Rocket Launch Control System

Fuzzy Logic System

Abstract

This paper explains how a MATLAB system from a Fuzzy Logic perspective is capable or predicting rocket launch success, this paper rigorously tests the system to show its capability and prove end results; whilst systems are not built from a Fuzzy Logic environment this system show the potentiality of a new type of control system that could allow new adaptabilities for rocket success.

Introduction

Rockets have been in use since 1942, no matter where and when there has always been a mandatory requirement for a control system, a deterministic if you will; to cater for simulating as many possibilities to amount to perfect conditions to allow a rocket to successfully launch. There are failed launches; where not all variables we accounted for in ever-developing control systems. For example, Vanguard TV3 (1957), Apollo 6 (1968), Challenger (1986), even with knowledge base systems in developing progression there were still calamities that appeared from unmeasured variables. There are closed loop countermeasure systems being developed to allow systems to react accordingly to unknown variables that don’t yet exist [[1]](#_Bibliography).

My system aims to envelope the current necessary variables available to predict whether a rocket can launch under a pre-defined set of rules from a fuzzy logic perspective. I will be documenting the variables chosen for the Fuzzy Inference System (FIS) and why they are relevant and how the logic of the system calculates whether a rocket could successfully launch under these pretences.

Literature Review

When thinking about creating a system from a logical understanding such as Fuzzy; you’re accepting the interpretation of vagueness but with consistent degrees and bounded control. What makes Fuzzy Logic such a popular choice for control systems is the bounded control of degrees for unknown variables [[1]](#_Bibliography_1) that can be defined through a rule base even if those variables only make use in specific scenarios; making vagueness one of the best forms of control since control systems are simply limited to the number of calculable variables. If Fuzzy Logic was pure crisp set making probability a simple Boolean true or false; control systems would be limited to linear logic making variables that require degrees of truth (unknown/scalable variables) impossible to monitor and cater for.

All the data culminated based on the processing that the data evaluates on and based on a rule set that always abides by a degree of accuracy, even when the final output is defuzzified to control a final value, that output represents data that calculates on a gradient which can give more accurate definitions of whether a control system can evaluate the environment it was created for, which in my scenario would be using the degree of membership gradient to control unknown variables to give an evaluation of a rocket launch environment.

Rocket Launch Control systems usually benefit from more probability based control systems as Boolean values evaluate in pure facts rather than in vagueness which does accommodate closed loop system control [[1]](#_Bibliography_2) better than vagueness could. The amount of control inputs that define “what launches a rocket?” is more clearly identified from a Fuzzy perspective and can evaluate small details that cannot be identified by Boolean logic.

My solution to this problem is to remove the use of all closed loop systems and apply a Fuzzy Inference System that tracks vagueness to anticipate unknown variables that could occur and catch any errors that could happen and resolve that issue before it would become a problem. The degree of truth in Fuzzy logic allows me to verify all variables on a gradient to apply Membership Functions to perfect a launch. This method is a lot less linear even with a concise rule base to determine output values, since there is so much data being ruled as inputs; the evaluation of even one output would be determined in mass detail and provide all the relevant control aspects required to not only launch but control on-board systems after the rocket has left the launch pad.

There are not any public control systems available that can be used for launching a rocket; and any that are used now are purely used to control loop feedback systems that then maintain the launch control. My control system compensates for that attribute by cutting out the middle machine potentiality and controlling all aspects and variables of a launch in to one system, this is where my system varies to what is already been created currently. The attempts up to this point don’t factor the probability of the rocket propulsion mixture success, this is a vital control that maintains and controls the efficiency of the of the fuel at high temperatures so not only does the control system cover variables that proceed to a successful launch; it also maintains control throughout the launch ensuring the highest rate of flight success past the Exosphere.

The rocket propulsion formula that is used in a launch is the base evaluator to all variables proceeding the launch; if the propulsion is poor and the design of the engine is not sufficient in fuel release equalling to propulsion then other variables could fail creating void variables [[2]](#_Bibliography_3). Below defines a hybrid motor fuel system using Liquid Oxygen (Lox) and Hydroxyl – terminated polybutadiene (HTPB) as its base source. This is one example of how a launch control system could currently be designed around a propulsion aspect that would assure stable variable dimensions within the configuration and boundedness of the system; providing an accurate control over all launches. My system takes advantage of that procedure by calculating all formulas that can be used (at the time of the written report) as a propulsion factor deriving the highest solution of success that binding that formula to all future calculations made by the system as it distinguishes additional variables based on fuel propulsion and a vagueness rule set.

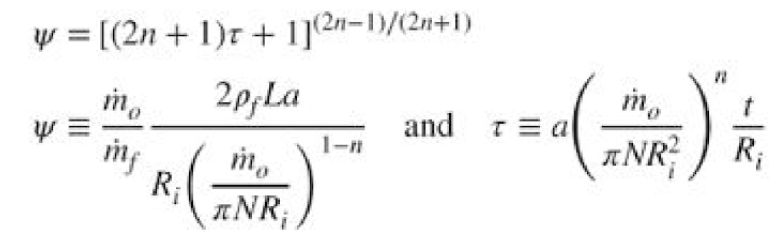


Figure 1 [[2]](#_Bibliography_4)

Displayed by figure two is a basic example of how boosters operate during stages of launch, this type of control (the type being used right now) is a closed loop system feeding static information to provoke thruster and propulsion flow changes (George, S.), whereas my system would dynamically choose a thruster configuration by using input data the most effective propulsion mixture and environmental effects; with variables scaling to each individual day, this way deterministic predictions can be identified and implemented to decide on the best thrust flow and the engine to thruster configuration.

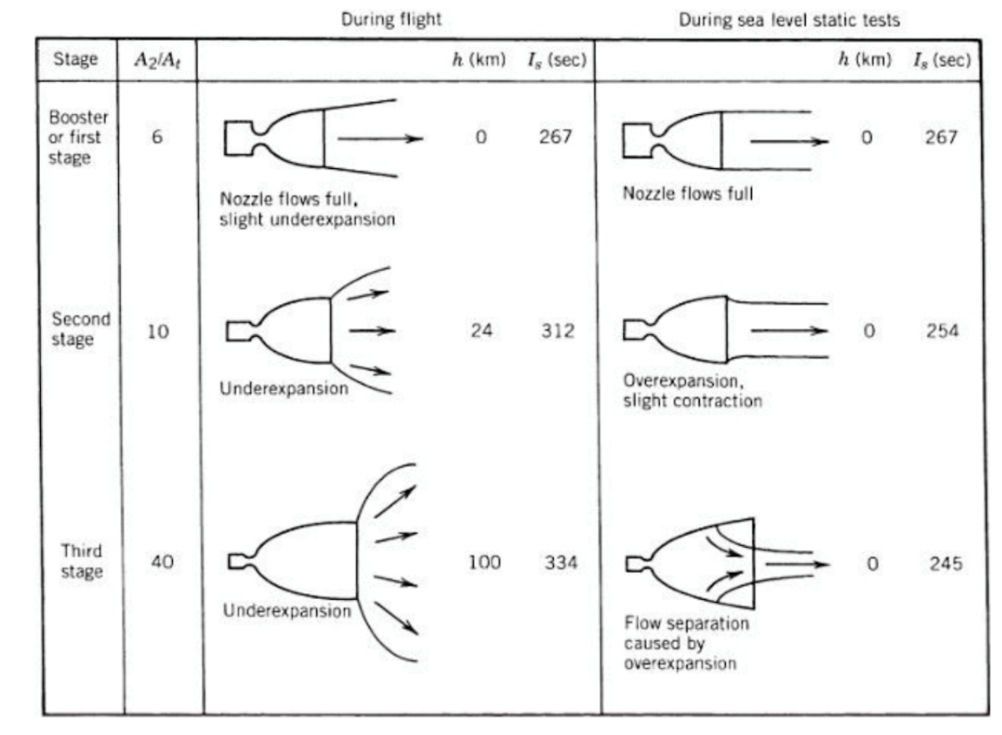


Figure 2 [[2]](#_Bibliography_4)

My solution to an Ever-Developing Problem

The first step to solving any problem is to identify in detail the problem that needs solving initially. In this scenario, the problem that I have identified is the adaptation of a control system when data is run on closed loop systems rather than a functional open network controlled by a Launch Control System. In simpler terms in rockets most solutions come from using closed loop systems that analyses static values and repeat the same protocol the same way every time; until altered manually. This method of design causes unknown variables that were not accounted for to roam around an on-board system on a rocket, sometimes these variables become harmless in the process and other times they can cause catastrophic results. This is indeed an ever-developing problem as the more technology that is implemented to provide more successful launches the more each system interacts with less co-ordination and rely on static inputs the more errors that cannot be accounted for and the system cannot adapt to this issue to resolve it.

The solution that I have developed from a Fuzzy Logic perspective is to use degrees of truth to develop a system that measures all the static variables that a closed loop system would, however my system is not closed loop which means the data gathered from those systems can be used as a statistic for another variable and more accurate predictions further along in the control system. Therefore, my system can detect unknown variables that could occur at any time and have a rule base that can accommodate such variables arising in a system. The system takes in to account the very basic structure from propulsion, to engine dynamics, thrust flow, mass, gravity, rotation, so forth. The more variables that can be retrieved and accommodated in to my system the more successful launches that can occur and the distinguishable errors can be recognised.

Unintentionally my system also alleviates the necessity of manually altering a closed loop system, you can alter the master value in the control system and the rule base will allow the system to adapt to new data and new inputs.

Rockets usually have engine simulations within the testing for successful launches, if my system adapts to that information it can also make predictions on the engine configuration that would be most successful based on the current environment; such as Weather, Rotational speeds, Fuel Loss based on gravitational strength. For example, where ‘F’ is the force of attraction between the two bodies, ‘G’ is the universal gravitational constant, ‘m sub 1’ is the mass of the first object, ‘m sub 2’ is the mass of the second object and ‘r’ is the distance between the centres of each object. This formula allows for calculations on success rates for a launch as one of the vital input variables which in turn calculates the current track for fuel loss.

Closed loop systems have developed to anticipate errors [[1]](#_Bibliography_5) and unknown variables and have designed static variables to be able to deal with foreseen implications, unbeknownst to every closed loop system is that the interconnection between those systems to allow for successful launches is where errors and unknown variables can appear as well, when this occurs there isn’t anything a closed loop system can do. A system can protect itself and its data but cannot prevent any external damage outside of the scope of that system.

Launch Control System

I believe it is crucial to note how my system started, below there is a very simplistic design of the original full system topology, I chose this topology as it allows for the crucial calculation crunching to happen away from the main “Hive System”. This approach was chosen on the basis that it maximises efficiency of the system controlling and adapting inputs and calculating the final resulting outputs.

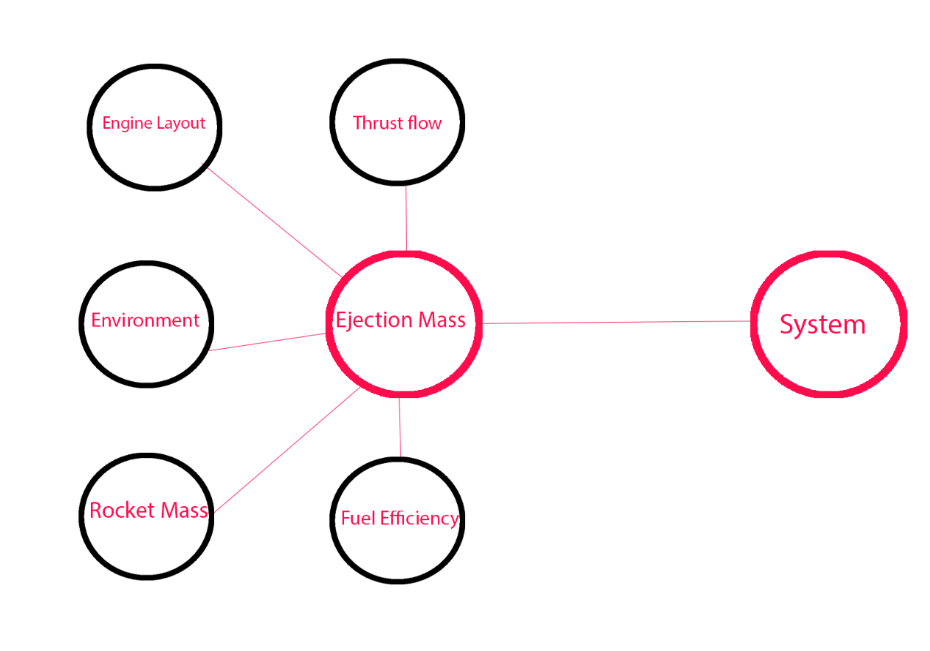
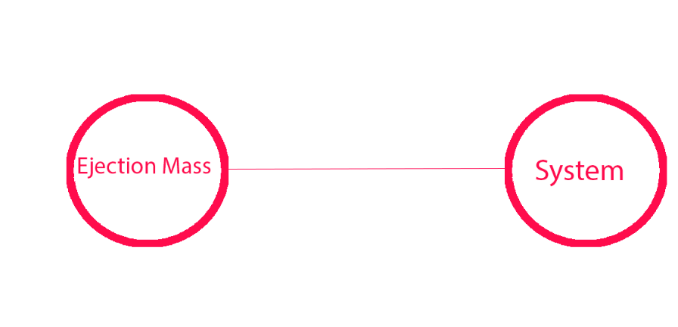


Figure 3

There were 5 sub systems that calculate resulting data and factor that in to my Ejection Mass system, this system fuses elements that are currently used as fuel sources and chooses the perfect option to use for fuel; the efficiency is then matched in to the Hive system and compared against other variables which I’ll explain in detail further on in the report. The problem with this topology is creating a new Fuzzy Inference system (FIS) every time is wasting efficiency and re-writing external data collection files means more time processing and an unnecessary input for the Hive system to compensate for.

Figure 4

This is the new and improved topology I have condensed 5 sub systems in to one sub system and 1 Hive system. I did this by changing the logical rule base or my Hive system so that it responds to incoming variables as the data is being processed for an output, this is mainly to solve the closed loop problem that occurs all too often in rocket launches, but it also allows me to a topology and make it more efficient from the last. Whilst efficiency might be lost in the Hive system overall the system will run much smoother and will be a lot more accurate in determining a successful launch.

System Configuration

My current system configuration is a lot more simplistic whilst keeping the formula output complexity, to determine accurate results. I’ll be starting with the sub system input calculations and the progression of the deterministic output for the Ejection Mass Propulsion.

Inputs/Output for Sub System

As shown in figure 5 these are the 4 inputs that represent the input data for a potential mixture that can be converted in to fuel propulsion for rockets. For this system, it is the deterministic variable that is the base of all the formulas in the system from this point, if these variables match the data that is presented then there will be a good or perfect output when evaluating the mixture; depending on this result will depend on the final output of the whole launch. My system will automatically attempt to compensate a fault in one of the input values by boosting another variable and then determining the result, but if it is a failed mixture the launch will be aborted.

[A](#_Appendices_3)

Appendix A displays the output the sub system is attempting to fulfil; this is the data that will be converted over to the Hive system based on the results; if the data is anything lower than “Low Conversion” then the data will not even be acknowledged by the Hive system since it is impossible to launch without efficient fuel, my system will repeat the process with its base input variables until it is happy with the mixture result. If the result is satisfactory then it will move on to the next determining factor using the mixture result to determine the quality of the upcoming input variables. The reason my system can do this is because the excel spreadsheet that was used to write the output data from the sub system, feeds that data directly in to the spreadsheet that hold the input data for the Hive system so it can treat that data appropriately as if it was always there to begin with.

[B](#_Appendices_4)

Appendix B defines my choice of variables that define my sub system (Ejection); are based around the data entirely; since the input values are the basis for everything in the system on first launch the ranges on the graph are sufficient in providing the exact necessary outputs for the correct formulas, the amount of inputs was chosen based on the elemental configuration of types that could be converted in to a fuel source and all the data is perfectly aligned to represent results. The reason for this is because if the graphs defined anything other than exactly where the data needs to be then there will not be a perfect mixture and the results of the base formula will be skewed; presenting false unknown variables once the output data is transferred over to the Hive system. The output ranges are specifically designed to feed directly in to the main Hive system, depending on the conversion ratio will depend on whether the data is used in the main Hive system, the system is designed to remove any data that is at a failed conversion classification. If a failed classification could be allowed in the main system it would distort all relevant values within that system concocting inaccurate results; which in my system is catastrophic if those variables are invalid.

[C](#_Appendices_5)

Appendix C shows the graphs displayed represent the physical representation of how the 4 inputs and 1 output look and where the sets align on the axis ranges. Whilst at first glance it is shown that this doesn’t look very fuzzy since the overlapping of membership functions is what create vagueness and therefor making it less of a crisp representation; and more a Fuzzy Logic representation. The graph representation is a physical display of the precision that is required for the input elements to be able to create a successful mixture; whilst it is a bad choice to have dead zones on a graph as that means certain variables cannot be catered for, in this scenario having those specific membership function in that architecture is the only way a fuel that can thrust the mass of the rocket can be created. If any variable falls within those parameters the main system wouldn’t even attempt to use that data since there is a 100% chance of failure.

I chose to use the Gaussian membership functions (MF) to display appropriate gradient and distribution towards the chamber pressure variants, since that variant is particularly important as increasing the pressure allows for the increase in the adiabatic temperature increase as well. I chose to rotate to a triangular MF since it better represents the exact precision in which the input variables need to be assigned and specifies a more precise classification, this suits the input data since that data is extremely rigid in its current state and cannot be changed much.

[D](#_Appendices_1)

Appendix D represents the rule base for my sub system, the rules are categorised in order based on the labels they have been given. “Perfect Biconditional fusion” represents a set of rules that will design the output that specifies a fuel source that is perfect in as many ways as possible with the input data provided. This data would be incorporated in the main Hive system and would identify and be the determining value for achieving the best variables in the main system.

“Regular & High Quality Biconditional Fusion” specifies lower forms of quality and the rule set uses the MFs to identify if the lower forms of quality are acceptable for the main hive system to absorb and use as a base formula for future variables that might not exist yet.

“Low Biconditional Fusion” is the rule classification that defines the lowest form of conversion that is eligible data to be used in the main Hive system, the main system will have to adjust variables to compensate for a poor conversion rate of fuel. For example, there is a possibility in the reduction of “thrust potential” as that would consume fuel at a more vigorous rate, if the fuel was of a higher conversion rate the quality of the fuel would allow for more consistency in thrust flow. However, there is no luxury with a low conversion.

“Failed Biconditional Fusion” has a larger rule base to filter data that cannot be transferred and used in the main Hive system and data that can be used; whilst some of the data may fail in fusion not all the element combinations will fail and some will classify under other categories and they can be used in the main system. There is a very basic rotation used to identify the rule, the first 2 rules are lead and the other two rules after that are after effects or added conditions. I cannot simply lower the rule strength since I need all four so using a rotational 2 pair system works instead.

I used the ‘LOM’ defuzzification method since most of my MFs lie to the left most extreme it’s logical to accommodate all those MFs in the output when it is compared against the rule base, if I had used ‘MOM’ or ‘SOM’ it would have no intercepted any MFs and would have given me very inaccurate values to put forward to my main system.

Hive System

[E](#_Appendices_2)

Appendix E is introducing all 13 inputs for my main Hive system, these variables are set with more appropriate membership functions than my sub system since the data can be a variety of information, as well as that the data that the system starts with will more than likely change in most scenarios as the system will develop unknown variables that were calculated based on the propulsion formula chosen by the sub system. The design of this system is to evaluate the data from the excel spreadsheet with the new data the system has received on propulsion and design inputs around the propulsion quality, all the decisions the system makes is based on the quality of that conversion. One example would be, the Km/s graph and the mass of the rocket, these two variables will have a different MF chosen if the quality of the fuel conversion is higher; since the burn time on the rocket is likely to always start at stage 1 the amount of Km/s that is achieved can also be based around the amount of fuel as well, if the quality is higher, then the Km/s is more resulting in a faster burn time; if the Km/s is higher; the rocket can have a greater mass since there is a faster velocity increase resulting in faster burn time which could allow for stage two to be reached. The overall result of this would mean the rocket passes the Exosphere into the vacuum of space; with a good amount of resources.

Another resulting conclusion is that if the fuel conversion is of a lower quality but is still accepted by the Hive system, then the Km/s could be lower and then allow for more thrust potential and a slow burn time build up to conserve fuel as well as choosing an appropriate engine infrastructure (Merlin 1C) to allow for optimum thrust but minimal exponential waste. Since Gravity will be applied to the mass of the rocket, if the rocket is lighter there will be less stage 2 burn time to allow gravity to re-align the rocket since it will launch on earth’s rotation at 1674.9 Km/h. This configuration might be an example the system would pre-configure before giving an output to allow for adjustments since it recognises low fuel conversion. The Hive system may also recommend a higher weighting in fuel with a lower rocket mass and uses the standard 2 stage rocket (Falcon 9) to allow for safe departure past the Exosphere.

System Preparation

The Hive system has the capability to run checks based on testing data that has been added, the test data can be evaluated against values that contribute to the launch of the rocket, these values have the same importance and meaning as the actual values because if appropriate checks are not conducted before launch the system may abort entirely. The graphs and membership functions that conclude this data are the “Thrust vacuum” and “specific impulse vacuum”, these inputs are designed to calculate a comparison judgement on what the system believes would be a good choice on engine configuration and what specific impulse is best to configure on the engine the system has chosen. Since I am using the Merlin Family for rocket engines; the system may choose Merlin 1D based on the mass of the rocket since it has the highest recorded impulse which would contribute to a better burn time overall, the vacuum data may evaluate to using a different engine configuration but the system makes a final decision based around other variables: mass, gravity, thrust, amount of fuel, and fuel efficiency.

Thrust vacuum data specifies ranges that can be achieved in a vacuum for thrust based on numerous testing on engine configurations and testing different engines from the Merlin family, the real data is evaluated based on mathematical probability and conclusion, which determines the thrust required by the distance of each Ozone layer times by the thrust value of the engine from sea level times by the specific impulse of the engine. The real data from the mathematics is then evaluated against test data performed in a vacuum and the resulting pair is considered by the system and the system will then use a deterministic variable to decide whether the vacuum data is correct in the current configuration of a launch, if not the system will provide a better option, if the system does agree with the test data then the system will use that in its variable calculation. Sometimes the vacuum data is not required, the system will only evaluate this data if it is not certain of a definitive engine configuration to use.

There are some hard check variables that cannot be overlooked in the system, one of these variables is the ‘Weather’, this is a very important variable since it is not one that can be controlled but is one that is evaluated on the day of the system checks for the launch. If the weather is sunny or cloudy then the rocket is capable of launching, however if the weather specifies rain fall/storms or snow then the launch will automatically be aborted, the only exception is wind, if it is windy there is a possibility to launch. If it is windy the system will re-evaluate other variables to accommodate this change in weather, if the system can find an optimal solution then it will adjust the necessary values and allow the launch under those specific pretences, if the system cannot find an optimal solution then it will not allow the launch until that variable is defined as an acceptable adjustment; for example, “sunny”.

[F](#_Appendices)

Appendix F generalises and filters the inputs in to 5 separate outputs that are a final system check to evaluate if the rocket can launch or not, whilst this may not be necessary I opted to add one more evaluation at the very end to make sure there were no miscalculations at any step of the system; therefore, I chose to evaluate using 5 outputs; I could have used 1 output and use that value to evaluate whether the system can launch or not. I decided to presume the latter was a more appropriate choice of safety, the output would have been whether the rocket could launch or not, with this these new checks in place the system can deter any errors or unknown variables that the system could develop that may not be necessary. These checks are necessary as the original design for the system was to only launch the rocket but since the system would control all closed loop systems, it sets the variables for those systems as well and therefor, also projects the variables required to not only launch but also travel into space.

**Fuel Consumption** is an output that evaluates against the quality of the fuel formula submitted by the sub system, it then bases the conversion percentage with the amount of fuel that will be used and gives a consumption percentage rate, this rate checks against the other input variables such as “Velocity” and “Burn Time”.

**Fuel Loss** is a gauge to determine how much out of a tank of fuel would be lost by increasing thrust potential and decreasing the spread from the flow of the thrust to allow for an exertion of gravity to realign the rocket to accommodate for earth’s rotational speeds of 1674.99 Km/h. This output checks the inputs for the rocket mass as well as the counter mass depending on how many stages a rocket will have. If the rocket has 2 stages then the depleted fuel tanks will be removed so the system will calculate that those tanks can be used for gravitational exertion and will be detached at a certain period.

**Acceleration Over Time (AOT)** refers to the calculation formulas of the inputs for “Velocity” and “Burn Time”, depending on the MFs chosen for these inputs, the output will be decided in AOT, this is an important output because this is the output that can be modified to make up for other variables or a poor fuel conversion formula. For example, if the fuel conversion is poor the system may lower Velocity and choose a stage 1 burn time to provide a smoother transition of AOT to reduce fuel consumption.

**Thrust Potential** depicts the combination of the mass of the rocket and the engine configuration allowing for a better and more concise thrust flow, this output defines whether the rocket has the potential to reach past the exosphere and is evaluated against every variable and the chosen MFs for those. If the engine configuration is correct but the mass of the rocket is too heavy then the thrust potential will be lowered and if the system evaluates that potential as too low then it will not launch.

**Abort** the final output was intended to be the sole output for the system until I decided to add another layer of validation to correct any final variables. The Abort is simply an override which will allow the system to force an abort if required at the last moments, more of a final safety precaution.

[G](#_Appendices_6)

Appendix G distinguishes my reasoning as to why I chose the membership functions that I chose for my system. For Velocity, Fuel Efficiency, Gravity, Thrust, Earth Rotation, and Launch Abort, I designed them using the membership function “trapmf” or a trapezoid, I chose to do this as the data that is configured in the system is required to hold a maximum membership to a specific set for more than a period of an apex on the graph. For example, Velocity has a MF that defines it that the Km/s that the rocket would be travelling at would not allow it to exit past the Exosphere and therefor fail in its launch attempt, the degree of membership to this set must stay at 1 whilst there is no chance of the rocket achieving the velocity required by adjusting other variables. The reason why it starts to decline when it approaches the next MF is because by adjusting other values; for example, increasing the Burn Time would boost the Km/s the rocket is travelling sending that rocket in to a MF that would allow it to reach orbit (Low Km/s).

Acceleration Over Time, Counter Mass, Fuel Consumption, Specific Impulse Vacuum (Vac) and Sea Level (SL), and Burn Time are represented by “zmf” and “smf”, I chose these MFs because the sets require a slow gradient for increasing the degree of membership but then they must stay at that range until the next set is invoked, this is a one-time variable that would be applied whilst the rocket is being guided in to space since it achieves maximum degree of membership and then holds that apex.

Fuel, Mass, Thrust (Vac), Thrust Potential, and Weather Condition are designed by “trimf” as the degree of membership at a certain period will require a one-time apex and then a slow gradient in to the next invoked MF, since thrust is controlled by the Burn Time as one of its variables when the stages switches for that variables there will be a peak where the current set will reach its apex and then slowly traverse in to the next set.

[H](#_Appendices_7)

Appendix H uses a rule base to condition the inputs and outputs of the main Hive system, to clarify how my system is conditioned is as follows, the inputs taken from the sub system and from the main Hive system are traversed in to an output, this would be launch or abort. The system then uses the other 4 outputs to evaluate the original output of the system to make sure the output is correct, if the output is incorrect then the system re-evaluates and starts over. The other 4 outputs are another form of validation to make sure the outputs are correct.

The rule base then conditions the inputs and outputs by evaluating the membership functions to the excel data, once the rule base is completed its conditioning then the boundary checks take over. The boundary checks are operating to evaluate the current rules the system has chosen, if the rules are incorrect that the system has chosen when the rules are compared against the boundaries; if there is a problem the system will have to re-evaluate its rule set, so there is a rule base to condition and evaluate another rule base. Whilst this may seem extreme it is imperative that the results outputted are not incorrect as it could be disastrous in practicality for what this system is designed to control.

The defuzzification method chosen is ‘Centroid’, I chose this method because it provided the most accurate output whilst incorporating data from the sub system, as the data from the sub system evaluates an output currently at ‘75%’ for 11 outputs and ‘99%’ for 1 outputs, since the data is used was real data that is used for rocket launches, the output must be capable of evaluating this data correctly.

The rule base in the main Hive system has decreased from its original value of 200 – 300 rules, the reason for this is because I used the base formula from the sub system as a basis for the outputs on the main system and conducted the necessary conditioning for that in the sub system which is why the sub system rules are far longer in comparison to my main system, I found this method of generalisation to be most effective in the entirety of the design implementation of my system.

Testing

[I](#_Appendices_8)

Appendix I depicts the difference in the results of the data with and without the boundary checking rules, specifically looking at input 2 and input 3, these two inputs differentiate the difference in the boundary checks, the initial result without these checks shows the system defining input 3 as all values averaging to ‘0.19’, this is not the correct average for this scenario; when the system chooses the rule to condition the inputs and outputs by; without a boundary checking rule base as well it give incorrect answers, this is also depicted in my sub system as well by an anomalies checker which evaluates the rule base to make sure it is conditioning the inputs and outputs correctly.

This test also defines how the system can re-evaluate itself and choosing a more appropriate set of rules to condition the inputs and outputs; whilst the overall result may be correct for some of the outputs, the goal of the system is that it can create outputs and variables that could be evaluated in by the current closed loop systems allowing them to function so that the whole rocket itself can launch in to space, this is where we need boundary checks because we need the system to predict the outputs that would suit those closed loop systems without pre-defined variables.

[J](#_Appendices_8)

Whilst testing and re-defining the excel data to test my system on whether it will still evaluate and re-balance inputs to allow for appropriate results. I found some interesting results; whilst the system does re-balance to some degree eventually it will either ignore that data entirely if the data is skewed to the point where it cannot recover it by adjusting other inputs. This could cause problems if the data is too far off since it could skip an input and still give an output that would allow a launch to proceed; obviously, this is an issue that would require the input data to be completely wrong to begin with. However, I have new found my system’s boundaries and to what point it will register that data input, the fact that it completely ignores the data and evaluates the system without it is interesting; I did not program it to do that and yet it has allowed itself to be dependable to a point of losing an input would still create a result.

Would the data still be accurate if that input is ignored? That is a very interesting prospect. After further testing I have concluded that currently it is not using the data from the input, however it is using it to re-balance other variables so that the output can still be valid, this is not what I was expecting for a result and I’m not entirely certain where it learned the capability to do it either, I have backtracked the entirety of the code and I am yet to find a defining feature that is telling the system to dispose of that data but use it as a type of plugin variable if one of the other outputs need a re-adjustment.

Checking the sub system data has proven that this main Hive feature does not derive nor traverse to the sub system, whilst after data testing the sub system still provides appropriate outputs, the outputs become inaccurate only after the ability for the elements to create a mixture is lost. This is a perfect result as it allows my sub system to work completely as intended.

[K](#_Appendices_8)

After preforming some tests on the rule base, itself I found that changing the operator to ‘OR’ on the specific rule section that catches anomalies did change one of the outputs within my sub system. Since the rule is now firing without the need for extra condition because of the ‘AND’ operator the ‘anomaly’ rules can fire with only one of the conditions being met, this has coincidentally changed my first output from its ‘99%’ fuel mixture strength to only ‘24%’, this would ultimately lead to a low mixture output which would disrupt the main system’s data output. The other values were unchanged from the operator change in the anomalies section of the rule base, meaning there isn’t another anomaly currently in the system. After subjecting the excel data to testing I have found that changing the data to match what the anomalies would catch allows the system to compensate for that in the output, this is working as intended and if data does get through it will not affect the output of the sub system. Therefore, not affecting the output to the main system either.

[L](#_Appendices_8)

Appendix L is performing a small test on displaying the outputs on the main system to see whether the output would change just by a simple change in the range of data coming from the sub system. The test proves that even changing the range of the data is enough to affect the systems output in a drastic way by throwing the output data off, whilst the system can readjust itself, it is not capable of adjusting the variables if the base formula it creates outputs from is incorrect.

Critical Evaluation

The outcome of the system I am very pleased with and it makes me more enthusiastic about Fuzzy Logic as a general topic and concept of research, whilst all these values are what I have gained from the design of this system, there are some components of the system that could be improved overall to increase efficiency and potentially even dependability in the grand scheme of the project. For example, I used a topology of implementation that allowed me to calculate and distinguish data outputs and support a main system so that the main system would not have to do any complex calculations since I wanted to evaluate it so heavily. This theory has proved effective overall but could be improved as well. If I had used one more sub system to relieve half of the inputs from the main system it would save on efficiency by 20%-30%, therefor allowing for more productive results whilst potentially allowing for more accurate results in the outputs as I could perform more advanced calculations, since the performance complex of the system is divided in to thirds.

Distributing some of the input data load to another sub system would have also allowed me to create a more rigorous rule base, I doubled my rule base by having boundary checks, however, if I could distribute that rule base to another sub system I would be able to designate a rigorous rule base which would define a baseline formula just like my current sub system and would allow me to shorten the rule base in my main Hive system, this would create more efficiency and allow the main Hive system to make more accurate decisions as it has less variables to process but has the same amount of data acquired as it does now.

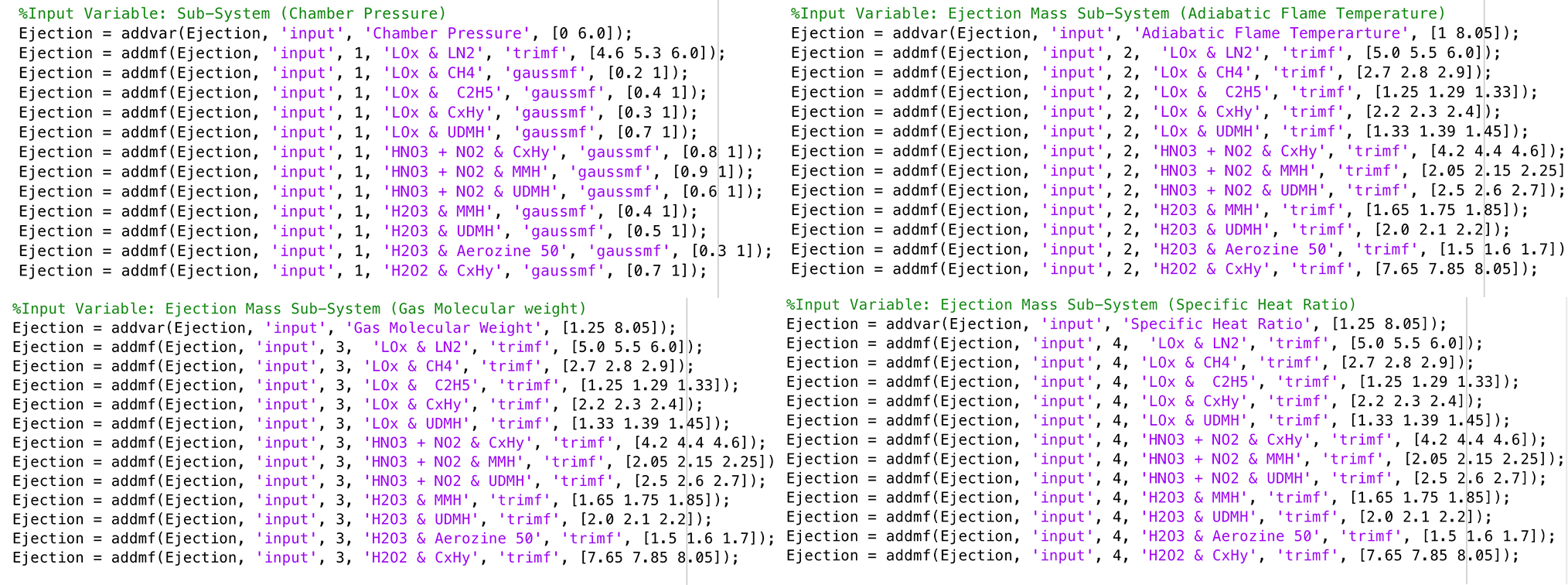
Conclusion

Evaluating my decision on the conclusion for my system, I would say that it is an overall success, the system works as intended; it is capable of using formulas to launch a rocket, one of the endeavours I set out for; which would be the system’s capability to redistribute input data to allow for accurate results if one input was lacking for a successful launch. Whilst my testing yielded some very strange, yet interesting results I am pleased that even with the system reacting that way it still can produce an output that is accurate and will predict whether a rocket can launch or not. Furthermore, I allowed my system to flourish to its potential with the data I could gather on the system; this system could a lot larger in the future and span to many closed loop systems, essentially having a near endless number of variables and calculations. However, even defining the data restrictions I am still surprised that the system can produce an output that is accurate from the sub system formula, I was uncertain as to whether it would interact and develop the data correctly by using a formula from a sub system, but It seems to provide relevant data.

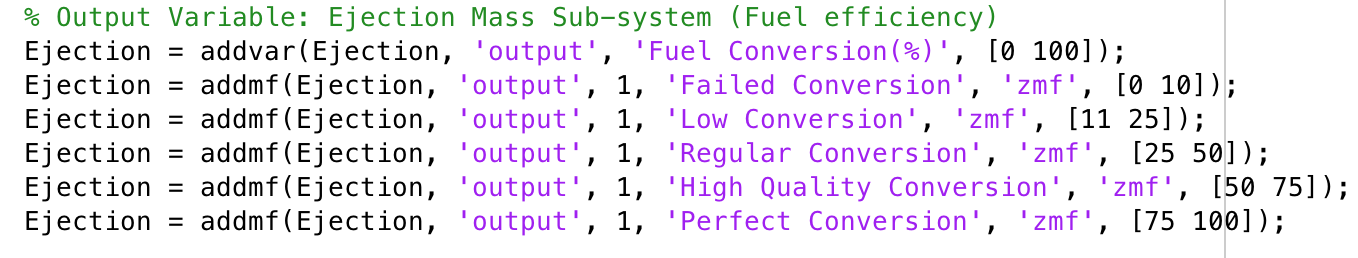
In my opinion if the data from the base formula was skewered in a certain way or an element was changed out I think the system would struggle to re-align itself since the system’s ability to re-evaluate itself does not traverse to the sub system, meaning it is entirely possible as shown in the testing to throw the system off if the base formula is incorrect.

# Appendices

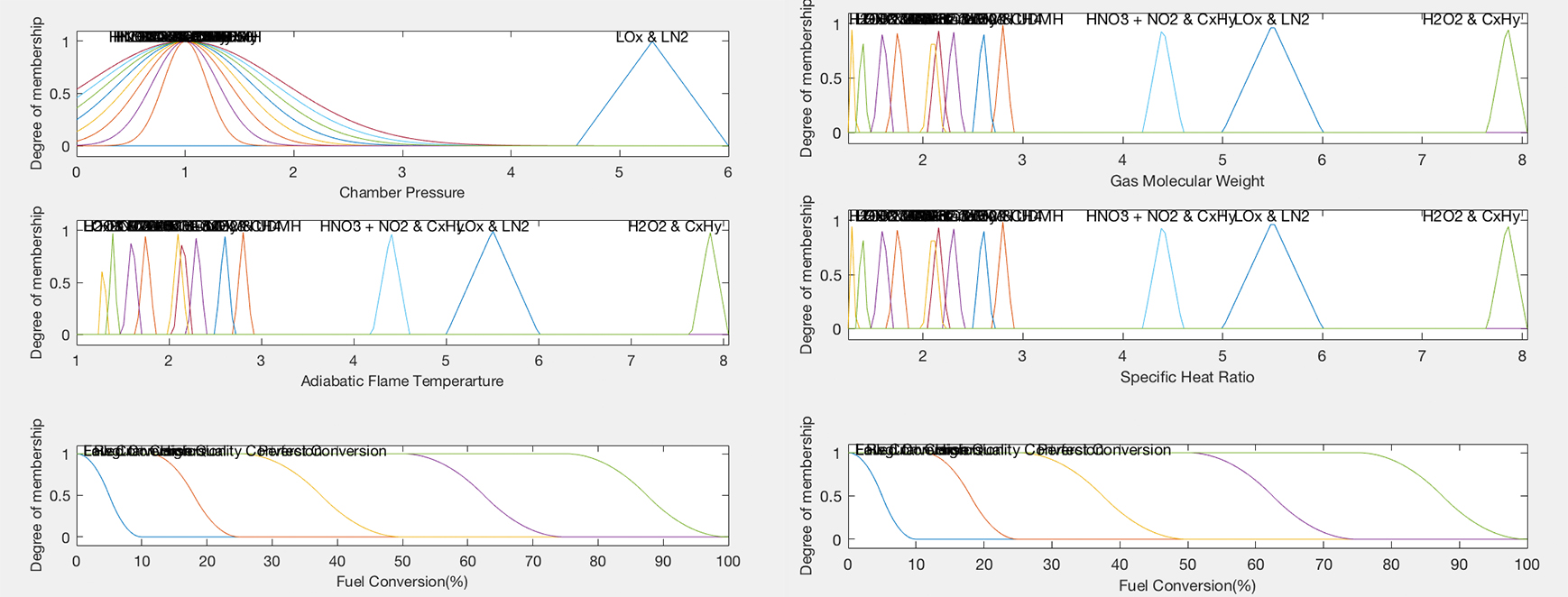
A



B



C



D

%Rule base for Elemental Fusion

%Perfect Biconditional Fusion

rule1 = [1,1,1,1,5,1,2];

rule2 = [2,2,2,2,5,1,2];

rule3 = [3,3,3,3,5,1,2];

rule4 = [4,4,4,4,5,1,1];

rule5 = [5,5,5,5,5,1,1];

rule6 = [6,6,6,6,5,1,1];

rule7 = [7,7,7,7,5,1,1];

rule8 = [8,8,8,8,5,1,1];

rule9 = [9,9,9,9,5,1,1];

rule10 = [10,10,10,10,5,1,1];

rule11 = [11,11,11,11,5,1,1];

rule12 = [12,12,12,12,5,1,1];

% Regular & High Quality Biconditional Fusion

rule13 = [1,1,2,1,4,1,1];

rule14 = [1,1,1,2,3,1,1];

rule15 = [2,2,4,3,4,1,1];

rule16 = [2,2,3,4,3,1,1];

rule17 = [3,3,1,5,4,1,1];

rule18 = [3,3,5,4,3,1,1];

rule19 = [4,4,1,3,4,1,1];

rule20 = [4,4,5,2,3,1,1];

rule21 = [5,5,4,2,4,1,1];

rule22 = [5,5,1,3,3,1,1];

rule23 = [6,6,5,5,4,1,1];

rule24 = [6,6,4,2,3,1,1];

rule25 = [7,7,1,1,4,1,1];

rule26 = [8,8,2,2,3,1,1];

rule27 = [9,9,4,4,4,1,1];

rule28 = [10,10,5,3,3,1,1];

rule29 = [11,11,1,1,4,1,1];

rule30 = [12,12,5,5,3,1,1];

%Low Biconditional Fusion

rule31 = [2,2,3,4,2,1,1];

rule32 = [2,3,4,5,2,1,1];

rule33 = [3,4,5,6,2,1,1];

rule34 = [4,5,6,7,2,1,1];

rule35 = [5,6,7,8,2,1,1];

rule36 = [6,7,8,9,2,1,1];

rule37 = [7,8,9,10,2,1,1];

rule38 = [8,9,10,11,2,1,1];

rule39 = [9,10,11,12,2,1,1];

rule40 = [10,11,12,12,2,1,1];

%Failed Bioconditional Fusion

rule41 = [1,2,2,2,1,1,1];

rule42 = [2,3,3,3,1,1,1];

rule43 = [3,4,4,4,1,1,1];

rule44 = [4,5,5,5,1,1,1];

rule45 = [5,6,6,6,1,1,1];

rule46 = [6,7,7,7,1,1,1];

rule47 = [7,8,8,8,1,1,1];

rule48 = [8,9,9,9,1,1,1];

rule49 = [9,10,10,10,1,1,1];

rule50 = [10,11,11,11,1,1,1];

rule51 = [11,12,12,12,1,1,1];

rule52 = [12,1,1,1,1,1,1];

rule53 = [2,1,2,2,1,1,1];

rule54 = [3,2,3,3,1,1,1];

rule55 = [4,3,4,4,1,1,1];

rule56 = [5,4,5,5,1,1,1];

rule57 = [6,5,6,6,1,1,1];

rule58 = [7,6,7,7,1,1,1];

rule59 = [8,7,8,8,1,1,1];

rule60 = [9,8,9,9,1,1,1];

rule61 = [10,9,10,10,1,1,1];

rule62 = [11,10,11,11,1,1,1];

rule63 = [12,11,12,12,1,1,1];

rule64 = [1,12,1,1,1,1,1];

rule65 = [2,2,1,2,1,1,1];

rule66 = [3,3,2,3,1,1,1];

rule67 = [4,4,3,4,1,1,1];

rule68 = [5,5,4,5,1,1,1];

rule69 = [6,6,5,6,1,1,1];

rule70 = [7,7,6,7,1,1,1];

rule71 = [8,8,7,8,1,1,1];

rule72 = [9,9,8,9,1,1,1];

rule73 = [10,10,9,10,1,1,1];

rule74 = [11,11,10,11,1,1,1];

rule75 = [12,12,11,12,1,1,1];

rule76 = [1,1,12,1,1,1,1];

rule77 = [2,2,2,1,1,1,1];

rule78 = [3,3,3,2,1,1,1];

rule79 = [4,4,4,3,1,1,1];

rule80 = [5,5,5,4,1,1,1];

rule81 = [6,6,6,5,1,1,1];

rule82 = [7,7,7,6,1,1,1];

rule83 = [8,8,8,7,1,1,1];

rule84 = [9,9,9,8,1,1,1];

rule85 = [10,10,10,9,1,1,1];

rule86 = [11,11,11,10,1,1,1];

rule87 = [12,12,12,11,1,1,1];

rule88 = [1,1,1,12,1,1,1];

%Anomalies - Caters for extreme measures.

rule89 = [1,6,9,10,1,1,1];

rule90 = [2,7,10,11,1,1,1];

rule91 = [3,8,11,12,1,1,1];

rule92 = [2,4,6,8,2,1,1];

rule93 = [4,6,8,10,2,1,1];

rule94 = [6,8,10,12,2,1,1];

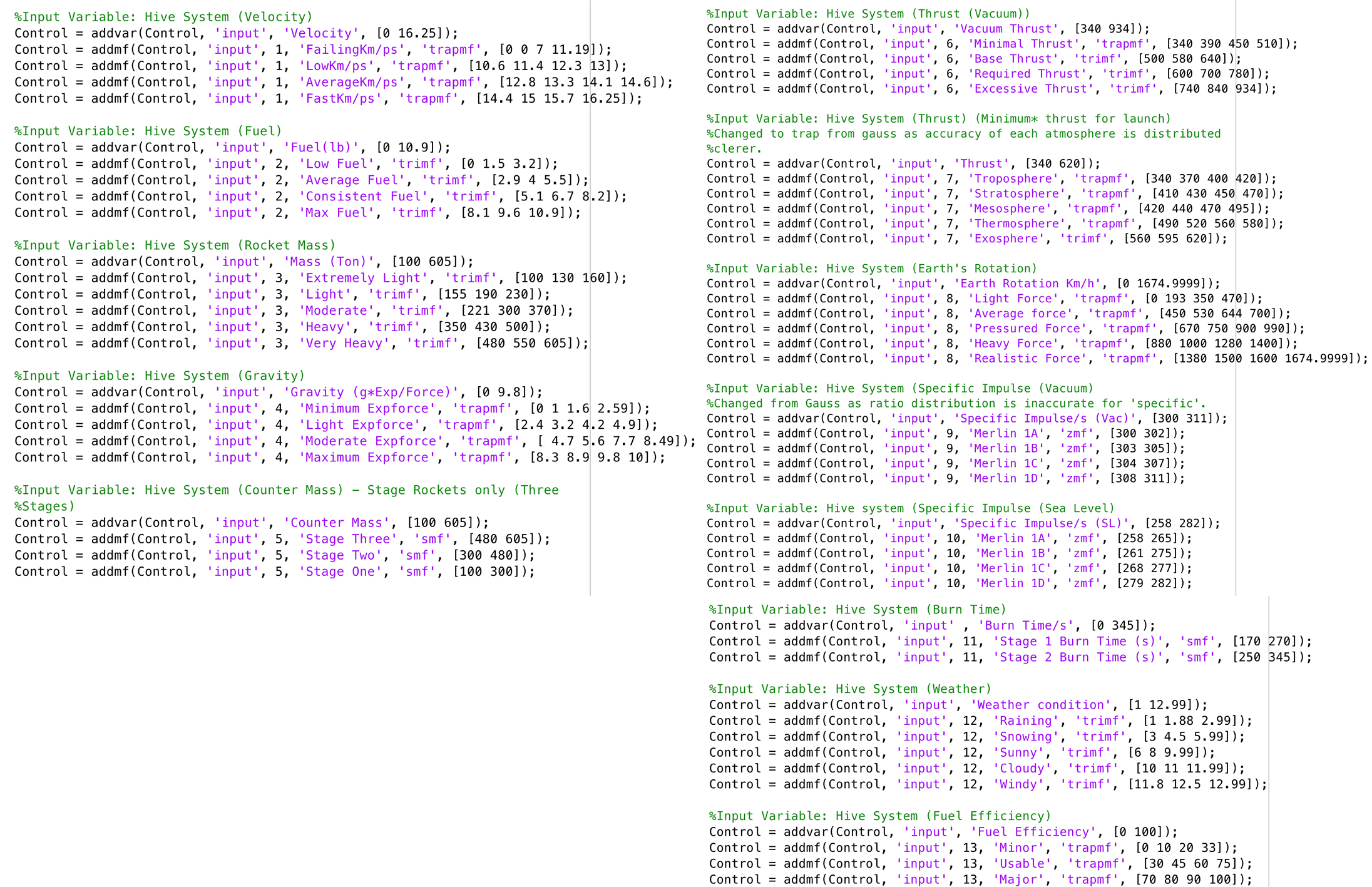
rule95 = [1,3,5,7,1,1,1];

rule96 = [3,5,7,9,2,1,1];

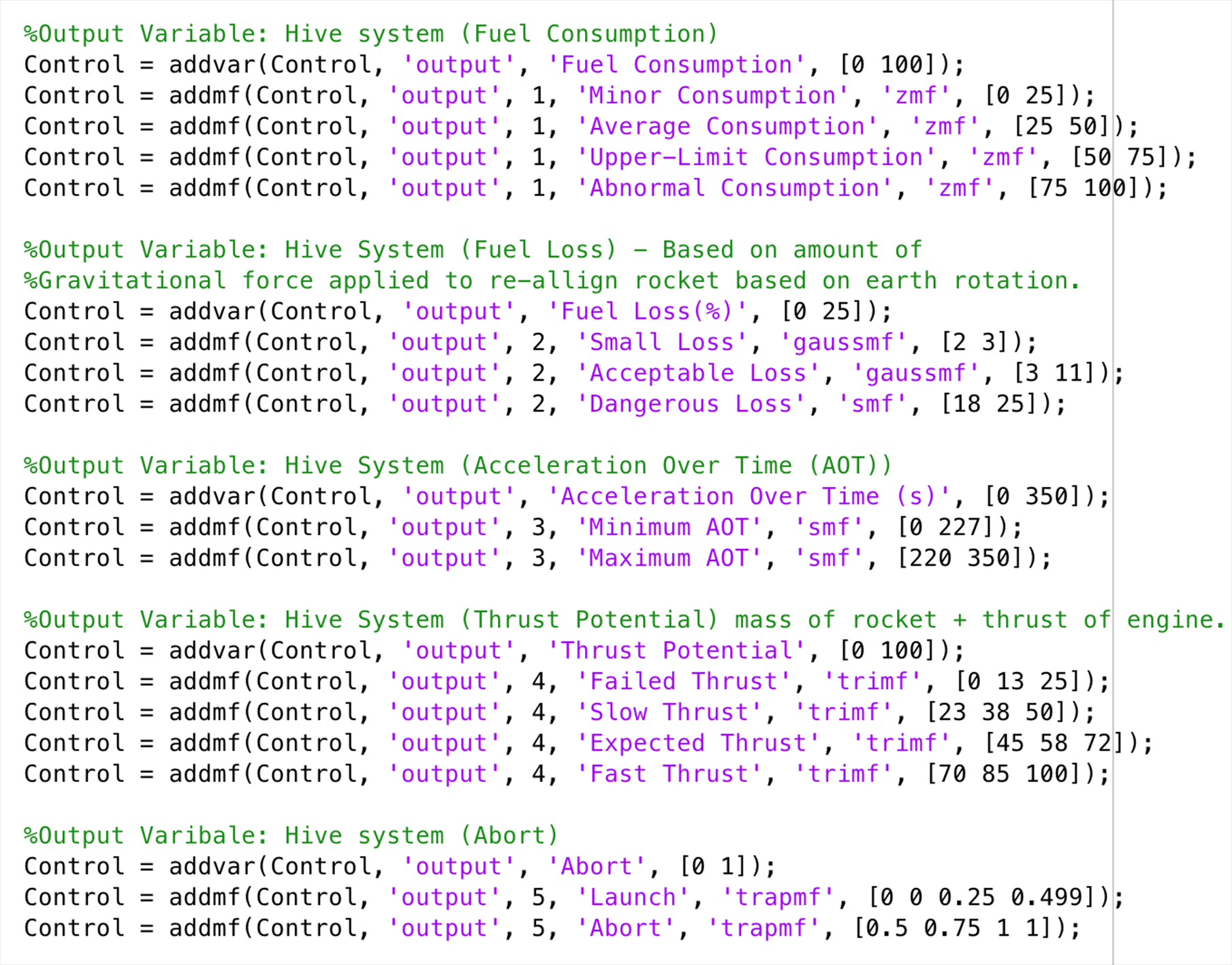
rule97 = [5,7,9,11,2,1,1];

Ejection.defuzzMethod = 'lom';

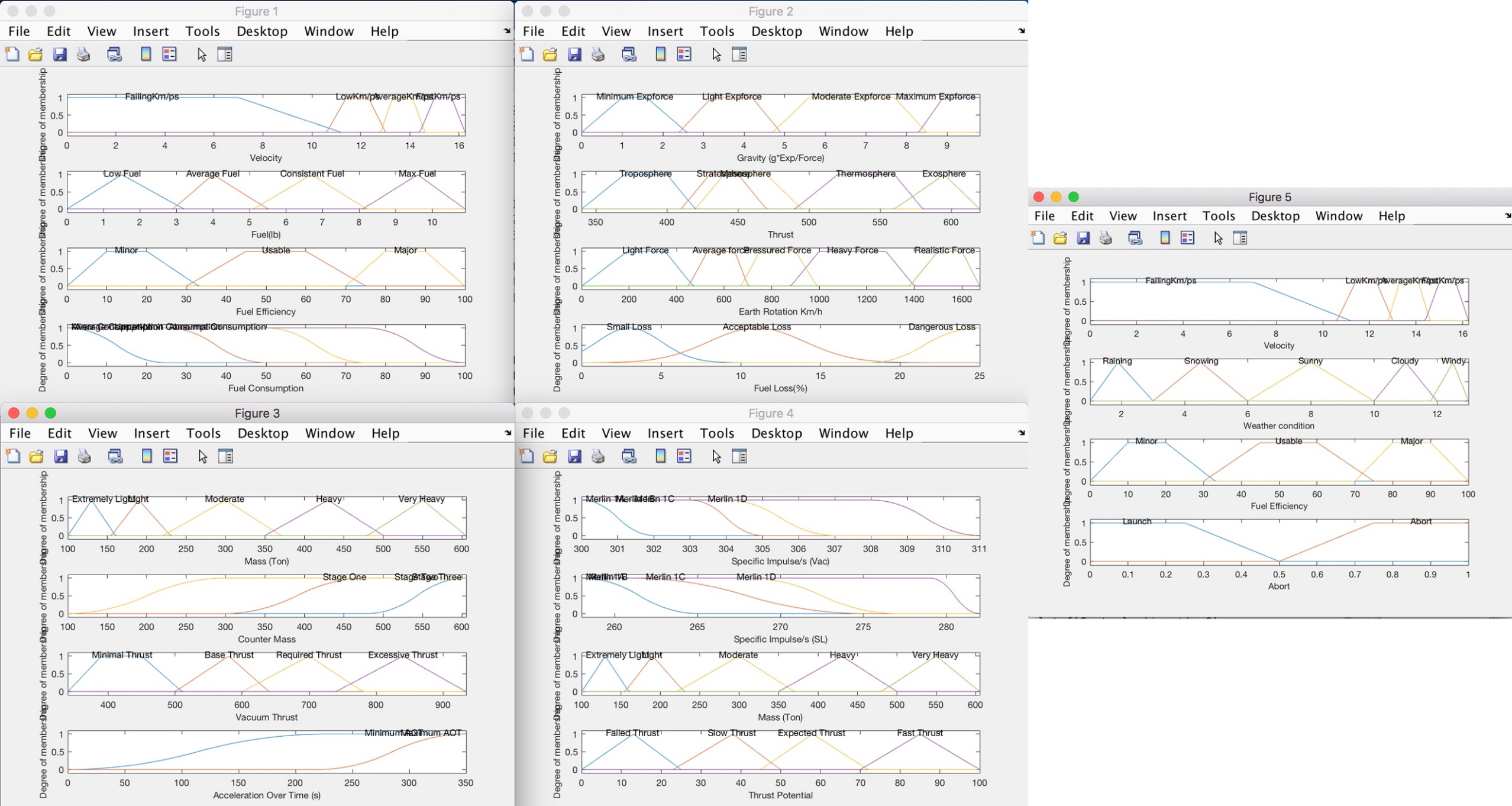
E



F



G



H

% System Checks for Hive System

%Launch on Lowest Consumption of Resources.

rule1 = [2,1,1,1,3,0,1,1,0,1,1,3,1,1,1,1,2,1,1,2];

rule2 = [1,1,2,2,3,0,2,1,0,2,1,3,2,1,1,1,2,1,1,1];

rule3 = [2,2,1,2,3,2,2,2,3,2,1,3,2,1,1,1,2,1,1,1];

rule4 = [2,1,3,4,2,0,1,5,4,4,1,4,2,1,2,1,2,1,1,2];

rule5 = [1,3,2,2,2,0,1,3,0,3,1,3,1,2,2,1,2,1,1,2];

rule6 = [3,2,1,3,2,0,2,5,0,3,1,4,2,2,1,1,3,1,1,1];

rule7 = [3,4,3,1,1,2,2,2,2,3,2,3,1,2,2,2,3,1,1,1];

rule8 = [1,2,2,4,2,0,3,2,0,3,1,4,2,2,1,1,2,1,1,1];

rule9 = [1,3,5,2,2,2,3,3,3,2,1,3,3,2,2,2,2,1,1,1];

%Launch that can safely enter the Exosphere with all resources available

rule10 = [3,3,4,4,2,3,5,5,3,2,1,3,2,2,3,2,3,1,1,1];

rule11 = [3,4,4,3,1,3,4,4,2,4,2,5,2,3,3,1,3,1,1,1];

rule12 = [4,3,2,4,2,0,1,4,0,1,1,3,3,3,3,1,1,1,1,1];

rule13 = [2,4,4,4,2,0,4,5,0,2,2,3,2,2,2,2,3,1,1,1];

rule14 = [3,2,3,4,1,2,5,3,1,3,1,4,2,2,3,2,3,1,1,1];

%Launches that cannot Launch because of minimal testing

rule15 = [2,1,3,4,2,0,1,5,0,1,2,3,1,2,2,1,3,2,1,1];

rule16 = [4,2,4,2,1,0,5,5,0,1,2,4,2,2,2,2,4,2,1,1];

rule17 = [1,2,4,3,3,0,1,4,0,3,2,5,2,2,2,2,3,2,1,1];

%Failed launches based on weather conditions

rule18 = [0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,2,1,1];

rule19 = [0,0,0,0,0,0,0,0,0,0,0,2,0,0,0,0,0,2,1,1];

rule20 = [2,1,1,4,2,0,1,5,0,1,1,5,1,3,3,1,1,2,1,1];

rule21 = [1,3,5,3,2,1,1,5,2,2,1,5,3,2,2,1,2,2,1,1];

%Launches failed based on thrust potential

rule22 = [1,4,4,4,2,1,1,5,4,1,1,4,2,2,2,1,1,2,1,1];

rule23 = [4,3,5,4,2,4,5,5,4,2,1,3,2,2,2,2,1,2,1,1];

rule24 = [0,0,0,0,0,0,0,0,4,1,0,0,0,0,0,0,1,2,1,1];

rule25 = [0,0,0,0,0,0,0,0,4,2,0,0,0,0,0,0,1,2,1,1];

%Launches failed based on high fuel loss

rule26 = [4,2,5,4,2,0,0,0,0,0,1,0,1,0,3,0,0,2,1,1];

rule27 = [4,2,5,4,2,0,0,0,0,0,2,0,1,0,3,0,0,2,1,1];

%Launches that need high fuel conversion

rule28 = [4,2,4,4,1,3,5,5,4,4,2,3,3,4,3,2,4,1,1,1];

rule29 = [3,4,5,4,2,3,3,5,4,4,2,3,3,3,2,2,3,1,1,1];

rule30 = [4,4,2,4,3,3,4,5,3,3,2,4,3,4,3,2,3,1,1,1];

rule31 = [3,3,3,4,2,4,5,5,3,4,2,5,3,3,2,2,4,1,1,1];

rule32 = [3,3,3,4,2,4,5,5,2,4,2,5,3,4,3,2,4,1,1,1];

%Boundary Checks (Rule Base Evaluation)

rule33 = [1,0,0,0,0,0,0,0,4,1,1,0,0,0,0,0,1,2,1,1];

rule34 = [2,1,0,0,0,0,0,0,0,0,1,0,0,0,0,1,1,2,1,1];

rule35 = [1,2,0,0,0,2,3,0,0,0,1,0,2,0,0,2,3,1,1,1];

rule36 = [4,0,0,0,0,0,0,0,0,0,0,0,0,4,3,2,4,2,1,1];

rule37 = [0,1,0,0,0,0,0,0,0,0,0,0,0,4,3,2,4,2,1,1];

rule38 = [4,1,0,0,0,0,0,0,0,0,0,0,1,4,3,2,4,2,1,1];

rule39 = [4,2,0,0,0,0,0,0,0,0,0,0,1,4,3,2,4,2,1,1];

rule40 = [0,0,1,4,3,0,0,0,0,3,0,0,0,0,3,0,1,2,1,1];

rule41 = [0,0,1,4,2,0,0,0,0,3,0,0,0,0,3,0,2,1,1,1];

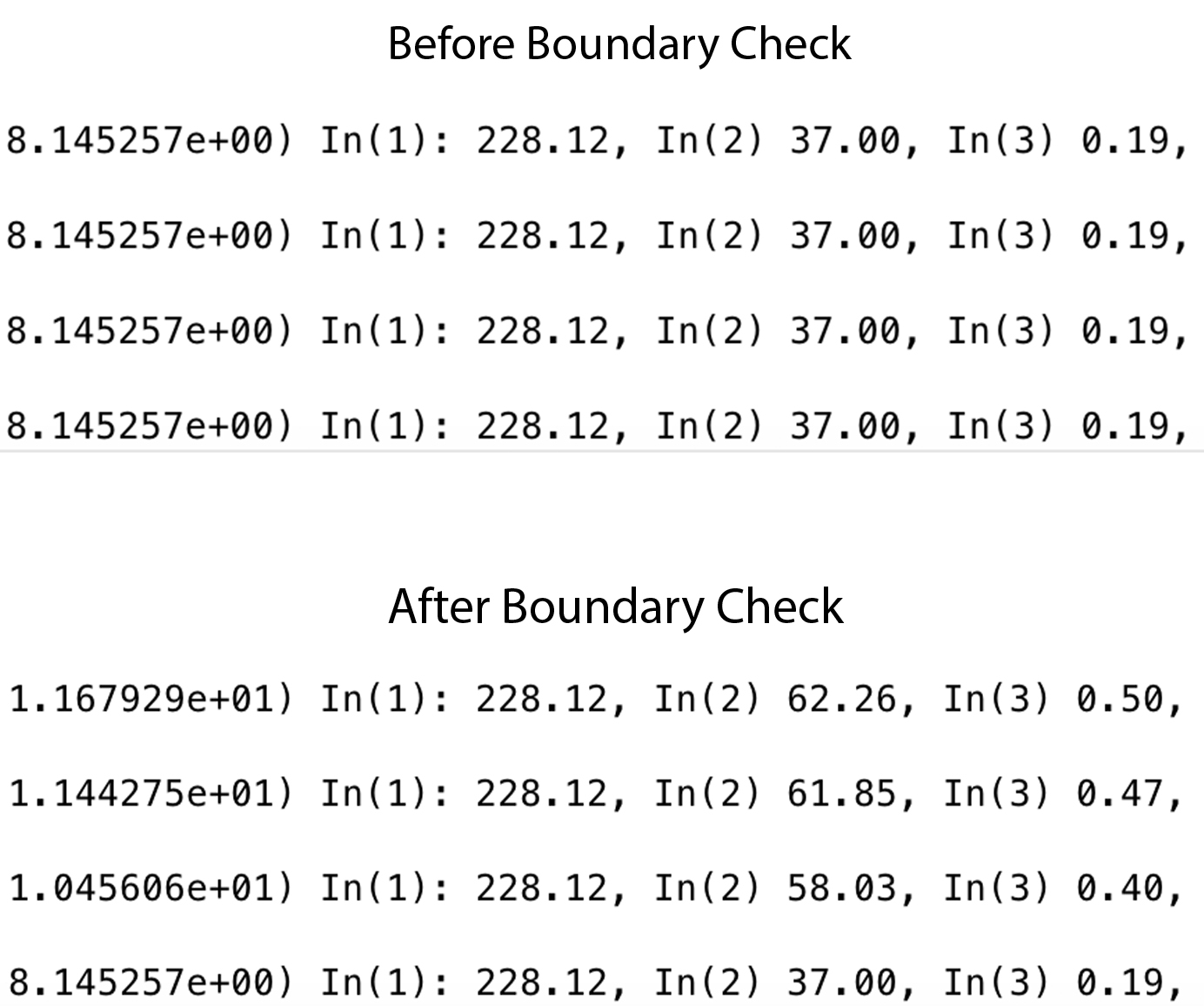
rule42 = [0,0,1,4,1,0,0,0,0,3,0,0,0,0,3,0,2,1,1,1];

rule43 = [2,4,5,0,0,0,0,0,0,0,1,0,3,1,1,1,1,2,1,1];

rule44 = [0,1,5,1,0,0,0,0,0,0,2,0,2,1,2,1,1,2,1,1];

Control.defuzzMethod = 'centroid';

I



J

1) In(1): 116.20, In(2) 0.00, In(3) 100.00, In(4) 0.00, In(5) 100.00,In(6) 340.00, In(7) 340.00, In(8) 0.00, In(9) 300.00, In(10) 258.00, In(11) 170.00, In(12) 1.00, In(13) 99.00 => Out: 18.84

1) In(1): 16.25, In(2) 0.00, In(3) 100.00, In(4) 0.00, In(5) 100.00,In(6) 340.00, In(7) 340.00, In(8) 0.00, In(9) 300.00, In(10) 258.00, In(11) 170.00, In(12) 1.00, In(13) 99.00 => Out: 18.84

In(1): 228.12, In(2) 25.30, In(3) 0.50, In(4) 2) In(1): 11.50, In(2) 1.95, In(3) 130.00, In(4) 1.00, In(5) 190.00,In(6) 360.00, In(7) 380.00, In(8) 233.90, In(9) 301.00, In(10) 260.00, In(11) 190.00, In(12) 2.00, In(13) 75.00 => Out: 41.45

In(1): 228.12, In(2) 46.18, In(3) 0.50, In(4) 3) In(1): 11.90, In(2) 2.30, In(3) 160.00, In(4) 1.60, In(5) 250.00,In(6) 390.00, In(7) 400.00, In(8) 303.30, In(9) 302.00, In(10) 261.00, In(11) 210.00, In(12) 3.00, In(13) 75.00 => Out: 38.38

In(1): 1.00, In(2) 1.70, In(3) 1.70, In(4) 1.70 => Out: 75.00

In(1): 6.00, In(2) 6.00, In(3) 6.00, In(4) 6.00 => Out: 99.00

K

%Anomalies - Caters for extreme measures.

rule89 = [1,6,9,10,1,1,2];

rule90 = [2,7,10,11,1,1,2];

rule91 = [3,8,11,12,1,1,2];

rule92 = [2,4,6,8,2,1,2];

rule93 = [4,6,8,10,2,1,2];

rule94 = [6,8,10,12,2,1,2];

rule95 = [1,3,5,7,1,1,2];

rule96 = [3,5,7,9,2,1,2];

rule97 = [5,7,9,11,2,1,2];

1) In(1): 6.00, In(2) 6.00, In(3) 6.00, In(4) 6.00 => Out: 24.00

In(1): 6.00, In(2) 6.00, In(3) 6.00, In(4) 6.00 => Out: 99.00

L

%Input Variable: Hive System (Fuel Efficiency)

Control = addvar(Control, 'input', 'Fuel Efficiency', [0 10]);

Control = addmf(Control, 'input', 13, 'Minor', 'trapmf', [0 1 2 3]);

Control = addmf(Control, 'input', 13, 'Usable', 'trapmf', [3 4 6.0 7.5]);

Control = addmf(Control, 'input', 13, 'Major', 'trapmf', [7.0 8.0 9.0 10]);

1. In(1): 11.20, In(2) 0.00, In(3) 100.00, In(4) 0.00, In(5) 100.00,In(6) 340.00, In(7) 340.00, In(8) 0.00, In(9) 300.00, In(10) 258.00, In(11) 170.00, In(12) 1.00, In(13) 99.00 => Out: 38.84

M

%Boundary Checks (Rule Base Evaluation)

rule33 = [1,0,0,0,0,0,0,0,4,1,1,0,0,0,0,0,1,2,1,2];

rule34 = [2,1,0,0,0,0,0,0,0,0,1,0,0,0,0,1,1,2,1,2];

rule35 = [1,2,0,0,0,2,3,0,0,0,1,0,2,0,0,2,3,1,1,2];

rule36 = [4,0,0,0,0,0,0,0,0,0,0,0,0,4,3,2,4,2,1,2];

rule37 = [0,1,0,0,0,0,0,0,0,0,0,0,0,4,3,2,4,2,1,2];

rule38 = [4,1,0,0,0,0,0,0,0,0,0,0,1,4,3,2,4,2,1,2];

rule39 = [4,2,0,0,0,0,0,0,0,0,0,0,1,4,3,2,4,2,1,2];

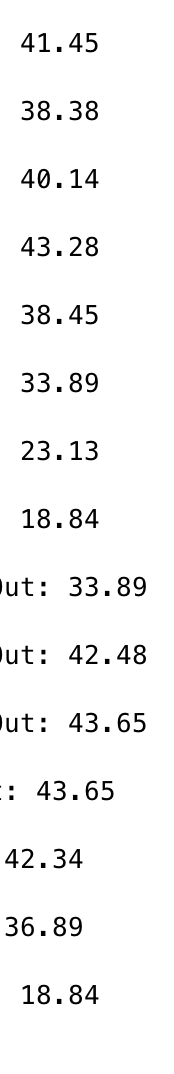
rule40 = [0,0,1,4,3,0,0,0,0,3,0,0,0,0,3,0,1,2,1,2];

rule41 = [0,0,1,4,2,0,0,0,0,3,0,0,0,0,3,0,2,1,1,2];

rule42 = [0,0,1,4,1,0,0,0,0,3,0,0,0,0,3,0,2,1,1,2];

rule43 = [2,4,5,0,0,0,0,0,0,0,1,0,3,1,1,1,1,2,1,2];

rule44 = [0,1,5,1,0,0,0,0,0,0,2,0,2,1,2,1,1,2,1,2];



# Bibliography

[1] TANG, X. and TAO, G. and WANG, L. and STANKOVIC, J. A. (2004) Robust and adaptive actuator failure compensation design for a rocket fairing structural-acoustic model. IEEE Transactions on Aerospace and Electronic Systems, 40 (4), pp. 1359-1366.

[2] SUTTON, P.G., BIBLARZ, O. (2016) Rocket Propulsion Elements. 9th ed. N/A: John Wiley & Sons.

[3] RAMAMURTHI, K. (2010) Rocket Propulsion. India: Macmillan Publishers India Limited.

[4] MISHRA, DP. (2017) Fundamentals of Rocket Propulsion. Global: CRC Press.

[5] YAN, Q. et al. (2018) Nanomaterials in Rocket Propulsion Systems. Amsterdam, Netherlands: Elsevier.