTD_DADS[200] MECSUP_DADS[50]	SRBSTD_DADS[200] SRBSUP_DADS[50]
9	Procedure Process_REFTRAJ (read in record)		
10	STDRCD_REFT[250]	MECSTD_REFT[200] MECSUP_REFT[50]	SRBSTD_REFT[200] SRBSUP_REFT[50]
11	Procedure Evaluate Rule S21		
12	QPLOY[4] THROT[4] RESULT		
13	Procedure DIVDTPLT		

Table A.18: Rule S22

1	procedure SETUP_INPUT_FILES		
2	Procedure CHECK_UNIT15 (format for all files)		
3	Procedure Read Limits RuleS22		
4	TDEL_LIM		
5	Procedure Process_DOLITRAJ (read in record)		
6	STDRCD_DOLIT[250]	MECSTD_DOLIT[200] MECSUP_DOLIT[50]	SRBSTD_DOLIT[200] SRBSUP_DOLIT[50]
7	Procedure Process_DADSTRAJ (read in record)		
8	STDRCD_DADS[250]	MECSTD_DADS[200] MECSUP_DADS[50]	SRBSTD_DADS[200] SRBSUP_DADS[50]
9	Procedure Process_REFTRAJ (read in record)		
10	STDRCD_REFT[250]	MECSTD_REFT[200] MECSUP_REFT[50]	SRBSTD_REFT[200] SRBSUP_REFT[50]

11	Procedure Evaluate RuleS22		
12	RESULT		
13	Procedure DIVDTPLT		

Table A.19: Rule S23

1	procedure SETUP_INPUT_FILES		
2	Procedure CHECK_UNIT15 (format for all files)		
3	Procedure Read Limits RuleS23		
4	ARRAR_SIZE MACH_REF[50] ACCEL_X_LOW[50] ACCEL_X_HI[50]	ACCEL_Y_LOW[50] ACCEL_Y_HI[50] ACCEL_Z_LOW[50] ACCEL_Z_HI[50]	
5	Procedure Process_DOLITRAJ (read in record)		
6	STDRCD_DOLIT[250]	MECSTD_DOLIT[200] MECSUP_DOLIT[50]	SRBSTD_DOLIT[200] SRBSUP_DOLIT[50]
7	Procedure Process_DADSTRAJ (read in record)		
8	STDRCD_DADS[250]	MECSTD_DADS[200] MECSUP_DADS[50]	SRBSTD_DADS[200] SRBSUP_DADS[50]
9	Procedure Process_REFTRAJ (read in record)		
10	STDRCD_REFT[250]	MECSTD_REFT[200] MECSUP_REFT[50]	SRBSTD_REFT[200] SRBSUP_REFT[50]
11	Procedure Evaluate Rule23		
12	X_CPMT Y_CPMT Z_CPMT RESULT		
13	Procedure DIVDTPLT		

Table A.21: Rule S24

1	procedure SETUP_INPUT_FILES		
2	Procedure CHECK_UNIT15 (format for all files)		
3	Procedure Read Limits RuleS24		
4	ARRAY_SIZE TIME[50] ALT_PLUME_LIM[50] LOW_CONST[50]		
5	Procedure Process_DOLITRAJ (read in record)		
6	STDRCD_DOLIT[250]	MECSTD_DOLIT[200] MECSUP_DOLIT[50]	SRBSTD_DOLIT[200] SRBSUP_DOLIT[50]
7	Procedure Process_DADSTRAJ (read in record)		
8	STDRCD_DADS[250]	MECSTD_DADS[200] MECSUP_DADS[50]	SRBSTD_DADS[200] SRBSUP_DADS[50]
9	Procedure Process_REFTRAJ (read in record)		
10	STDRCD_REFT[250]	MECSTD_REFT[200] MECSUP_REFT[50]	SRBSTD_REFT[200] SRBSUP_REFT[50]
11	Procedure Evaluate RuleS24		
12	RESULT		
13	Procedure DIVDTPLT		

Table A.22: Rule E1

1	procedure SETUP_INPUT_FILES		
2	Procedure CHECK_UNIT15 (format for all files)		
3	Procedure Read Limits RuleE1		
4	GAMMA_STG_LIM_MAX GAMMA_STG_LIM_MIN		
5	Procedure Process_DOLITRAJ (read in record)		
6	STDRCD_DOLIT[250]	MECSTD_DOLIT[200] MECSUP_DOLIT[50]	SRBSTD_DOLIT[200] SRBSUP_DOLIT[50]
7	Procedure Process_DADSTRAJ (read in record)		
8	STDRCD_DADS[250]	MECSTD_DADS[200] MECSUP_DADS[50]	SRBSTD_DADS[200] SRBSUP_DADS[50]
9	Procedure Process_REFTRAJ (read in record)		
10	STDRCD_REFT[250]	MECSTD_REFT[200] MECSUP_REFT[50]	SRBSTD_REFT[200] SRBSUP_REFT[50]
11	Procedure Evaluate RuleE1		
12	GAMMA_STG RESULT		
13	Procedure DIVDTPLT		

Table A.23: Rule E2

1	procedure SETUP_INPUT_FILES		
2	Procedure CHECK_UNIT15 (format for all files)		
3	Procedure Read Limits RuleE2		
4	ALT_STG_LIM_MAX ALT_STG_LIM_MAX		
5	Procedure Process_DOLITRAJ (read in record)		
6	STDRCD_DOLIT[250]	MECSTD_DOLIT[200] MECSUP_DOLIT[50]	SRBSTD_DOLIT[200] SRBSUP_DOLIT[50]
7	Procedure Process_DADSTRAJ (read in record)		
8	STDRCD_DADS[250]	MECSTD_DADS[200] MECSUP_DADS[50]	SRBSTD_DADS[200] SRBSUP_DADS[50]
9	Procedure Process_REFTRAJ (read in record)		
10	STDRCD_REFT[250]	MECSTD_REFT[200] MECSUP_REFT[50]	SRBSTD_REFT[200] SRBSUP_REFT[50]
11	Procedure Evaluate RuleE2		
12	ALT_STG RESULT		
13	Procedure DIVDTPLT		

Table A.24: Rule E3

1	procedure SETUP_INPUT_FILES
2	Procedure CHECK_UNIT15 (format for all files)
3	Procedure Read Limits RuleE3
4	MAX_DEL_AZI MIN_DEL_AZI
5	Procedure Process_DOLITRAJ (read in record)

6	STDRCD_DOLIT[250]	MECSTD_DOLIT[200] MECSUP_DOLIT[50]	SRBSTD_DOLIT[200] SRBSUP_DOLIT[50]
7	Procedure Process_DADSTRAJ (read in record)		
8	STDRCD_DADS[250]	MECSTD_DADS[200] MECSUP_DADS[50]	SRBSTD_DADS[200] SRBSUP_DADS[50]
9	Procedure Process_REFTRAJ (read in record)		
10	STDRCD_REFT[250]	MECSTD_REFT[200] MECSUP_REFT[50]	SRBSTD_REFT[200] SRBSUP_REFT[50]
11	Procedure Evaluate RuleE3		
12	AZIMUTH_STG RESULT		
13	Procedure DIVDTPLT		

Table A.25: Rule E4

1	procedure SETUP_INPUT_FILES		
2	Procedure CHECK_UNIT15 (format for all files)		
3	Procedure Read Limits RuleE4		
4	ARRAY_SIZE TIME[50] EBFB1_LIM_LOW[50] EBFB1_LIM_HI[50] EBFB2_LIM_LOW[50] EBFB2_LIM_HI[50] EBFB3_LIM_LOW[50] EBFB3_LIM_HI[50]		
5	Procedure Process_DOLITRAJ (read in record)		
6	STDRCD_DOLIT[250]	MECSTD_DOLIT[200] MECSUP_DOLIT[50]	SRBSTD_DOLIT[200] SRBSUP_DOLIT[50]
7	Procedure Process_DADSTRAJ (read in record)		
8	STDRCD_DADS[250]	MECSTD_DADS[200] MECSUP_DADS[50]	SRBSTD_DADS[200] SRBSUP_DADS[50]
9	Procedure Process_REFTRAJ (read in record)		
10	STDRCD_REFT[250]	MECSTD_REFT[200] MECSUP_REFT[50]	SRBSTD_REFT[200] SRBSUP_REFT[50]
11	Procedure Evaluate RuleE4		
12	ROLL_ERROR YAW_ERROR PITCH_ERROR RESULT		
13	Procedure DIVDTPLT		

Table A.26: Rule E5

1	procedure SETUP_INPUT_FILES
2	Procedure CHECK_UNIT15 (format for all files)
3	Procedure Read Limits RuleE5

4	ARRAY_SIZE TIME[50] WFBFB1_LIM_LOW[50] WFBFB1_LIM_HI[50] WFBFB2_LIM_LOW[50] WFBFB2_LIM_HI[50] WFBFB3_LIM_LOW[50] WFBFB3_LIM_LOW[50]		
5	Procedure Process_DOLITRAJ (read in record)		
6	STDRCD_DOLIT[250]	MECSTD_DOLIT[200] MECSUP_DOLIT[50]	SRBSTD_DOLIT[200] SRBSUP_DOLIT[50]
7	Procedure Process_DADSTRAJ (read in record)		
8	STDRCD_DADS[250]	MECSTD_DADS[200] MECSUP_DADS[50]	SRBSTD_DADS[200] SRBSUP_DADS[50]
9	Procedure Process_REFTRAJ (read in record)		
10	STDRCD_REFT[250]	MECSTD_REFT[200] MECSUP_REFT[50]	SRBSTD_REFT[200] SRBSUP_REFT[50]
11	Procedure Evaluate RuleE5		
12	ROLL_RATES YAW_RATES PITCH_RATES RESULT		
13	Procedure DIVDTPLT		

Table A.27: Rule E6

1	procedure SETUP_INPUT_FILES		
2	Procedure CHECK_UNIT15 (format for all files)		
3	Procedure Read Limits RuleE6		
4	ARRAY_SIZE TIME[50] DPD_LIM_LOW[50] DPD_LIM_HI[50] DQD_LIM_HI[50] DQD_LIM_LOW[50] DRD_LIM_LOW[50] DRD_LIM_HIH[50]		
5	Procedure Process_DOLITRAJ (read in record)		
6	STDRCD_DOLIT[250]	MECSTD_DOLIT[200] MECSUP_DOLIT[50]	SRBSTD_DOLIT[200] SRBSUP_DOLIT[50]
7	Procedure Process_DADSTRAJ (read in record)		
8	STDRCD_DADS[250]	MECSTD_DADS[200] MECSUP_DADS[50]	SRBSTD_DADS[200] SRBSUP_DADS[50]
9	Procedure Process_REFTRAJ (read in record)		
10	STDRCD_REFT[250]	MECSTD_REFT[200] MECSUP_REFT[50]	SRBSTD_REFT[200] SRBSUP_REFT[50]
11	Procedure Evaluate RuleE6		
12	ACCEL1 ACCEL2 ACCEL3 RESULT		
13	Procedure DIVDTPLT		

Table A.27: Rule E7

1	Procedure SETUP_INPUT_FILES		
2	Procedure CHECK_UNIT15 (format for all files)		
3	Procedure Read Limits RuleE7		
4	ARRAY_SIZE TIME[50] PD_LIM_LOW[50] PD_LIM_HI[50] PD_LIM_LOW[50] RD_LIM_LOW[50] RD_LIM_LOW[50]		
5	Procedure Process_DOLITRAJ (read in record)		
6	STDRCD_DOLIT[250]	MECSTD_DOLIT[200] MECSUP_DOLIT[50]	SRBSTD_DOLIT[200] SRBSUP_DOLIT[50]
7	Procedure Process_DADSTRAJ (read in record)		
8	STDRCD_DADS[250]	MECSTD_DADS[200] MECSUP_DADS[50]	SRBSTD_DADS[200] SRBSUP_DADS[50]
9	Procedure Process_REFTRAJ (read in record)		
10	STDRCD_REFT[250]	MECSTD_REFT[200] MECSUP_REFT[50]	SRBSTD_REFT[200] SRBSUP_REFT[50]
11	Procedure Evaluate RuleE7		
12	YAW_ACT_BODY_RATE PITCH_ACT_BODY_RATE ROLL_ACT_BODY_RATE RESULT		
13	Procedure DIVDTPLT		

Table A.27: Rule E8

1	procedure SETUP_INPUT_FILES		
2	Procedure CHECK_UNIT15 (format for all files)		
3	Procedure Read Limits RuleE8		
4	ARRAY_SIZE TIME[50] VREL[50] SSME1_PITCH_LIM_LOW[50] SSME1_PITCH_LIM_HI[50] SSME2_PITCH_LIM_LOW[50] SSME2_PITCH_LIM_HI[50] SSME3_PITCH_LIM_LOW[50] SSME3_PITCH_LIM_HI[50] SSME4_PITCH_LIM_LOW[50] SSME4_PITCH_LIM_HI[50]	SSME5_PITCH_LIM_LOW[50] SSME5_PITCH_LIM_HI[50] SSME1_YAW_LIM_HI[50] SSME1_PITCH_LIM_LOW[50] SSME2_PITCH_LIM_HI[50] SSME2_PITCH_LIM_LOW[50] SSME3_PITCH_LIM_HI[50] SSME3_PITCH_LIM_LOW[50] SSME4_PITCH_LIM_HI[50] SSME4_PITCH_LIM_LOW[50]	
5	Procedure Process_DOLITRAJ (read in record)		
6	STDRCD_DOLIT[250]	MECSTD_DOLIT[200] MECSUP_DOLIT[50]	SRBSTD_DOLIT[200] SRBSUP_DOLIT[50]
7	Procedure Process_DADSTRAJ (read in record)		
8	STDRCD_DADS[250]	MECSTD_DADS[200] MECSUP_DADS[50]	SRBSTD_DADS[200] SRBSUP_DADS[50]
9	Procedure Process_REFTRAJ (read in record)		
10	STDRCD_REFT[250]	MECSTD_REFT[200] MECSUP_REFT[50]	SRBSTD_REFT[200] SRBSUP_REFT[50]
11	Procedure Evaluate RuleE8		

12	PITCH_RATES[3] YAW_RATES[3] RESULT		
13	Procedure DIVDTPLT		

Table A.28: Rule E9

1	procedure SETUP_INPUT_FILES		
2	Procedure CHECK_UNIT15 (format for all files)		
3	Procedure Read Limits RuleE9		
4	ARRAY_SIZE VREL SRBRC_LEFT_LIM_LOW SRBRC_LEFT_LIM_HI SRBRC_RIGHT_LIM_LOW	SRBRC_LEFT_LIM_HI SRBTC_LEFT_LIM_LOW SRBRC_LEFT_LIM_HI SRBRC_RIGHT_LIM_LOW SRBRC_LEFT_LIM_HI	
5	Procedure Process_DOLITRAJ (read in record)		
6	STDRCD_DOLIT[250]	MECSTD_DOLIT[200] MECSUP_DOLIT[50]	SRBSTD_DOLIT[200] SRBSUP_DOLIT[50]
7	Procedure Process_DADSTRAJ (read in record)		
8	STDRCD_DADS[250]	MECSTD_DADS[200] MECSUP_DADS[50]	SRBSTD_DADS[200] SRBSUP_DADS[50]
9	Procedure Process_REFTRAJ (read in record)		
10	STDRCD_REFT[250]	MECSTD_REFT[200] MECSUP_REFT[50]	SRBSTD_REFT[200] SRBSUP_REFT[50]
11	Procedure Evaluate RuleE9		
12	RESULT		
13	Procedure DIVDTPLT		

Table A.29: Rule E10

1	procedure SETUP_INPUT_FILES		
2	Procedure CHECK_UNIT15 (format for all files)		
3	Procedure Read Limits RuleE10		
4	MISSION_INCLINATION ARRAY_SIZE ARV[50] THET_LOW[50]	THET_HI[50] PSI_LOW_DELTA[50] PSI_HI_DELTA[50]	
5	Procedure Process_DOLITRAJ (read in record)		
6	STDRCD_DOLIT[250]	MECSTD_DOLIT[200] MECSUP_DOLIT[50]	SRBSTD_DOLIT[200] SRBSUP_DOLIT[50]
7	Procedure Process_DADSTRAJ (read in record)		
8	STDRCD_DADS[250]	MECSTD_DADS[200] MECSUP_DADS[50]	SRBSTD_DADS[200] SRBSUP_DADS[50]
9	Procedure Process_REFTRAJ (read in record)		
10	STDRCD_REFT[250]	MECSTD_REFT[200] MECSUP_REFT[50]	SRBSTD_REFT[200] SRBSUP_REFT[50]
11	Procedure Read DADS_Iloads		
12	ILRECD[200]		
13	Procedure Evaluate RuleE10		

14	YAW PITCH ROLL RESULT		
15	Procedure DIVDTPLT		

Table A.30: Rule E11

1	procedure SETUP_INPUT_FILES		
2	Procedure CHECK_UNIT15 (format for all files)		
3	Read Limits RuleE11		
4	ARRAY_SIZE ELVHM_INB_EXP_LIM_HI ELVHM_INB_EXP_LIM_LOW ELVHM_OUTB_EXP_LIM_HI ELVHM_OUTB_EXP_LIM_LOW		
5	Procedure Process_DOLITRAJ (read in record)		
6	STDRCD_DOLIT[250]	MECSTD_DOLIT[200] MECSUP_DOLIT[50]	SRBSTD_DOLIT[200] SRBSUP_DOLIT[50]
7	Procedure Process_DADSTRAJ (read in record)		
8	STDRCD_DADS[250]	MECSTD_DADS[200] MECSUP_DADS[50]	SRBSTD_DADS[200] SRBSUP_DADS[50]
9	Procedure Process_REFTRAJ (read in record)		
10	STDRCD_REFT[250]	MECSTD_REFT[200] MECSUP_REFT[50]	SRBSTD_REFT[200] SRBSUP_REFT[50]
11	Procedure Evaluate RuleE11		
12	ELV_LEFT_INBOUND ELV_RIGHT_INBOUND ELV_RIGHT_OUTBOUND ELV_LEFT_OUTBOUND RESULT		
13	Procedure DIVDTPLT		

Table A.31: Rule E14

1	procedure SETUP_INPUT_FILES		
2	Procedure CHECK_UNIT15 (format for all files)		
3	Procedure Read Limits RuleE14		
4	VREL_LIM_HI VREL_LIM_LOW		
5	Procedure Process_DOLITRAJ (read in record)		
6	STDRCD_DOLIT[250]	MECSTD_DOLIT[200] MECSUP_DOLIT[50]	SRBSTD_DOLIT[200] SRBSUP_DOLIT[50]
7	Procedure Process_DADSTRAJ (read in record)		
8	STDRCD_DADS[250]	MECSTD_DADS[200] MECSUP_DADS[50]	SRBSTD_DADS[200] SRBSUP_DADS[50]
9	Procedure Process_REFTRAJ (read in record)		
10	STDRCD_REFT[250]	MECSTD_REFT[200] MECSUP_REFT[50]	SRBSTD_REFT[200] SRBSUP_REFT[50]
11	Procedure Evaluate RuleE14		
12	RESULT		

13	Procedure DIVDTPLT		

Table A.32: Rule E15

1	procedure SETUP_INPUT_FILES		
2	Procedure CHECK_UNIT15 (format for all files)		
3	Procedure Read Limits RuleE11		
4	HDOT_STG_LIM_HI HDOT_STG_LIM_LOW		
5	Procedure Process_DOLITRAJ (read in record)		
6	STDRCD_DOLIT[250]	MECSTD_DOLIT[200] MECSUP_DOLIT[50]	SRBSTD_DOLIT[200] SRBSUP_DOLIT[50]
7	Procedure Process_DADSTRAJ (read in record)		
8	STDRCD_DADS[250]	MECSTD_DADS[200] MECSUP_DADS[50]	SRBSTD_DADS[200] SRBSUP_DADS[50]
9	Procedure Process_REFTRAJ (read in record)		
10	STDRCD_REFT[250]	MECSTD_REFT[200] MECSUP_REFT[50]	SRBSTD_REFT[200] SRBSUP_REFT[50]
11	Procedure Evaluate RuleE15		
12	ALT_RATE RESULT		
13	Procedure DIVDTPLT		

Appendix B

B.1 Solve Integration by Parts

Integration by parts may be done according to the following formula

$$I = \int u dv = u \int dv - \int \left(\int dv \right) du . \tag{B.1}$$

This implies

$$I = uv - \int v du . \tag{B.2}$$

We may represent the equation in another form

$$I = \int uv dx = u \int v dx - \int \left(\int v dx \right) \left(\frac{du}{dx} \right) dx . \tag{B.3}$$

Consider the generalized complete beta function given by

$$B(a,b) = \int_0^1 \theta^{a-1} (1-\theta)^{b-1} d\theta . \tag{B.4}$$

We have *a*>0 and *b*>0, assuming only integer values for *a*, *b* we shall now repeatedly integrate equation (B.4) using (B.3). Let *I* denote the final integration result, therefore

$$I = B(a,b) = \int_0^1 \theta^{a-1} (1-\theta)^{b-1} d\theta . \tag{B.5}$$

According to equation (B.3) we have

$$I = \theta^{a-1} \int (1-\theta)^{b-1} d\theta - \int \left(\int (1-\theta)^{b-1} d\theta \right) \left(\frac{d\theta^{a-1}}{d\theta} \right) d\theta . \tag{B.6}$$

Therefore, applying the limits [0,1], we get

$$I = \frac{-1}{b} [\theta^{a-1} (1-\theta)^b]_0^1 - \int_0^1 \frac{-(a-1)}{b} \theta^{a-2} (1-\theta)^b d\theta . \tag{B.7}$$

This evaluates to

$$I = \frac{-1}{b}(0-0) + \frac{(a-1)}{b}\int_0^1 \theta^{a-2}(1-\theta)^b d\theta \,. \tag{B.8}$$

Therefore

$$I = \frac{(a-1)}{b}\int_0^1 \theta^{a-2}(1-\theta)^b d\theta \,. \tag{B.9}$$

Let I_1 denote

$$I_1 = \int_0^1 \theta^{a-2}(1-\theta)^b d\theta \;. \tag{B.10}$$

I now become

$$I = \frac{(a-1)}{b} I_1 \tag{B.11}$$

Using equation (B.3) again to evaluate I_1 we get

$$I_1 = \theta^{a-2}\int (1-\theta)^b d\theta - \int\left(\int (1-\theta)^b d\theta\right)\left(\frac{d\theta^{a-2}}{d\theta}\right) d\theta \,. \tag{B.12}$$

Therefore applying limits [0,1] we get I_1 to be

$$I_1 = \frac{-1}{(b+1)}[\theta^{a-2}(1-\theta)^{b+1}]_0^1 - \int_0^1 \frac{-(a-2)}{b+1}\theta^{a-3}(1-\theta)^{b+1} d\theta \,. \tag{B.13}$$

This implies

$$I_1 = \frac{-1}{(b+1)}(0-0) + \frac{(a-2)}{(b+1)}\int_0^1 \theta^{a-3}(1-\theta)^{b+1} d\theta \tag{B.14}$$

$$I_1 = \frac{(a-2)}{(b+1)}\int_0^1 \theta^{a-3}(1-\theta)^{b+1} d\theta \tag{B.15}$$

Let I_2 denote the integration in equation (B.15). Therefore I_1 is

$$I_1 = \frac{(a-2)}{(b+1)} I_2 \tag{B.16}$$

This is clearly seen to be recursive (observe equations (B.16) and (B.11)). In general

$$I_i = \frac{(a-i-1)}{(b+i)} I_{i+1} \tag{B.17}$$

where I_{i+1} denotes

$$I_{i+1} = \int_0^1 \theta^{a-1-(i+1)}(1-\theta)^{(b-1)+(i+1)} d\theta \tag{B.18}$$

This continues till the power of θ^{a-1} becomes 0. Therefore

$$I_{a-2} = \frac{(a-a+2-1)}{(b+a-2)} I_{a-1} \tag{B.19}$$

and $$I_{a-1} = \int_0^1 \theta^{a-1-(a-1)}(1-\theta)^{(b-1)+(a-1)} d\theta \tag{B.20}$$

Solving for I_{a-1}, we get

$$I_{a-1} = \int_0^1 (1-\theta)^{(b+a-2)} d\theta \tag{B.21}$$

$$I_{a-1} = \frac{-1}{(b+a-1)} \left[(1-\theta)^{b+a-2}\right]_0^1 \tag{B.22}$$

$$I_{a-1} = \frac{-1}{(b+a-1)}(0-1) = \frac{1}{(b+a-1)}. \tag{B.23}$$

Substituting for I_{a-1} in equation (B.19) we get

$$I_{a-2} = \frac{1}{(b+a-2)} \cdot \frac{1}{(b+a-1)}. \tag{B.24}$$

Therefore I would be equal to

$$I = \frac{(a-1)}{b} \cdot \frac{(a-2)}{(b+1)} \cdot \frac{(a-3)}{(b+2)} \cdots\cdots \frac{1}{(b+a-1)}. \tag{B.25}$$

If we take a=p and b=q+U then we

$$B(p,q+U)=\frac{(p-1)}{(q+U)}\cdot\frac{(p-2)}{(q+U+1)}\cdots\cdots\frac{1}{(q+U+p-1)}\,. \tag{B.26}$$

This implies

$$B(p,q+U)=\frac{(p-1)!}{(q+U)(q+U+1)(q+U+2)\cdots\cdots(q+U+p-1)} \tag{B.27}$$

Table B.1: priors for corresponding belief in θ

Value of θ	*Value p*	*Value q*
10^{-2}	***5***	990
10^{-3}	***8***	9850
10^{-4}	***10***	99800
10^{-5}	***100***	4510488
10^{-6}	***120***	1997988
10^{-7}	***150***	990371123

B.2 Priors based on μ and σ² for beta distribution

We know that the mean and variance for beta distribution is given by

$$\mu \quad = \quad \frac{p}{(p+q)} \tag{B.28}$$

and

$$\sigma^2 \quad = \quad \frac{pq}{(p+q)^2(p+q+1)} \tag{B.29}$$

We believe that the system has exhibited a mean $\mu = 10^{-5}$. It is required that the variance be very less to instill better confidence in our belief. Let us assume that the variance be atleast 10^{-10}. We shall now derive p and q in terms for the mean and variance.

Using equation (B.28), we get

$$(p+q)\mu = p\,. \tag{B.30}$$

This implies

$$(1-\mu)p = \mu q \, . \tag{B.31}$$

Therefore

$$p = \frac{\mu q}{(1-\mu)} \, . \tag{B.32}$$

Now we have variance given by equation (B.29). Using (B.29), we get

$$(p+q)^2(p+q+1) = \frac{pq}{\sigma^2} \, . \tag{B.33}$$

This implies

$$(p^2+2pq+q^2)(p+q+1) = \frac{pq}{\sigma^2} \, . \tag{B.34}$$

Therefore

$$p^3+2p^2q+pq^2+p^2q+2pq^2+q^3+p^2+2pq+q^2 = \frac{pq}{\sigma^2} \, . \tag{B.35}$$

Rearranging and combining terms we have

$$p^3+q^3+p^2+q^2+3p^2q+3pq^2+2pq = \frac{pq}{\sigma^2} \, . \tag{B.36}$$

Substituting for *p* in equation (B.36) from (B.32) we get

$$\frac{\mu^3}{(1-\mu)^3}q^3+q^3+\frac{\mu^2}{(1-\mu)^2}q^2+q^2+3\frac{\mu^2}{(1-\mu)^2}q^3+3\frac{\mu}{(1-\mu)}q^3+2\frac{\mu}{(1-\mu)}q^2 = \frac{\mu}{(1-\mu)\sigma^2}q^2 \tag{B.37}$$

Since q>0, the above equation reduces to (dividing throughout by q^2) we get,

$$\frac{\mu^3}{(1-\mu)^3}q+q+\frac{\mu^2}{(1-\mu)^2}+1+3\frac{\mu^2}{(1-\mu)^2}q+3\frac{\mu}{(1-\mu)}q+2\frac{\mu}{(1-\mu)} = \frac{\mu}{(1-\mu)\sigma^2} \, . \tag{B.38}$$

Combining different parts of the equation, we get

$$\left(\frac{\mu^3}{(1-\mu)^3}+1+3\frac{\mu^2}{(1-\mu)^2}+3\frac{\mu}{(1-\mu)} \right)q = \frac{\mu}{(1-\mu)\sigma^2}-\left(1+\frac{\mu}{(1-\mu)}2+\frac{\mu^2}{(1-\mu)^2}\right). \tag{B.39}$$

This simplifies to

$$\left(1+\frac{\mu}{(1-\mu)}\right)^3 q = \frac{\mu}{(1-\mu)}\frac{1}{\sigma^2} - \left(1+\frac{\mu}{(1-\mu)}\right)^2 . \tag{B.40}$$

Simplifying further

$$\left(\frac{1}{(1-\mu)}\right)^3 q = \frac{\mu}{(1-\mu)}\frac{1}{\sigma^2} - \left(\frac{1}{(1-\mu)}\right)^2 . \tag{B.41}$$

Therefore

$$q = \left(\frac{\frac{\mu}{(1-\mu)}\frac{1}{\sigma^2}}{\left(\frac{1}{(1-\mu}\right)^3}\right) - \left(\left(\frac{1}{(1-\mu)}\right)^2 \Big/ \left(\frac{1}{(1-\mu)}\right)^3\right). \tag{B.42}$$

Simplifying, we get

$$q = \frac{\mu\,(1-\mu)^2}{\sigma^2} - (1-\mu)\,. \tag{B.43}$$

Taking $\sigma^2 = 10^{-10}$, and $\mu = 10^{-5}$, we get

$$q = 10^{-5} \text{ x } 10^{10} \text{ x } (1\text{-}0.00001)^2 \text{ - } (1\text{-}0.00001) \tag{B.44}$$

Approximating to the nearest integer

$$q = 99997$$

Therefore p will be

$$p = 10^{-5} \text{ x } 99997 / 0.99999 = 1$$

If we take $\sigma^2 = 10^{-15}$, we get q = 9999800000 and p = 99998.

Table B.2 Variation in tests with decreasing variance, confidence = 0.99, $\theta = 10^{-4}$

*Var*Variance	*P*	*q*	*tests*
$2x10^{-9}$	5	49995	13691
$1.25x10^{-9}$	8	79992	24277
$4x10^{-10}$	25	249975	67597
$1.6x10^{-10}$	60	599940	126773
$1.4x10^{-10}$	70	699930	140632

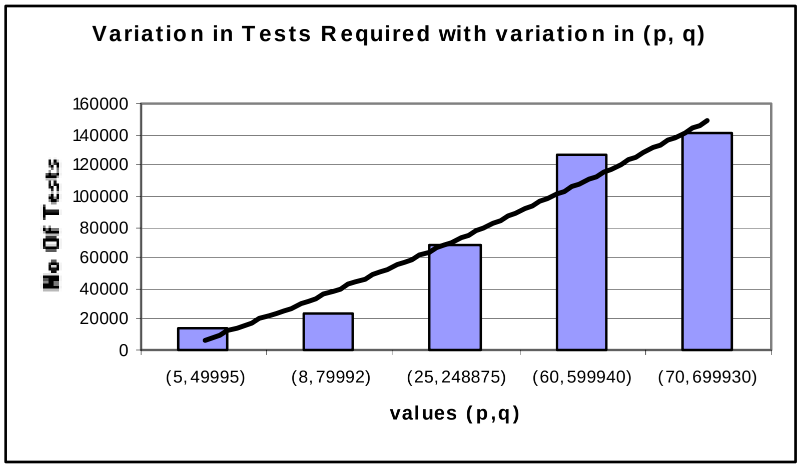

Figure B.4 Showing Variation of no. of test for $\theta = 10^{-4}$, C=0.99 with variation in p and q.