Engineering and Statistical Analysis of Solid and Liquid Missile Systems

By

Noel Cervantes

A dissertation submitted to the Graduate Faculty of
Auburn University
in partial fulfillment of the
requirements for the Degree of
Doctor of Philosophy

Auburn, Alabama
December 10, 2022

Keywords: Missiles, Data Science, Engineering Analysis, Statistical Analysis, Model Robustness/Sensitivity, and Model Explainability

Abstract

Engineering and statistical analysis is applied to gain insights and information regarding liquid and solid missile systems. This dissertation outlines five main key topic areas, physics modeling, data generation, model generation, model robustness/sensitivity, and model explainability. The Auburn University Liquid Rocket Code (AULRC) is used to generate waypoint parameters such as time of flight. Models are then generated to predict time of flight and the performance of said model is assessed using various metrics. Data generated from the AULRC is used for quantification (regression) purposes throughout this dissertation. Similarly, the Auburn University Solid Rocket Code (AUSRC) is used to generate waypoint parameters such as max thrust, however, the AUSRC is used to build classes of rockets for classification models. Both programs use a Latin Hypercube (LHC) algorithm to randomly generate data between minimum and maximum values.

Regression models developed will be classical linear regression including ridge and lasso and will include higher order interaction terms and more complex models developed include neural networks (NNETs). Classification models developed will be classical linear/quadratic discriminant analysis and NNETs. All the models were built using either Scikit-Learn or TensorFlow with Keras generated using Python. The classification models assume that all the data is available while in real-life scenarios this is not always true. Therefore, parameters are chosen to be "missing" and are replaced using imputation methods. The sensitivity and robustness of the models can be assessed by evaluating the classification metrics on the imputed data. To explain models and their predictions, the feature contributions can be assessed to see which parameters are most influential and how the model makes a prediction. For the classical models, coefficients can be used to assess which parameters had the most influence, but for NNETs, the SHapley Additive exPlanation

(SHAP) values can be used to determine which parameters were most important in the model and assess how model makes predictions.

Acknowledgements

I would first like to thank Dr. Roy Hartfield for his extensive guidance through both my master’s and doctoral research degrees. I greatly appreciate Dr. Hartfield's mentorship and willingness to guide me when the research times were tough. Secondly, I would also like to thank Dr. Mark Carpenter very much for his extensive guidance and mentorship throughout my doctoral research. With his help, I was able to quickly build my statistics, data science, and machine learning skills. I would also to thank my other committee members Dr. Majdalani and Dr. Taheri for their recommendations and knowledge background. Also, I would like to thank Dr. John Burkhalter for his help in understanding the Auburn University Liquid & Solid codes and for helping me debug many Fortran issues. Lastly, I would like to thank my family and friends for all their support when I needed it most.

Table of Contents

List of Tables

List of Figures

List of Abbreviations

A_e	=	Nozzle Exit Area
A_b	=	Propellant Burn Area
A_t	=	Nozzle Throat Area
a	=	Minimum Value on Interval
a_b	=	Propellant Burn Rate Constant
AULRC	=	Auburn University Liquid Rocket Code
AUSRC	=	Auburn University Solid Rocket Code
b	=	Maximum Value on Interval
C^*	=	Characteristic Exhaust Velocity of Propellant
$C_{T,Prop}$	=	Propellant Thrust Coefficient
d	=	Number of Inputs to AULRC or AUSRC
F_T	=	Thrust, (Pound-Force)
$F_{T,SL}$	=	Sea Level Thrust, (Pound-Force)
GL	=	Grain Length
h	=	Star/Spoke Length
I_{sp}	=	Specific Impulse
$I_{sp,prop}$	=	Propellant Specific Impulse
L	=	Length
Lbm	=	Pound Mass
Lbf	=	Pound Force
L_B	=	Missile Body Length
L_{Cone}	=	Conical Nozzle Length
L_F	=	Fractional Nozzle Length
L_N	=	Bell Nozzle Length
LHC	=	Latin Hypercube Sampling
M_{CG}	=	Compressed Gas Mass
$M_{Fuel,Tank}$	=	Fuel Tank Mass
m_{fuel}	=	Fuel Mass

$M_{Ox,\ Tank}$	=	Oxidizer Tank Mass
m_{ox}	=	Oxidizer Mass
N	=	Number of Star/Spoke Points
N_S	=	Number of Samples
n	=	Burn Rate Index
P_e	=	Nozzle Exit Pressure
P_{CG}	=	Compressed Gas Pressure
$p_{e,\ prop}$	=	Assumed Nozzle Exit Pressure
pdf	=	Probability Density Function
p_o	=	Chamber Pressure
$p_{o,\ prop}$	=	Assumed Chamber Pressure
R_{CG}	=	Gas Constant for Compressed Gas
R_I	=	Inner Propellant Radius
R_P	=	Outer Propellant Radius
r_{body}	=	Missile Body Radius
r_t	=	Tank Radius
SHAP	=	Shapley Additive exPlanation
SLHC	=	Scaled LHC Values on Interval [a,b]
t_b	=	Burn Time, Seconds
Unif	=	Random Uniform Values on [0, 1]
V	=	Fuel or Oxidizer Volume
x_{grain}	=	Starting Point of Propellant Grain
Z_U	=	Random Uniform Values
Z_N	=	Random Normal Values
6-DOF	=	Six Degrees of Freedom
α	=	Conical Nozzle Half Angle
Δp_{CG}	=	Compressed Gas Pressure Regulator
ε	=	Star/Spoke Angular Fraction
γ	=	Specific Heat Ratio
λ	=	Nozzle Divergence Correction Factor

μ	=	Mean (Average) or Expected Value of X
π	=	pi, Mathematical Constant
θ	=	Star/Spoke Angular Opening
ρ	=	Fuel or Oxidizer Density
ρ_b	=	Propellant Density
σ	=	Standard Deviation
∞	=	Infinity

I. Introduction

Data science applied to engineering is a relatively new emerging field in engineering analysis. The driving factor for engineering and statistical analysis is to gain new information about engineering systems so that either designs can be improved and/or better decisions can be made. There are several data analysis fields such as data science, machine learning, statistical analysis, and artificial intelligence and from here on, the term data science will be used to assume that any of fields could be used interchangeably. Data science is used in this work to derive information from missiles using solid or liquid propulsion. This dissertation will outline the data science methods used to apply engineering and statistical analysis. Before any data science method can be used, data itself is needed and to generate data the Auburn University Liquid/Solid Codes (AULRC, AUSRC) are used to generate missile designs. Engrained in the codes are the data sampling algorithms which randomly generate designs. Data generation methodology is extremely important as sample distributions can affect the performance of data driven models. As part of data generation, the summary statistics will be introduced to show various metrics such as standard deviation and variance. Various statistical plots will be used to show the summary statistics such as scatter matrices and box plots.

A large portion of this dissertation will be defining the models that can be used for regression and classification. Regression referring to a model which generates a continuous output such as time of flight as a function of launch angle. Regression methods such as linear, ridge, and lasso regression are introduced as baseline metrics followed by the use of NNETs. Performance metrics are also introduced to show the validity of model and include the mean squared error, residual error, and mean absolute percentage error. Classification referring to a model which generates a discrete output such as class 1 as function of missile design. Classification methods such as linear discriminant analysis and quadratic discriminant analysis are introduced as baseline metrics, again followed by the use of NNETs. Performance metrics for the classification models include accuracy, precision, recall, and confusion matrices. It will be shown that while the baseline metrics perform very well, NNETs must be used to gain higher performance at the cost of training time. All models introduced in this dissertation are built in Python [1] using Scikit-Learn [2] for lower order modeling and TensorFlow with Keras [3-4] for NNETs.

To address the issue of robustness and sensitivity, data will be assumed to be missing, since the data generated is in a controlled environment, meaning all the data needed is always available. However, in most real-life scenarios data can be missing for samples. It could have been lost, unavailable, or unreliable. Also, when inputting samples into the prediction model, programmatically a missing value cannot just be NAN/missing otherwise it will cause errors. Therefore, the missing value is replaced or imputed. Several methods exist for imputation models such as mean value imputation, which just replaces the missing value with the mean of the feature column. Other methods include using linear regression and NNET models to impute data. Missing data will be simulated for the classification database and models will be rebuilt to assess performance using the missing data.

Finally, model interpretation is a huge concern in data science. With more traditional models like linear regression, the coefficients are used to evaluate which parameters have more weight in the model. A positive coefficient would show that the parameter increases the output of the model, and a negative coefficient would show that the parameters decrease the output of the model. With linear regression this is easier to do since coefficients can be obtained for each term including the higher degree polynomials and even though there may be a large number of coefficients, the model is still interpretable. However, when using NNETs, the model becomes a black box because there are nonlinear activations of units, and it is challenging to understand how both the weights and inputs affect the model. The other issue with NNETs is that multiple networks with different parameters including weights can give the same result. So, to interpret NNETs the SHapley Additive exPlanation (SHAP) values can be used to determine how the inputs affect the model prediction. SHAP values are used like coefficients in a linear model to show how important a parameter is globally but can also be used to gain local input feature importance measures.

II. Physics Modeling

Both the AUSRC and AULRC contain a suite of subroutines that design and simulate a missile launch. Each program contains subroutines which calculate the mass properties, aerodynamics, propulsion, and 6-DOF flight with or without guidance/autopilot. If the reader is at the Aerospace Department, the reader may obtain the physical copy of Anderson's dissertation which does a thorough description of the AUSRC subroutines [5]. The reader may also reference the MSIC reports (Hartfield [6-17]) and notes [18] which outline a majority of the tools and rocket propulsion design. For more specifics on liquid propulsion, the readers should refer to [19-26]. For more specifics on solid propulsion, the readers should refer to [27-36]. A brief discussion on the main subroutines and only certain derivations will be shown.

II.I AULRC: Propulsion Subroutine

The propulsion subroutine is the first main subroutine in the AULRC. There are various liquid rocket engine cycles such as pressure fed, gas generator, and full-flow combustion [23]. The engine cycle used in this work is a constant pressure fed system which uses a high-pressure tank to force fuel and oxidizer into the thrust chamber. The AULRC currently contains 29 different fuel and oxidizer combinations and allow for a variable equivalence ratio. Since equivalence ratio can be varied, the specific impulse (I_{sp}) is varied using polynomial fit (I_{sp}) equations. Table 1 shows the fuel and oxidizer combinations available in the AULRC and shows which fuels are available with each oxidizer denoted by a checkmark (✓). For example, RP-1 is available with IRFNA. Each combination also supplies the ratio of specific heats, chamber temperature, molecular weight, characteristic velocity, optimum oxidizer to fuel ratio, oxidizer density, and fuel density. First, to calculate the thrust, the nozzle pressure ratio must be determined using the input nozzle expansion ratio and ratio of specific heats as shown in Eq. (2.1), which has to be solved numerically and can then be used to solve for exit pressure since the chamber pressure (p_o) is an input. The thrust (F_T) in Eq. (2.2) can then be determined and the reader should note that the formulation includes (λ) which is a nozzle divergence correction factor and is determined using interpolation based on the expansion ratio and fractional nozzle length. Also, the thrust is corrected during the 6-DOF simulation to account for altitude varying the atmospheric pressure using Eq. (2.3).

Table 1 AULRC Fuel & Oxidizer Options

Oxidizer / Fuel	IRFNA	H_2O_2-95%	N_2O_4	ClF_3	BrF_5	LOX	LF_2
UDMH	✓	✓	✓	✓		✓	
Hydrazine	✓	✓	✓	✓	✓	✓	✓
Hydyne	✓	✓	✓	✓		✓	
RP-1	✓	✓					
JP-X	✓						
MMH	✓		✓				
Kerosene			✓	✓		✓	
Ammonia						✓	✓
LH_2						✓	✓

From Eq. (2.1), the nozzle pressure ratio is a function of nozzle expansion ratio and the fuel and oxidizer specific heat ratio. Equation (2.2) is a function of nozzle divergence correction factor, propellant thrust coefficient ($C_{T,\ Prop}$), gravity constant (g_e), propellant specific impulse $\left(I_{sp,prop}\right)$, chamber pressure (p_o), propellant thrust coefficient, nozzle throat area (A_t), nozzle exit area (A_e), fuel/oxidizer specific heat ratio (γ), nozzle exit pressure (p_e), an assumed nozzle propellant exit/chamber pressure ratio $\left(p_{e,prop}/p_{o,prop} = 0.0147\right)$, and characteristic exhaust velocity (C^*) of the propellant.

$$\frac{A_e}{A_t} = \frac{\sqrt{\gamma}\left(\frac{2}{\gamma+1}\right)^{\frac{\gamma+1}{2(\gamma-1)}}}{\left(\frac{p_e}{p_0}\right)^{\frac{1}{\gamma}}\sqrt{\frac{2\gamma}{\gamma-1}}\left[1-\left(\frac{p_e}{p_o}\right)^{\frac{\gamma-1}{\gamma}}\right]^{\frac{1}{2}}} \tag{2.1}$$

$$F_T = \lambda \frac{p_o A_t C_{T,prop}}{\left[1-\left(\frac{p_{e,prop}}{p_{o,prop}}\right)^{\frac{\gamma-1}{\gamma}}\right]^{\frac{1}{2}}}\left[1-\left(\frac{p_e}{p_o}\right)^{\frac{\gamma-1}{\gamma}}\right]^{\frac{1}{2}} + A_e p_e - A_e p\left(altitude\right) \tag{2.2}$$

$$F_T = \lambda \frac{p_o A_t\left(\frac{g_e I_{sp,prop}}{C^*}\right)}{\left[1-\left(0.0147\right)^{\frac{\gamma-1}{\gamma}}\right]^{\frac{1}{2}}}\left[1-\left(\frac{p_e}{p_o}\right)^{\frac{\gamma-1}{\gamma}}\right]^{\frac{1}{2}} + A_e p_e - A_e p\left(altitude\right) \tag{2.3}$$

Equation (2.4) shows the nozzle divergence correction factor which is a function of the conical nozzle half angle. The correction factor is used since one-dimensional isentropic flow does not account for nozzle viscous losses and losses due to two/three-dimensional geometry flow. Since the AULRC and AUSRC both use bell nozzles, Eq. (2.4) does not accurately define the correction factor needed, but the correct bell nozzle performance parameters are shown in Appendix A: Bell Nozzle Correction Factor.

$$\lambda \equiv \frac{1+\cos\alpha}{2} \tag{2.4}$$

Thrust in this subroutine is simulated with no varying atmospheric pressure and is saved to a database which can be accessed during the 6-DOF and will then include varying atmospheric pressure. Since the thrust is developed beforehand, the mass flow through the nozzle can be calculated from Eq. (2.5), which is the thrust ($F_{T,SL}$) and includes the atmospheric pressure at sea level. The mass flow through the nozzle can then be determined from Eq. (2.6), which is a function of the nozzle throat diameter, chamber pressure, and characteristic exhaust velocity. The mass flow will be used in the mass subroutine to determine the mass of fuel and oxidizer components. The characteristic exhaust velocity is supplied for each propellant combination.

$$F_{T,SL} = \lambda \frac{p_o A_t \left(\frac{g_e I_{sp,prop}}{C^*} \right)}{\left[1 - (0.0147)^{\frac{\gamma-1}{\gamma}} \right]^{\frac{1}{2}}} \left[1 - \left(\frac{p_e}{p_o} \right)^{\frac{\gamma-1}{\gamma}} \right]^{\frac{1}{2}} + A_e p_e - A_e p(Sea-Level) \tag{2.5}$$

$$\dot{m} = \frac{F_{T,SL}}{C_T C^*} = \frac{p_o A^* C_T}{C_T C^*} = \frac{p_o A^*}{C^*} \tag{2.6}$$

II.II AULRC: Mass Subroutine

The mass subroutine is the second main subroutine and develops various properties such mass, length, volume, centers of gravity, moments, and inertia. Inertia cross products are all zero since the missile geometry is considered axisymmetric. Full derivations of each component are shown from Burkhalter [6-13], and many of the derivations were taken from Sutton [23], Huzel and Haung [24], and Humble [25]. The only parameters derived here are for the fuel, oxidizer, and compressed gas so the reader should refer to Burkhalter [6]. Readers should also refer to [19-27] for various liquid propulsion derivations and discussions. Masses are derived from densities of components input by the user such as aluminum or steel. Densities can be changed per component so the tail fins can be made of a different material from the body casing. Mass properties such as density for the fuel and oxidizer are programmed and only need to be selected by the user in the input. The equations to determine the fuel and oxidizer are shown in the subsequent equations, since they are determined after the thrust has been developed. Following the fuel and oxidizer sizing, the properties of the pressurization system are developed. Equation (2.6) is used to determine the mass flow through the nozzle, which is then used to determine propellant mass shown in Eq. (2.7) and is mass flow times the burn time (t_b). Equation (2.7) can be used since the chamber pressure is kept constant.

$$m_{prop} = \dot{m} t_b \tag{2.7}$$

The fuel mass (m_{fuel}) can then be determined from Eq. (2.8), which relates the propellant mass to the optimal oxidizer to fuel mass ratio (r_{opt}). The oxidizer mass (m_{ox}) can finally be determined by subtracting fuel mass from propellant mass using Eq. (2.9).

$$m_{fuel} = \frac{m_{prop}}{r_{opt}+1} = \frac{m_{prop}}{\left(\frac{m_{ox}}{m_{fuel}}\right)_{opt}+1} \tag{2.8}$$

$$m_{ox} = m_{prop} - m_{fuel} \tag{2.9}$$

The mass of the fuel $(M_{Fuel\,Tank})$ and oxidizer tanks $(M_{Ox\,Tank})$ can be determined empirically from Eq.'s (2.10) and (2.11).

$$M_{\text{Fuel Tank}} = 0.6952\left(\frac{m_{fuel}}{2.2}\right)^{0.6} \tag{2.10}$$

$$M_{\text{Ox Tank}} = 0.6952\left(\frac{m_{ox}}{2.2}\right)^{0.6} \tag{2.11}$$

The theoretical volumes of the fuel and oxidizer can both be determined from Eq. (2.12) which is the oxidizer or fuel mass divided by oxidizer or fuel density. Initially, the oxidizer and fuel tanks are assumed to be spherical, and the radius of the tank (r_t) is calculated from Eq. (2.12) . If the tank radius is determined to be greater than the body radius, the spherical tank will not fit and will need to be resized to a configuration which includes hemispherical endcaps and a cylindrical midsection.

$$V_{Theoretical,Sphere} = \frac{m}{\rho} \tag{2.12}$$

$$r_t = \left(\frac{3V}{4\pi}\right)^{1/3} \tag{2.13}$$

Since there are two hemispherical endcaps, the volume is just a sphere using the body inside radius (r_{body}) , shown on Eq. (2.14). The volume of the cylindrical section is calculated by subtracting the endcaps volume from the theoretical sphere volume developed from Eq. (2.12), shown in Eq. (2.15). Equation (2.16) shows the length of the spherical endcaps is just twice the inside body radius and Eq. (2.17) shows the cylindrical length is the cylindrical volume divided by circular area. Therefore, the total tank length is the cylindrical length plus the length of the endcaps, shown in Eq. (2.18).

$$V_{Endcaps} = \frac{4}{3}\pi r_{body}^3 \tag{2.14}$$

$$V_{Cyl} = V_{Theoretical,Sphere} - V_{Endcaps} \tag{2.15}$$

$$L_{Endcaps} = 2r_{body} \tag{2.16}$$

$$L_{Cyl} = \frac{V_{Cyl}}{\pi r_{body}^2} \tag{2.17}$$

$$L_T = L_{Cyl} + L_{Endcaps} \tag{2.18}$$

Next, the amount of gas required to pressurize both the fuel and oxidizer tanks must be determined. The mass of compressed gas (M_{CG}) can be calculated individually for fuel and oxidizer using Eq.'s (2.19) and (2.20). The fuel/oxidizer pressure (p), fuel/oxidizer volume (V), compressed gas specific heat ratio(γ_{CG}), gas constant for compressed gas (R_{CG}), compressed gas temperature (T_{CG}), compressed gas pressure (p_{CG}), and compressed gas pressure regulator ($\Delta \mathrm{p}_{CG}$) are required to calculate the compressed gas for both the fuel and oxidizer. The total compressed gas is then just a sum of the compressed gas of the fuel and oxidizer using Eq. (2.21). The geometry and sizing of the compressed gas tank is done using Eq.'s (2.12)–(2.18). Again, if the radius of theoretical sphere is too large for the body, then the tank must be resized using a cylindrical body with hemispherical endcaps.

$$M_{CG,fuel} = \frac{p_{fuel} V_{fuel} \gamma_{CG}}{R_{CG} T_{CG} \left(1 - \frac{p_{fuel} - \Delta p_{CG}}{p_{CG}}\right)} \tag{2.19}$$

$$M_{CG,ox} = \frac{p_{ox} V_{ox} \gamma_{CG}}{R_{CG} T_{CG} \left(1 - \frac{p_{ox} - \Delta p_{CG}}{p_{CG}}\right)} \tag{2.20}$$

$$M_{CG} = M_{CG,fuel} + M_{CG,ox} \tag{2.21}$$

II.III AUSRC: Mass Subroutine

The AUSRC is programmed slightly different from the AULRC in that the mass properties are defined before calculating the thrust properties. Mass properties are calculated for most components in an actual missile which include wing/tail fins, body case, nose and warhead, actuators, and nozzle. The main difference in the AUSRC is obvious, the propellants are solid grain configurations. Right circular perforated solid propellant motors can be constructed using multiple types of configurations which include cylindrical, star, short/long spoke wagon wheels, dog bone, spherical, tubular, slotted, dendrite, and many others. Currently, the AUSRC can develop cylindrical, star, and short/long spoke wagon wheels using specific parameters. The tool does not allow users to directly declare the grain to be a star or wagon wheel, so users must understand how some of the geometry parameters can develop the specific grain type they desire. The MSIC reports and Rocket Propulsion course notes [6-18], Barrere [27], and Hartfield [28-36] discuss the derivations with the most detail and will help readers understand the solid motor design the best.

The first analytically programmable grain type is the star grain, shown on Fig. 1. There are multiple ways to define the geometry, but the program uses propellant inner radius (R_I), propellant outer radius (R_P), star/spoke angular fraction (ε), number of star/spoke points (N), and the star/spoke angular opening (θ) to obtain required parameters such as burn perimeter (S) and burn area (A_b) which is used to calculate thrust (F_T). It is noted that since the user can input the star angular opening, which gives star half angle($\theta/2$), the thrusting first phase can be progressive, neutral, or negative as shown in Eq. (2.22) which is a coefficient calculated for the phase 1 burn perimeter.

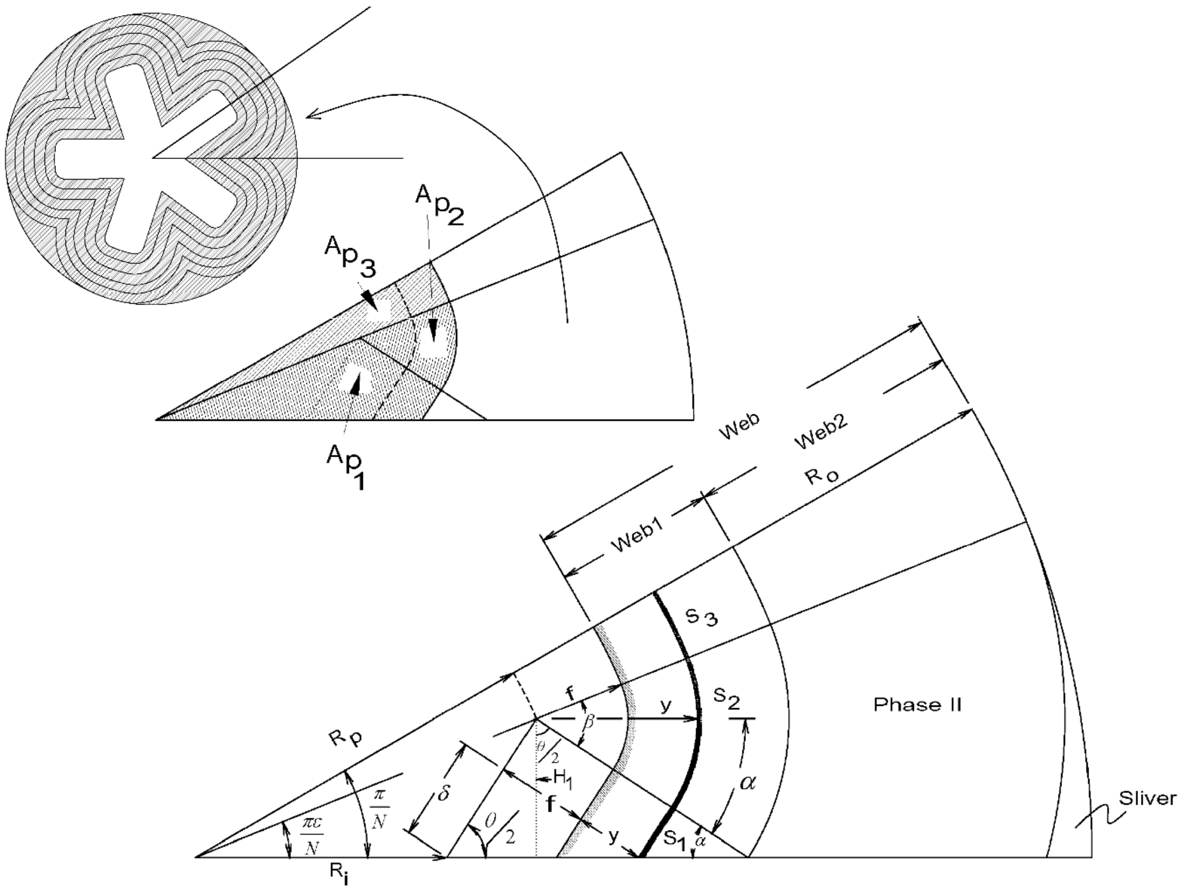


Fig. 1 Star Grain Geometry & Burn Area.

$$\frac{\pi}{2}-\frac{\theta}{2}+\frac{\pi}{N}-\cot\left(\frac{\theta}{2}\right)\rightarrow\begin{cases}>0 & \text{Progressive}\\ =0 & \text{Neutral}\\ <0 & \text{Regressive}\end{cases} \tag{2.22}$$

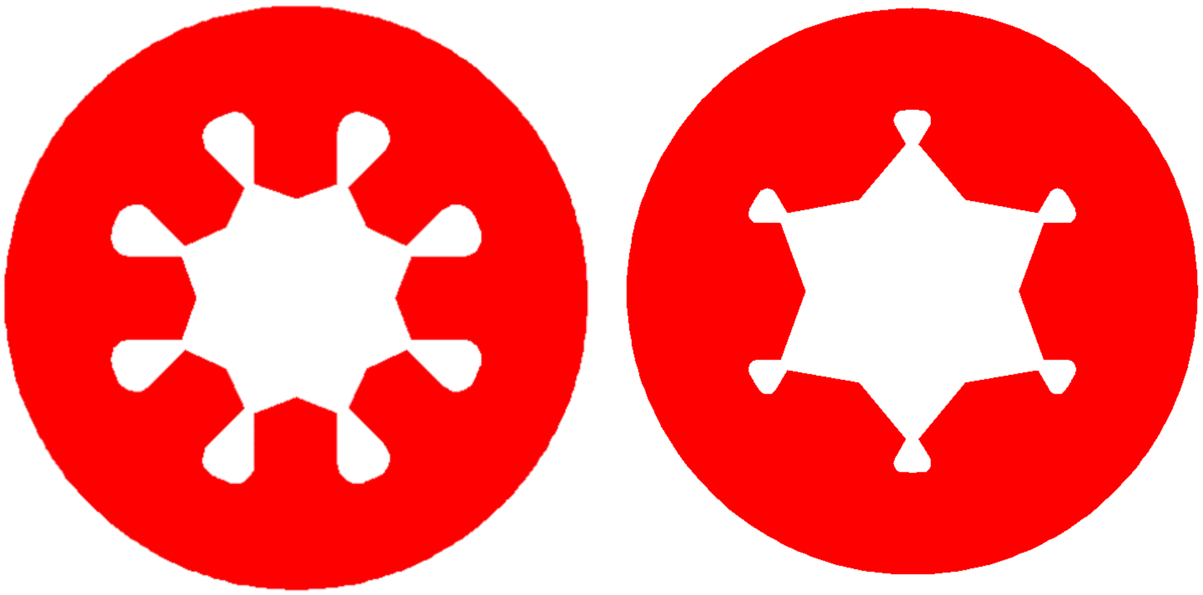


Fig. 2 Long Spoke (Left) & Short Spoke (Right) Wagon Wheels.

Wagon wheels are a special case of star configurations, so the program first determines whether the inputs make a star or wagon wheel configuration. Figure 2 shows the long and short spoke wagon wheels programmable in the AUSRC. Again, the propellant inner radius (R_I), propellant outer radius (R_P), star/spoke angular fraction (E), number of star/spoke points (N), and the star/spoke angular opening (θ) are used to develop the wagon wheels. However, there are two types of wagon wheels, long or short spoke, and the program will first determine if the grain geometry parameters develop a short spoke, if not then it is a star grain.

The long and short spokes have identically the same geometry as shown on Fig. 3, except their spokes are considered long or short. Their phase 1 burns use the same calculations; however, they deviate in their phase 2 burn since they have different web thicknesses. Similar to the star grain analysis, the perimeter calculation can be used to calculate a coefficient which determines the phase 1 thrust profile. Similar to Eq. (2.22), Eq. (2.23) will determine if the phase 1 thrust is progressive, neutral, or regressive.

$$\frac{\pi}{N}+\frac{\pi}{2}-\frac{2}{\sin\left(\frac{\theta}{2}\right)}+\frac{1}{\tan\left(\frac{\theta}{2}\right)}\rightarrow\begin{cases}>0 & \text{Progressive}\\ =0 & \text{Neutral}\\ <0 & \text{Regressive}\end{cases} \tag{2.23}$$

Another calculation is done to check and see if the grain is a long spoke or short spoke. First, at a minimum, to be a short spoke, the side spoke length (*h*) must be greater than zero and is shown in Eq. (2.24). If (*h*) is less than zero, then the grain configuration is over constrained and must be reverted to a star configuration. If (*h*) is positive then it is a spoke configuration, but further calculation must be done to determine if it is a long spoke or short spoke. From the derivations of long spokes in Barrere [27], for the long spoke to be valid it must follow the constraint defined by Eq. (2.25). Physically, Eq. (2.25) represents the radial length for a long spoke and if it is negative, it means the spokes are over lapping and the geometry must be reverted to a short spoke, shown on **Error! Reference source not found.**.

$$h=\left[R_p\cos\left(\frac{\pi\varepsilon}{N}\right)-\frac{R_p\sin\left(\frac{\pi\varepsilon}{N}\right)}{\tan\left(\frac{\theta}{2}\right)}-R_I\right]\sin\left(\frac{\theta}{2}\right)>0 \tag{2.24}$$

$$R_p\cos\left(\frac{\pi\varepsilon}{N}\right)-R_I-\frac{R_p\sin\left(\frac{\pi\varepsilon}{N}\right)}{\sin\left(\frac{\theta}{2}\right)}\geq 0 \tag{2.25}$$

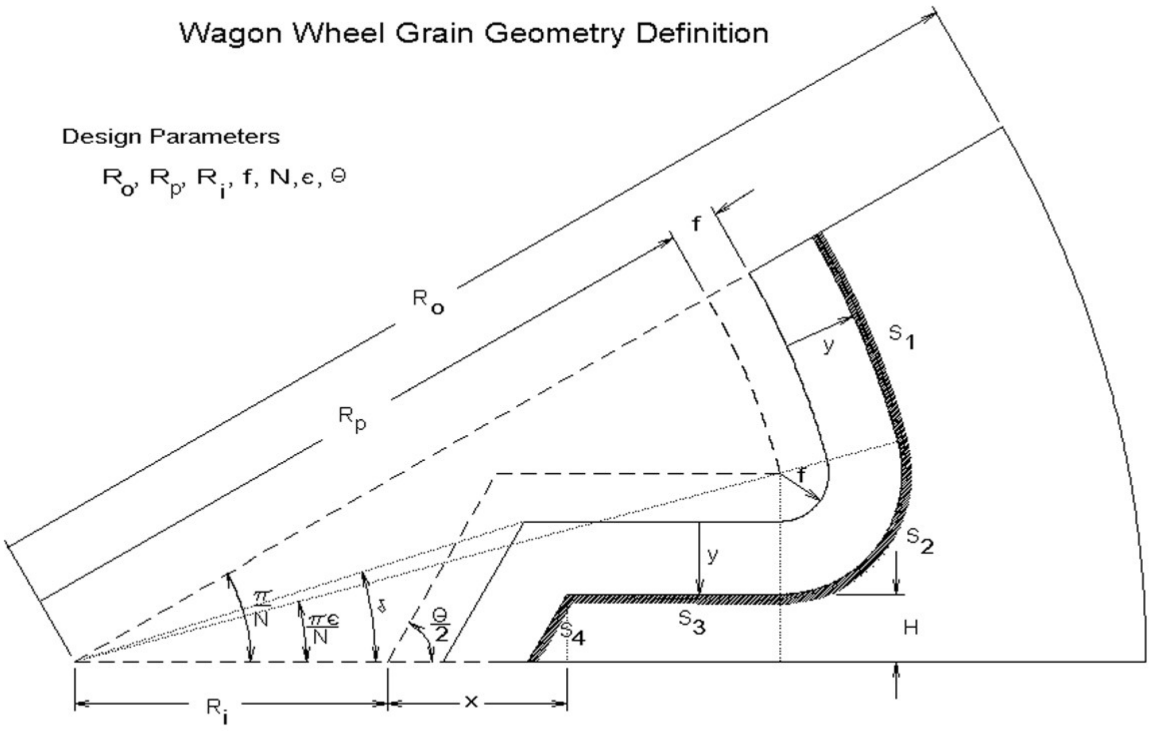


Fig. 3 Wagon Wheel Geometry.

Another design consideration that can get lost in the programming is the calculation of the propellant grain length (GL). GL is not a direct input by the user and is calculated after certain components have been designed. The starting point of the grain (x_{grain}) is determined right after the warhead section. Next, the nozzle design is determined by using the expansion ratio and fractional nozzle length to determine the nozzle throat circular arc based on bell nozzles. First, the nozzle length based on a 15-degree cone is determined by the empirical equation shown on Eq. (2.26). The diverging bell nozzle length (L_{Noz}) can then be calculated from the fractional nozzle length (L_F) and 15 degree conical nozzle (L_{Cone}) in Eq. (2.27). The total converging diverging nozzle length (L_{Noz}) is then the sum of the converging nozzle entrance (L_E), which is shown in Appendix B:, and diverging bell nozzle length.

$$L_{Cone} = \frac{r_e - r_{Cone}}{\tan\left(15^\circ\right)} = \frac{r_e - r_t - \left(\sigma = 0.382\right) r_t \left[1 - \cos\left(15^\circ\right)\right]}{\tan\left(15^\circ\right)} \tag{2.26}$$

$$L_N = L_F L_{Cone} \tag{2.27}$$

$$L_{Noz} = L_E + L_N \tag{2.28}$$

The grain length *GL* can then be calculated by subtracting the starting point of the grain $\left(x_{grain}\right)$, the gap between the grain, nozzle entrance location $\left(0.06R_{Body}\right)$, and (L_{Noz}) from the missile body length (L_B), shown on Eq. (2.29).

$$GL = L_B - x_{grain} - L_{Noz} - 0.06R_{Body} \tag{2.29}$$

II.IV AUSRC: Propulsion Subroutine

Once the mass properties are determined the propulsion subroutine then calculates chamber pressure over time, which then gets used to calculate thrust over time. For the complete derivation, refer to [6] in Chapter 3. In general, the first required calculation is chamber pressure (p_o), shown on Eq. (2.30), which is a function of propellant burn area (A_b), nozzle throat area(A_t), propellant burn rate constant (a_b), propellant density (ρ_b), propellant characteristic exhaust velocity (C^*), and the propellant burn rate index (n).

$$p_o = \left[\frac{A_b}{A_t} a_b \rho_b C^*\right]^{\frac{1}{1-n}} \tag{2.30}$$

Equation (2.1) is then used to calculate the nozzle exit pressure (p_e). Similar to Eq. (2.2), the thrust can then be determined by Eq. (2.31) and includes how thrust is corrected during flight to account for the change atmospheric pressure. Again, the reader should note that the bell nozzle correction factor can be obtained by using the empirical formulation from Appendix A: Bell Nozzle Correction Factor and interpolate for varying values of nozzle expansion ratio.

$$F_T = \lambda p_o A_t \sqrt{\frac{2\gamma^2}{\gamma - 1}\left[\frac{2}{\gamma + 1}\right]^{\frac{\gamma+1}{\gamma-1}}\left[1 - \left(\frac{p_e}{p_o}\right)^{\frac{\gamma-1}{\gamma}}\right]} + A_e\left[p_e - p\left(atmosphere\right)\right] \qquad (2.31)$$

II.V Aerodynamics Subroutine

The aerodynamics subroutine uses a modified version of AERODSN [37][38].[INCLUDE NACA 1307] The original AERODSN did not include drag calculations but was updated by Burkhalter to include drag calculations. For a given geometry, AERODSN will develop aerodynamic loads and moments over a set of Mach numbers and set of angles of attack. The missile is assumed to be symmetric, so the yawing moments are determined from the pitching moment coefficients and the side forces are determined from the normal force coefficients. The aerodynamic database is then saved to a matrix and can be accessed later during the 6-DOF simulation. AERODSN is very similar to Missile Datcom [39-41] and use many of the same methods to approximate aerodynamic interactions. Readers should also refer to [42-44] for more in depth discussions on similar missile aerodynamic prediction methods. Nielsen [45] provides many of the aerodynamic derivations and assumptions that many programs attempt to numerically solve such as AERODSN.

II.VI 6-DOF Subroutine

The 6-DOF subroutine is the main bulk of the AULRC & AUSRC. This algorithm uses the equations of motion to simulate the trajectory motion of a missile. The equations of motion that are used in the 6-DOF subroutine, can be taken from Etkin [46][47] or Schmidt [48]. There are several major assumptions, which greatly influence the equations and are the following: the missile structure is rigid; there are rotating masses such as rotor; and there are no cross products of inertia. The cross products of inertia are all zero since the missile is symmetric, which is shown on Eq. (2.32), and the principal axis runs through the missile, shown on Eq. (2.33). Therefore, we are left with only the *moments of inertia*.

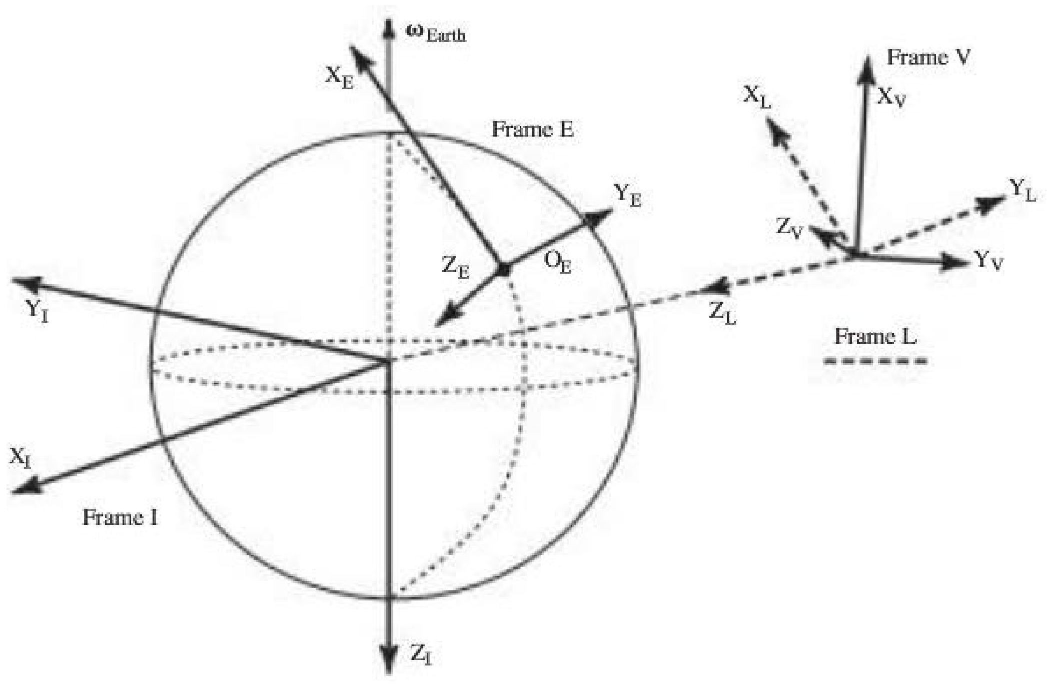


Fig. 4 Frames for Unsteady Motion [48].

$$I_{XY} = I_{YX} = I_{ZY} = I_{YZ} = 0 \tag{2.32}$$

$$I_{XZ} = I_{ZX} = 0 \tag{2.33}$$

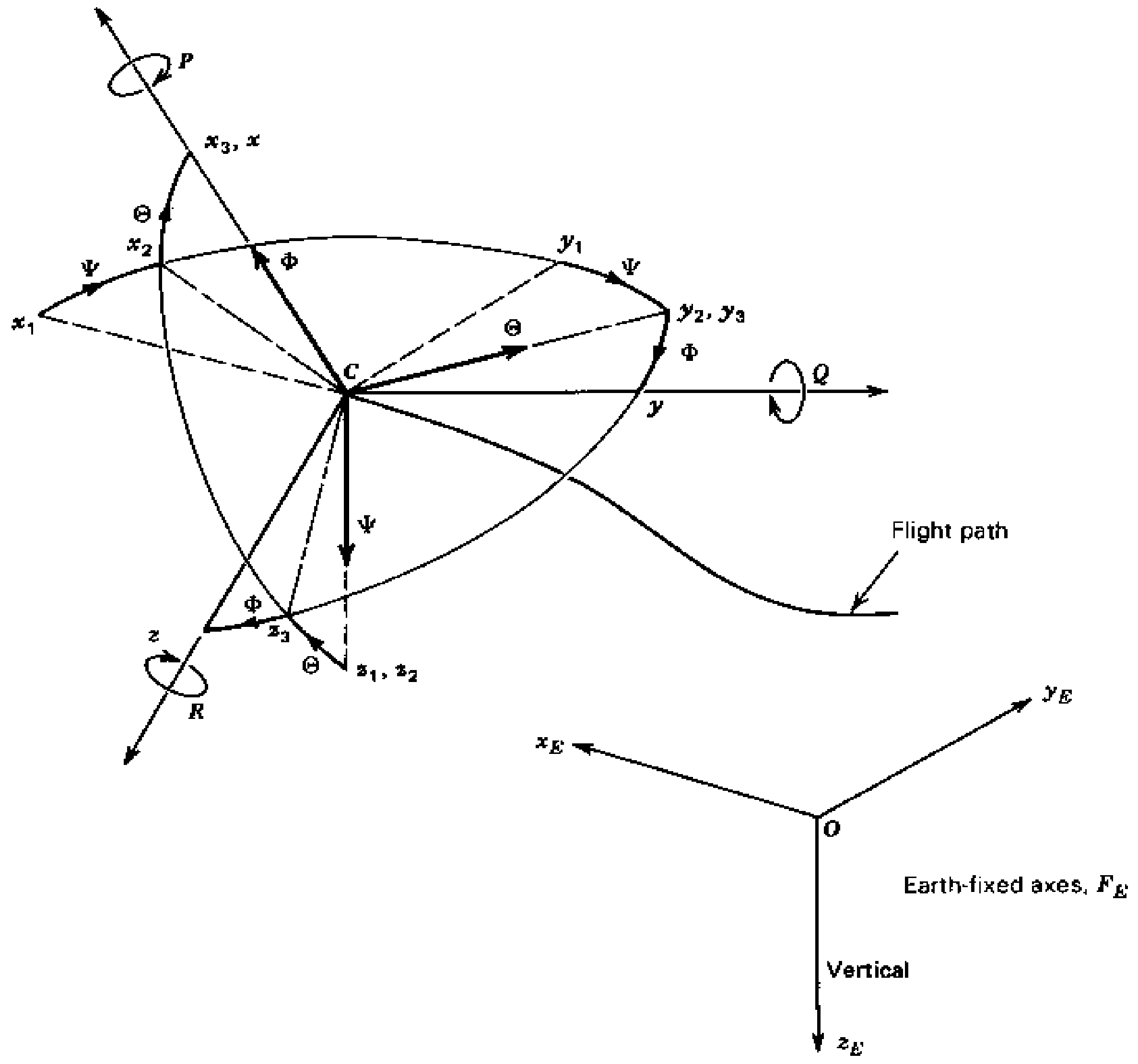


Fig. 5 Body Orientation [46].

We then use Eulerian definitions of ϕ, θ, and Ψ, which are all in degrees. ϕ is the missile bank angle which is a measure of the missile roll angle during flight. θ is the missile pitch angle and is initially the launch angle relative to the launch point. Ψ is the missile heading angle and shows what direction the missile is headed and so if it was 90 degrees initially the missile would be headed East. Because missiles often launch vertically, there is a sine of 90 degrees that gets divided in the equations of motion causing an infinite value to appear, which is known as "gimbal-lock" when $\theta = 90$ degrees. To avoid gimbal locking, the Euler angles are converted to quaternions before integration. Defining the quaternions on Eq. (2.34) shows how the three-dimensional Eulerian angles are converted to a 4-dimensional vector and can then convert the Eulerian angles to quaternions using Eq. (2.35).

$$\begin{aligned} & e_0 + e_1\hat{i} + e_2\hat{j} + e_3\hat{k} = \left(e_0, e_1, e_2, e_3\right) \\ & e_0^2 + e_1^2 + e_2^2 + e_3^2 = 1 \end{aligned} \tag{2.34}$$

$$\begin{aligned} e_0 &= \cos\left(\frac{\psi}{2}\right)\cos\left(\frac{\theta}{2}\right)\cos\left(\frac{\phi}{2}\right) + \sin\left(\frac{\psi}{2}\right)\sin\left(\frac{\theta}{2}\right)\sin\left(\frac{\phi}{2}\right) \\ e_1 &= -\cos\left(\frac{\psi}{2}\right)\cos\left(\frac{\theta}{2}\right)\sin\left(\frac{\phi}{2}\right) + \sin\left(\frac{\psi}{2}\right)\sin\left(\frac{\theta}{2}\right)\cos\left(\frac{\phi}{2}\right) \\ e_2 &= -\cos\left(\frac{\psi}{2}\right)\sin\left(\frac{\theta}{2}\right)\cos\left(\frac{\phi}{2}\right) - \sin\left(\frac{\psi}{2}\right)\cos\left(\frac{\theta}{2}\right)\sin\left(\frac{\phi}{2}\right) \\ e_3 &= \cos\left(\frac{\psi}{2}\right)\sin\left(\frac{\theta}{2}\right)\sin\left(\frac{\phi}{2}\right) - \sin\left(\frac{\psi}{2}\right)\cos\left(\frac{\theta}{2}\right)\cos\left(\frac{\phi}{2}\right) \end{aligned} \tag{2.35}$$

If we define the direction cosine matrix "3-2-1" in which we rotate about Ψ, then θ, and finally ϕ, we get the vehicle rotational frame *V* with respect to frame *L*, from Fig. 5, and the matrix shown on Eq. (2.36) defines the conversion of frame *V* to frame *L*. The *L* frame is the "local-vertical-local-horizontal: frame and is vehicle carried rather than vehicle fixed, so it does not rotate with the vehicle. The position of frame *L* is the vehicle's center of mass. $T_{L\text{-}V}$ is then the transformation of frame *V* to frame *L* and is denoted as "tbod" in the program, shown on Eq. (2.37) , however is in terms of Eulerian angles. Note that $T_{L\text{-}V}$ is a **Body 1-2-3** sequence. If converted to quaternions then we obtain Eq. (2.38). Since $T_{L\text{-}V}$ is orthogonal, the inverse is simply the transpose of $T_{L\text{-}V}$, which is shown on Eq. (2.39), and allows for the calculation of $T_{V\text{-}L}$ and is denoted as "b2i" in the program. It is noted that the rotation matrix and quaternion transformation is setup such that $-sin\theta = 2(e_0e_2 + e_1e_3)$, which gets used in this sequence version, and is due to the use of the Hamiltonian version of quaternions [49]. See Cooke [50] and Amoruso [51] for more information on conversion of Euler angles to quaternions and adapting quaternions to the equations of motion.

$$\begin{Bmatrix} i_V \\ j_V \\ k_V \end{Bmatrix} = T_{L-V}(\phi,\theta,\psi)\begin{Bmatrix} i_L \\ j_L \\ k_L \end{Bmatrix} \tag{2.36}$$

$$T_{L-V}(\phi,\theta,\psi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & \sin\phi \\ 0 & -\sin\phi & \cos\phi \end{bmatrix}\begin{bmatrix} \cos\theta & 0 & -\sin\theta \\ 0 & 1 & 0 \\ \sin\theta & 0 & \cos\theta \end{bmatrix}\begin{bmatrix} \cos\psi & \sin\psi & 0 \\ -\sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix}$$
$$T_{L-V} = \begin{bmatrix} \cos\theta\cos\psi & \cos\theta\sin\psi & -\sin\theta \\ \sin\phi\sin\theta\cos\psi - \cos\phi\sin\psi & \sin\phi\sin\theta\sin\psi + \cos\phi\cos\psi & \sin\phi\cos\theta \\ \cos\phi\sin\theta\cos\psi + \sin\phi\sin\psi & \cos\phi\sin\theta\sin\psi - \sin\phi\cos\psi & \cos\phi\cos\theta \end{bmatrix} \tag{2.37}$$

$$T_{L-V} = \begin{bmatrix} e_0^2 + e_1^2 - e_2^2 - e_3^2 & 2(e_1e_2 - e_0e_3) & 2(e_0e_2 + e_1e_3) \\ 2(e_0e_3 + e_1e_2) & e_0^2 - e_1^2 + e_2^2 - e_3^2 & 2(e_2e_3 - e_0e_1) \\ 2(e_1e_3 - e_0e_2) & 2(e_0e_1 + e_2e_3) & e_0^2 - e_1^2 - e_2^2 + e_3^2 \end{bmatrix} \tag{2.38}$$

$$T_{V-L} = T_{L-V}^{-1} = T_{L-V}^{T} = transpose(T_{L-V}) \tag{2.39}$$

We can now use Eq.'s (2.38) and (2.39) to solve the equations of motion. For example, Eq. (2.40) shows how to obtain the velocity of the missile with respect to the Earth frame *E* using the missile velocities *u, v, w.*

$$\begin{bmatrix} V_X \\ V_Y \\ V_Z \end{bmatrix} = T_{V-L} \begin{bmatrix} u \\ v \\ w \end{bmatrix} \tag{2.40}$$

Next, we can then define the rotational velocity of the Earth using Eq. (2.41) where (p_E, q_E, r_E) are the roll rate, pitch rate, and yaw rate, respectively, and subscript *E* refers to "respect with earth frame." (ω_{Earth}) is the Earth's rotational velocity, and λ_L is latitude in degrees.

$$\begin{bmatrix} p_E \\ q_E \\ r_E \end{bmatrix} = T_{L-V} \begin{bmatrix} \omega_{Earth} \cos \lambda_L \\ 0 \\ -\omega_{Earth} \cos \lambda_L \end{bmatrix} \tag{2.41}$$

We can then determine the accelerations (du, dv, dw) of the missile with respect to Earth using Eq. (2.42), which is Eq. 2.123 in Schmidt [48] and is written the same way in the Auburn tools. These are the scalar equations of motion governing the translational velocity of the vehicle body relative to the Earth Frame *E*. It is noted that (Q) is the dynamic pressure $\left(\frac{1}{2}\rho V^2\right)$, (S_{Ref}) is the reference area (ft^2), (F_T) is thrust (lbf), (g) is gravitational acceleration (ft/sec^2), (m) is the missile mass (lbm), and (c_X, c_Y, c_Z) are the aerodynamic component force coefficients on the missile.

$$\begin{aligned} du &= F_T \frac{g}{m} + c_X Q S_{\text{Ref}} \frac{g}{m} + g T_{L-V}[1,3] - (q - q_E) w + (r + r_E) v \\ dv &= c_Y Q S_{\text{Ref}} \frac{g}{m} + g T_{L-V}[2,3] + (p - p_E) w - (r + r_E) u \\ dw &= c_Z Q S_{\text{Ref}} \frac{g}{m} + g T_{L-V}[3,3] - (p - p_E) v + (q + q_E) u \end{aligned} \tag{2.42}$$

Next, the scalar equations of rotational motion are shown on Eq. (2.43) including the *cross products of inertia*. These equations govern the inertial rotational velocity and allow one to solve the rates of *p, q,* and *r*. Using Eq.'s (2.32) and (2.33), Eq. (2.43) can be greatly reduced to Eq. (2.44).

$$\begin{aligned}
&I_{XX}\dot{p} - I_{XZ}\left(\dot{r} + pq\right) - I_{YZ}\left(q^2 - r^2\right) - I_{XY}\left(\dot{q} - rp\right) + \left(I_{ZZ} - I_{YY}\right)rq = \sum L \\
&I_{YY}\dot{q} + \left(I_{XX} - I_{ZZ}\right)pr - I_{XY}\left(\dot{p} + qr\right) - I_{YZ}\left(\dot{r} - pq\right) + I_{XZ}\left(p^2 - r^2\right) = \sum M \\
&I_{ZZ}\dot{r} - I_{XZ}\left(\dot{p} - qr\right) - I_{XY}\left(p^2 - q^2\right) - I_{YZ}\left(\dot{q} + rp\right) + \left(I_{YY} - I_{XX}\right)pq = \sum N
\end{aligned} \tag{2.43}$$

$$\begin{aligned}
&I_{XX}\dot{p} + \left(I_{ZZ} - I_{YY}\right)rq = \sum L \\
&I_{YY}\dot{q} + \left(I_{XX} - I_{ZZ}\right)pr = \sum M \\
&I_{ZZ}\dot{r} + \left(I_{YY} - I_{XX}\right)pq = \sum N
\end{aligned} \tag{2.44}$$

Equation (2.44) can then be rearranged to solve for rates of inertial rotational velocity or the inertial rotational acceleration, shown on Eq. (2.45). (c_L, c_M, c_N) are the lift, moment, and normal coefficient, respectively, and L_{Ref} is the reference length (ft).

$$\begin{aligned}
\dot{p} &= \frac{c_L Q S_{\text{Ref}} L_{\text{Ref}} - \left(I_{ZZ} - I_{YY}\right)qr}{I_{XX}} \\
\dot{q} &= \frac{c_M Q S_{\text{Ref}} L_{\text{Ref}} - \left(I_{XX} - I_{ZZ}\right)pr}{I_{YY}} \\
\dot{r} &= \frac{c_N Q S_{\text{Ref}} L_{\text{Ref}} - \left(I_{YY} - I_{XX}\right)pq}{I_{ZZ}}
\end{aligned} \tag{2.45}$$

Next, we calculate the angular velocity of the vehicle of Frame *V* with respect to Frame *L* and is shown on Eq. (2.46). The angular velocity of frame *L* with respect to frame *I* is similarly defined from Eq. (2.47) and can then obtain the angular velocity of the vehicle, Frame *V*, with respect to frame *I*, shown on Eq. (2.48).

$$\omega_{V,L} \triangleq \begin{bmatrix} p_V & q_V & r_V \end{bmatrix} \begin{Bmatrix} i_V \\ j_V \\ k_V \end{Bmatrix} \tag{2.46}$$

$$\omega_{L,I} \triangleq \begin{bmatrix} p_L & q_L & r_L \end{bmatrix} \begin{Bmatrix} i_V \\ j_V \\ k_V \end{Bmatrix} \tag{2.47}$$

$$\omega_{V,I} = \omega_{V,L} + \omega_{L,I} \triangleq \begin{bmatrix} p & q & r \end{bmatrix} \begin{Bmatrix} i_V \\ j_V \\ k_V \end{Bmatrix} \tag{2.48}$$

Because $[p, q, r]$ and $[p_I, q_I, r_I]$ are known, Eq. (2.48) can be rearranged to solve for $\omega_{V,I}$ or $[p_V, q_V, r_V]$, shown on Eq.'s (2.49) & (2.50), respectively. Equation (2.50) is written as "capp", "capq", and "capr" for $[p_V, q_V, r_V]$ in the Auburn tools. μ_L is the longitudinal position of the missile, so $\dot{\mu}_L$ is the longitudinal position rate of the missile, and $\dot{\lambda}_L$ is latitudinal position rate of the missile.

$$\omega_{V,L} = \omega_{V,I} - \omega_{L,I} \tag{2.49}$$

$$\begin{bmatrix} p_V & q_V & r_V \end{bmatrix}\begin{Bmatrix} i_V \\ j_V \\ k_V \end{Bmatrix} = \begin{bmatrix} p & q & r \end{bmatrix}\begin{Bmatrix} i_V \\ j_V \\ k_V \end{Bmatrix} - \begin{bmatrix} p_L & q_L & r_L \end{bmatrix}\begin{Bmatrix} i_V \\ j_V \\ k_V \end{Bmatrix}$$

$$\begin{bmatrix} p_V \\ q_V \\ r_V \end{bmatrix} = \begin{bmatrix} p \\ q \\ r \end{bmatrix} - T_{L-V}\begin{bmatrix} \left(\omega_{Earth} + \dot{\mu}_L\right)\cos\lambda_L \\ -\dot{\lambda}_L \\ -\left(\omega_{Earth} + \dot{\mu}_L\right)\sin\lambda_L \end{bmatrix} \tag{2.50}$$

Because the Eulerian angles are converted to quaternions, the rates of Eulerian angles must be determined in quaternion form and is shown in Eq. (2.51). κ is a constant equal to 0.1.

$$\begin{aligned} \varepsilon &= 1 - \left(e_0^2 + e_1^2 + e_2^2 + e_3^2\right) \\ \dot{e}_0 &= \kappa\varepsilon e_0 + \frac{1}{2}\left(pe_1 + qe_2 + re_3\right) \\ \dot{e}_1 &= \kappa\varepsilon e_1 + \frac{1}{2}\left(-pe_0 + qe_2 - re_3\right) \\ \dot{e}_2 &= \kappa\varepsilon e_2 + \frac{1}{2}\left(-qe_0 - re_1 + pe_3\right) \\ \dot{e}_3 &= \kappa\varepsilon e_3 + \frac{1}{2}\left(-re_0 + qe_1 - pe_2\right) \end{aligned} \tag{2.51}$$

Finally, the vehicle's position relative to the surface of the Earth can be calculated using Eq. (2.52). Equation (2.52) is rearranged so that we can explicitly obtain the rates of change of longitude, latitude, and altitude, shown on Eq. (2.53) and is written using vector (*i, j*) notation to show components that are multiplied to the velocity terms. Again, readers should note in the Auburn tool, T_{V-L} is typed as "b2i" and $R_E + h$ is typed as "radi".

$$\begin{bmatrix} \dot{\lambda}_L\left(R_E+h\right) \\ \dot{\mu}_L\left(R_E+h\right)\cos\lambda_L \\ -\dot{h} \end{bmatrix} = T_{L-V}^T \begin{bmatrix} u \\ v \\ w \end{bmatrix} = T_{V-L} \begin{bmatrix} u \\ v \\ w \end{bmatrix} \tag{2.52}$$

$$\begin{aligned} \dot{\lambda}_L &= \frac{\left[T_{V-L}[1,1]*u+T_{V-L}[1,2]*v+T_{V-L}[1,3]*w\right]}{R_E+h} \\ \dot{\mu}_L &= \frac{\left[T_{V-L}[2,1]*u+T_{V-L}[2,2]*v+T_{V-L}[2,3]*w\right]}{\left(R_E+h\right)\cos\lambda_L} \\ \dot{h} &= -\left[T_{V-L}[3,1]*u+T_{V-L}[3,2]*v+T_{V-L}[3,3]*w\right] \end{aligned} \tag{2.53}$$

Equations (2.32) through (2.53) are the main portions of the 6-DOF that get calculated and used to track the missile. A 7th-8th order Runge-Kutta (RK7(8)) method is used to numerically integrate the equations of motion [52]. The RK7(8) is the same algorithm in both the AUSRC and AULRC. This algorithm is able to do variable time stepping to improve speed by halving the time step until the relative error is reduced below a tolerance. Similar to a 4th order Runge-Kutta, the RK7(8) method essentially calls the function of interest (typed as “deq” in the Auburn tool), i.e., Eq.’s (2.32)-(2.53), and records the inputs and outputs. The RK7(8) function is basically a wrapper around Eq.’s (2.32)-(2.53), it takes an input vector of 19 parameters and calculates 19 outputs. Table 2 shows the inputs to the “RK78” function in the Auburn tools and includes the description as well as its typed name in the Auburn tool.

Table 2 RK7(8) Inputs and Outputs

Input	Input Description (Program Name)	Output	Output Description (Program Name)
u	Velocity along X_V (u)	du	Acceleration along X_V (du)
v	Velocity along Y_V (v)	dv	Acceleration along Y_V (dv)
w	Velocity along Z_V (w)	dw	Acceleration along Z_V (dw)
p	Roll Rate, rad/sec (p)	dp	Roll Acceleration, rad/sec^2 (dp)
q	Pitch Rate, rad/sec (q)	dq	Pitch Acceleration, rad/sec^2 (dq)
r	Yaw Rate, rad/sec (r)	dr	Yaw Acceleration, rad/sec^2 (dr)
e_0	Quaternion (Scalar) (e0)	$\dot{e}_0$	Quaternion Rate (de0)
e_1	Quaternion (i) (e1)	$\dot{e}_1$	Quaternion Rate (de1)
e_2	Quaternion (j) (e2)	$\dot{e}_2$	Quaternion Rate (de2)
e_3	Quaternion (k) (e3)	$\dot{e}_3$	Quaternion Rate (de3)
λ_L	Latitude, degrees (xlamda)	$\dot{\lambda}_L$	Latitude Rate, deg/sec (dlamda)
μ_L	Longitude, degrees (ymu)	$\dot{\mu}_L$	Longitude Rate, deg/sec (dmu)
$R_E + h$	Altitude, ft (radi)	$(R_E + \dot{h})$	Altitude Rate, ft/sec (dradi)
p_G	Pitch Gyro Rate, rad/sec (pgyro)	$\dot{p}_G$	Pitch Gyro Accel., rad/sec^2 (pgryod)
y_G	Yaw Gyro Rate, rad/sec (ygro)	$\dot{y}_G$	Yaw Gyro Accel., rad/sec^2 (ygryod)
$\dot{\delta}_e$	Elevator Pitch Rate, Deg/sec (deledot)	$\ddot{\delta}_e$	Elevator Pitch Accel., deg/sec^2 (deleddot)
δ_e	Elevator Pitch, Degrees (dele)	$\dot{\delta}_e$	Elevator Pitch Rate, Deg/sec (deledot)
$\dot{\delta}_r$	Rudder Yaw Rate, Deg/sec (delrdot)	$\ddot{\delta}_r$	Rudder Yaw Accel., deg/sec^2 (delrddot)
δ_r	Rudder Yaw, Degrees (delr)	$\dot{\delta}_r$	Rudder Yaw Rate, Deg/sec (delrdot)

From the RK7(8) described in Fehlberg [52] "Part III. Seventh-Order Formula – Section XI.", shows how to setup the differential equations for a seventh order Runge-Kutta with step size control. RK7(8) is seventh order but is eighth order for the equations of condition.

$$
\begin{aligned}
&f_0 = f\left(x_0, y_0\right) \\
&f_k = f\left(x_0 + \alpha_k h, y_0 + h\sum_{\lambda=0}^{k-1} \beta_{k\lambda} f_\lambda\right) \quad \left(k = 1, 2, \ldots, 12\right) \\
&y = y_0 + h\sum_{k=0}^{10} c_k f_k + O\left(h^8\right) \\
&\hat{y} = y_0 + h\sum_{k=0}^{12} \hat{c}_k f_k + O\left(h^9\right)
\end{aligned}
\tag{2.54}
$$

Equation (2.54) describes the equation necessary for integration and we can see that there are 13 total calls to a function, i.e., Eq.'s (2.32)-(2.53) ("deq"). Fehlberg then describes the process for determining α_k, $\beta_{k\lambda}$, c_k, and $\hat{c}_k$, which are constants already programmed in the Auburn tools and described on "Table X. RK7(8)" [52]. The parameters in Eq. (2.54) are for example α_k are typed as "ALPH(1)", $\beta_{k\lambda}$ is typed as "B2_1" which is $2/27$ (the 2 in B2_1 means the second row and the 1 means the first column), c_k, & $\hat{c}_k$ are typed as "CH(1)". The $\hat{y}$ is the actual answer of the integration and y is used as an error estimation.

II.VII Guidance & Autopilot Subroutine

Most of the uses with the AULRC and AUSRC are to simulate ballistic launches. Ballistic launches are flyouts where there is no wing or fin or thrust defection influencing the equations of motion. Fin deflections influence the equations of motion through pitch/yaw gyro rates and elevator/rudder rates, which are shown on Table 2. Fins can be deflected by the use of two elevators which cause pitching and two rudders which cause yaw, the elevators and rudders are also known as the tail fins. The program does not currently allow forward wing (canard) deflection. There are other methods of deflection through the use of thrust vectoring [10]. Thrust can be vectored through the use of nozzle vanes and gimbaling. Currently, the AUSRC can use all three models of deflection and the AULRC can use nozzle vanes and fin deflection. During the flight, if there is no fin or thrust deflection then the aerodynamic and force coefficients are updated based on the missile's current angle of attack. If the fins are allowed to be deflected then the fin aerodynamic and force coefficients are updated and added to the total aerodynamic and force coefficients.

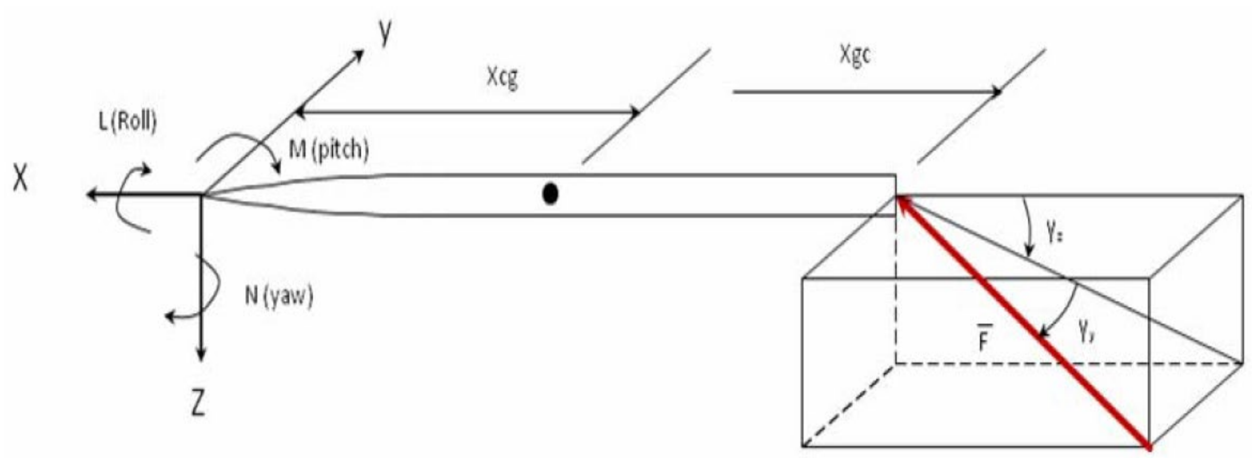


Fig. 6 Schematic of Gimbal Forces and Moments, Stability Coordinate System.

Equation (2.42) is then updated with an additional thrust to *dv* and *dw*. There are two different methodologies for determining thrust in *X, Y, Z* directions for vanes and gimbaling. If gimbaling is used then the thrust vector is augmented by the gimbaling angles γ_y and γ_z and the corrected thrust vector can be obtained from Fig. 7 and is shown on Eq. (2.56). F_T is the thrust developed during the propulsion subroutine. Because the thrust vector is off set from the center of gravity, it causes a moment and is added to right side of Eq. (2.44) but only pitching and yawing is considered.

$$\begin{aligned} du &= F_{T,X}\frac{g}{m} + c_X Q S_{\text{Ref}}\frac{g}{m} + gT_{L-V}[1,3] - (q - q_E)w + (r + r_E)v \\ dv &= F_{T,Y}\frac{g}{m} + c_Y Q S_{\text{Ref}}\frac{g}{m} + gT_{L-V}[2,3] + (p - p_E)w - (r + r_E)u \\ dw &= F_{T,Z}\frac{g}{m} + c_Z Q S_{\text{Ref}}\frac{g}{m} + gT_{L-V}[3,3] - (p - p_E)v + (q + q_E)u \end{aligned} \tag{2.55}$$

$$\vec{F} = F_T \cos\gamma_Y \cos\gamma_Z \hat{i} + F_T \sin\gamma_Y \hat{j} + F_T \sin\gamma_Y \hat{k}$$
$$\vec{F} = F_{T,X}\hat{i} + F_{T,Y}\hat{j} + F_{T,Z}\hat{k} \tag{2.56}$$

Equation (2.45) is then updated to include gimbaling thrust, shown on Eq. (2.57), and is also the same for vane deflection. X_{gc} and X_{cg} are the distance to the vane/gimbaling center and the distance to center of gravity measured from the nose.

$$\dot{p} = \frac{c_L Q S_{\text{Ref}} L_{\text{Ref}} - (I_{ZZ} - I_{YY}) qr}{I_{XX}}$$
$$\dot{q} = \frac{F_{T,Z}\left(X_{gc} - X_{cg}\right) + c_M Q S_{\text{Ref}} L_{\text{Ref}} - (I_{XX} - I_{ZZ}) pr}{I_{YY}} \tag{2.57}$$
$$\dot{r} = \frac{F_{T,Y}\left(X_{gc} - X_{cg}\right) + c_N Q S_{\text{Ref}} L_{\text{Ref}} - (I_{YY} - I_{XX}) pq}{I_{ZZ}}$$

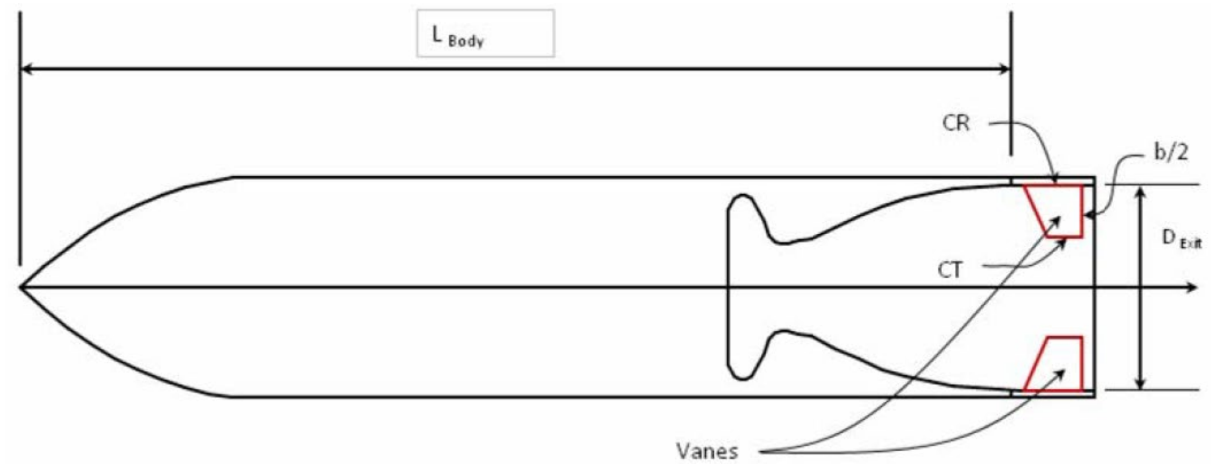


Fig. 7 Nozzle Vanes Geometry.

The nozzle vanes have a different approach to thrust vectoring. Similar to the fins, there are two sets of vanes, which cause pitching or yawing. Thrusting is affected by the sole existence of vanes in the direction of thrust, as shown on Fig. 8. The vanes are set in the exhaust plume, shown on Fig. 9, and the exhaust plume add aerodynamic forces, which cause pitching and yawing. To determine the forces on vanes, AERODSN is used approximate the aerodynamic forces and the resulting thrust vector is shown on Eq. (2.58).

$$\begin{aligned}\vec{F} &= F_{T,\text{Vane}}\hat{i} + c_{Y,\text{Vane}}Q_{\text{Exit}}S_{\text{Ref}}\hat{j} - c_{N,\text{Vane}}Q_{\text{Exit}}S_{\text{Ref}}\hat{k} \\ \vec{F} &= F_{T,X}\hat{i} + F_{T,Y}\hat{j} + F_{T,Z}\hat{k}\end{aligned} \tag{2.58}$$

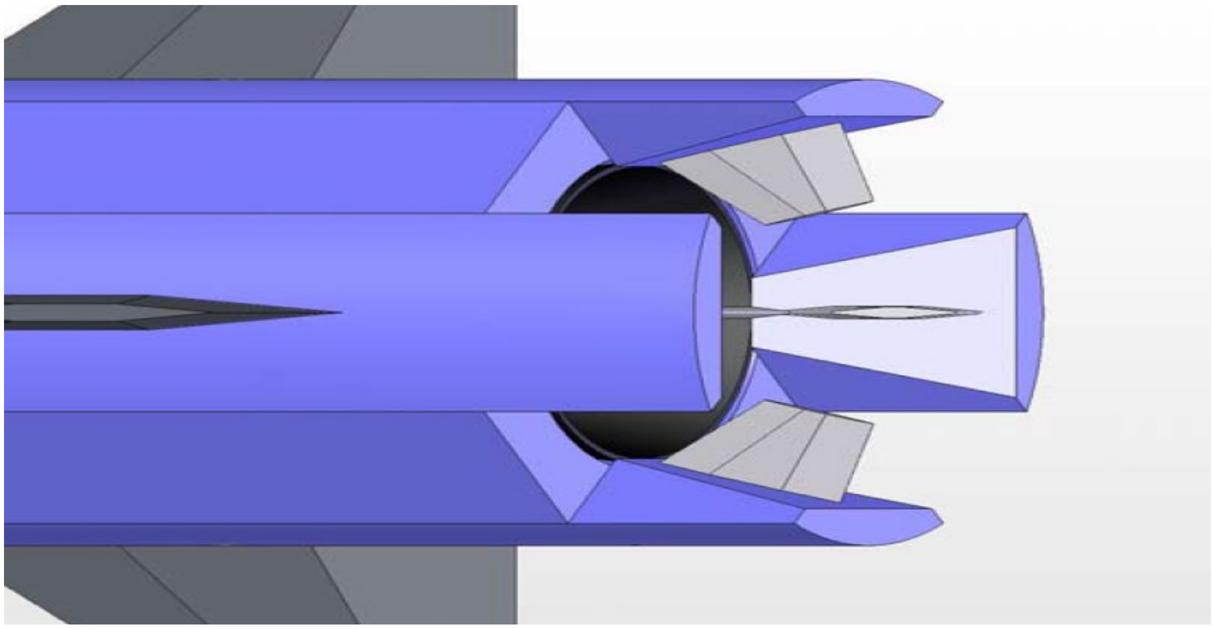


Fig. 8 Vane Geometry in Nozzle Exhaust Plume.

Notice that the $\hat{\imath}$ component includes $F_{T,Vane}$, which is not just the thrust generated during the thrust subroutine. Because the vanes are in the exhaust plume, they decrement the thrust by some amount, and unless CFD was used then we cannot directly calculate the thrust decremented. So, an approximation was made based on the vane profile area A_P and nozzle exit area A_E; therefore, the effective thrust is shown on Eq. (2.59). Realize that if there are no vanes, then, the thrust is not decremented by the vane profile area. Equations (2.55) and (2.57) are the same for nozzle vanes.

$$F_{T,Vane} = F_T \left(1 - \frac{A_P}{A_E} \right) \tag{2.59}$$

$$\begin{aligned}
A_P &= A_{P1} + A_{P2} \\
A_{P1} &= (\text{\#-Vanes})(\text{Vane Semi-Span})(\text{Vane Thickness}) \\
A_{P1} &= 4\frac{b_V}{2}t_k \\
A_{P2} &= (\text{\#-Deflected Vanes})(\text{Vane Planform Area}) \\
A_{P2} &= 2\frac{b_V}{2}\frac{C_R + C_T}{2}(\sin\gamma_Z + \sin\gamma_Y)
\end{aligned} \tag{2.60}$$

Next, we will briefly describe the overall guidance and autopilot algorithm. To develop the autopilot commands, a guidance law is used to develop the acceleration commands. This work has utilized the proportional navigation (PRONAV) guidance law to develop the acceleration commands required for pitch and yaw [54]. Because we assume there is no missile rolling, there is no third component of commanded acceleration; however, this algorithm still follows a three-dimensional setup. The PRONAV guidance law is shown on Eq. (2.61) where n_C is the acceleration command (ft/s^2), N' is the effective navigation ratio, V_C is the target closing velocity (ft/s), and $\dot{\lambda}$ is the line of sight rate (rad/s).

$$n_C = N' V_C \dot{\lambda} \tag{2.61}$$

We refrain from discussing other guidance laws here, but the user should refer to Zarchan [54], Yanushevsky [55], Blakelock [56], Siouris [57], and Gibeau [58] for more guidance and autopilot laws. This work will follow the derivation similar to Moran [59] and shows how to breakdown the three-dimensional PRONAV algorithm using two-dimensional solutions. First, we define $(R_{TMX}, R_{TMY}, R_{TMZ})$, which is the difference between the target position and the missile's position, shown on Eq. (2.62).

$$\begin{aligned} R_{TMX} &= X_T - X_M \\ R_{TMY} &= Y_T - Y_M \\ R_{TMZ} &= Z_T - Z_M \end{aligned} \tag{2.62}$$

We can then define the line of sight angles (LOS) (λ), where λ_{XY} is the LOS on the XY plane and λ_{XZ} is the LOS on the XZ plane and are shown on Eq. (2.63).

$$\begin{aligned} \lambda_{XY} &= \tan^{-1}\left(\frac{R_{TMY}}{R_{TMX}}\right) \\ \lambda_{XZ} &= \tan^{-1}\left(\frac{R_{TMZ}}{R_{TMX}}\right) \end{aligned} \tag{2.63}$$

Similar to Eq. (2.62), the missile to target velocities $(V_{TMX}, V_{TMY}, V_{TMZ})$, which are the difference between the target's velocity and the missile's velocity are shown on Eq. (2.64).

$$\begin{aligned} V_{TMX} &= VX_T - VX_M \\ V_{TMY} &= VY_T - VY_M \\ V_{TMZ} &= VZ_T - VZ_M \end{aligned} \tag{2.64}$$

The LOS plane rates $(\dot{\lambda})$ can then be calculated using Eq.'s (2.62) & (2.64) and is shown on Eq. (2.65).

$$\begin{aligned} \dot{\lambda}_{XY} &= \frac{R_{TMX}V_{TMY} - R_{TMY}V_{TMX}}{R_{TMX}^2 + R_{TMY}^2} \\ \dot{\lambda}_{XZ} &= \frac{R_{TMX}V_{TMZ} - R_{TMZ}V_{TMX}}{R_{TMX}^2 + R_{TMZ}^2} \end{aligned} \tag{2.65}$$

Equations (2.62) & (2.64) are then used to compute the closing velocities of each plane (V_{CXY}, V_{CXZ}) and are shown on Eq. (2.66).

$$\begin{aligned} V_{CXY} &= -\ddot{R}_{TMXY} = -\frac{\left(R_{TMX}V_{TMX} + R_{TMY}V_{TMX}\right)}{\sqrt{R_{TMX}^2 + R_{TMY}^2}} \\ V_{CXZ} &= -\ddot{R}_{TMXZ} = -\frac{\left(R_{TMX}V_{TMX} - R_{TMZ}V_{TMZ}\right)}{\sqrt{R_{TMX}^2 + R_{TMZ}^2}} \end{aligned} \tag{2.66}$$

Using Eq. (2.66) we can now develop the acceleration commands for the XY and XZ plane and is shown on Eq. (2.67).

$$\begin{aligned} n_{CXY} &= N'V_{CXY}\dot{\lambda}_{XY} \\ n_{CXZ} &= N'V_{CXZ}\dot{\lambda}_{XZ} \end{aligned} \tag{2.67}$$

Because (n_{CXY}, n_{CXZ}) are acceleration magnitudes, we can finally develop the acceleration vector using the acceleration magnitudes and the LOS. This is shown on Eq. (2.68) and we can see that in the XY plane that there is no Z or $(\hat{k})$ component and we can see that in the XZ plane that there is no Y or $(\hat{j})$ component. We can convert the acceleration vector into the L frame using Eq. (2.69) and notice that $(\mathrm{N}_{CXY}, \mathrm{N}_{CXZ})$ is simply the dot product between (n_{CXY}, n_{CXZ}) and

$(\hat{T}_{L-V}[2,:], \hat{T}_{L-V}[3,:])$. $(\mathrm{N}_{CXY}, \mathrm{N}_{CXZ})$ are finally the required acceleration commands that get used in the autopilot algorithm.

$$\begin{aligned} \bar{n}_{CXY} &= -n_{CXY} \sin(\lambda_{XY}) \hat{i} + n_{CXY} \cos(\lambda_{XY}) \hat{j} + 0\hat{k} \\ \bar{n}_{CXZ} &= -n_{CXZ} \sin(\lambda_{XZ}) \hat{i} + 0\hat{j} + n_{CXZ} \cos(\lambda_{XZ}) \hat{k} \end{aligned} \tag{2.68}$$

$$\begin{aligned} N_{CXY} &= \bar{n}_{CXY} \cdot \bar{T}_{L-V}[2,:] \\ N_{CXY} &= -n_{CXY} \sin(\lambda_{XY}) T_{L-V}[2,1] + n_{CXY} \cos(\lambda_{XY}) T_{L-V}[2,2] \\ N_{CXZ} &= \bar{n}_{CXZ} \cdot \bar{T}_{L-V}[3,:] \\ N_{CXZ} &= -n_{CXZ} \sin(\lambda_{XZ}) T_{L-V}[3,1] + n_{CXZ} \cos(\lambda_{XZ}) T_{L-V}[3,3] \end{aligned} \tag{2.69}$$

Now, we define some of the autopilot fundamentals. If it is not clear yet, the commanded accelerations are used to update the missile trajectory path. How does the commanded accelerations cause the missile to follow a target? Well, by using the tail fins, nozzle vanes, or by gimbaling. So, what the autopilot is doing is determining the amount of actuation required. The autopilot used in this work follows the derivation from Nesline [60], which is an improvement over Zarchan [54], shows the autopilot derivation using aerodynamic derivatives. Anderson [61] provides baseline values and basic rule of thumbs that should be followed using the autopilot. The main parameters which define the autopilot are the damping ratio (ζ), time constant (τ), crossover frequency (ω_{CR}), actuator frequency (ω_{Act}), and actuator damping ratio (ζ_{Act}). We refrain from going the entire derivation of the autopilot gains, since they are programmed in the Auburn tools and are shown by Nesline [60]. We will however show the final derivation of the amount of elevator/rudder deflection (δ) (degrees) and the elevator/rudder acceleration ($\ddot{\delta}$). The missile actuators (δ) are modeled as a second-order transfer function shown on Eq. (2.70).

$$\frac{\delta}{\delta_C} = 1 \bigg/ \left(1 + \frac{2\zeta_{ACT} s}{\omega_{ACT}} + \frac{s^2}{\omega_{ACT}^2}\right) \tag{2.70}$$

Converting Eq. (2.70) from Laplace transform back to the time domain gives Eq. (2.71). The elevator/rudder acceleration can now be solved for by simply rearranging Eq. (2.71) into Eq. (2.72), where ω_{ACT} is actuator frequency set to $\left(125\frac{rad}{sec}\right)$ and ζ_{ACT} is the actuator damping ratio set to (0.7) in this work.

$$\delta + \frac{2\zeta_{ACT}}{\omega_{ACT}}\dot{\delta} + \frac{\ddot{\delta}}{\omega_{ACT}^2} = \delta_C \tag{2.71}$$

$$\ddot{\delta} = \omega_{ACT}^2\left(\delta_C - \delta - \frac{2\zeta_{ACT}\dot{\delta}}{\omega_{ACT}}\right) \tag{2.72}$$

The only term left to determine before using Eq. (2.72) is the fin deflection command (δ_C), which is the amount of fin deflection to turn the missile. The fin deflection command (δ_{EC}) for pitching is simply the sum of the missile pitch gyro rate (G_E) and missile pitch rate (q) times a rate gyro autopilot gain (K_R). The fin deflection command (δ_{YC}) for yawing is similarly the sum of the missile yaw gyro rate (G_Y) and missile yaw rate (r) times a rate gyro autopilot gain (K_R). The rate gyro autopilot is defined by Nesline [60] on equation (26) and is programmed in the Auburn tools, and therefore not shown here. It also is a function of multiple functions and would require more derivations not required for this work.

$$\begin{aligned} \delta_{EC} &= K_R\left(G_E + q\right) \\ \delta_{YC} &= -K_R\left(G_Y - r\right) \end{aligned} \tag{2.73}$$

II.VIII SCUD-B Input Data & AULRC Model Verification

Now that the overall physics have been discussed, we must define the input parameters for the AULRC and AUSRC. In general, most of the inputs are based on the SCUD-B and classes are augmented from the baseline SCUD-B class. For the regression case using AULRC, we take the SCUD-B baseline and simply add “noise” to the inputs to generate a large database of missiles. For the classification task using AUSRC, we take the SCUD-B baseline and augment specific parameters and every parameter has a slight amount of noise. These databases will be described in chapter III. The SCUD-B parameters are mostly taken from Seyfert [62], the “Rock Nail Report” [63], and the wiki page on the scud missile [64]. Table 3 shows the data for the SCUD-B pulled from the references [62-64].

Table 3 SCUD-B Data

Parameter	Parameter Value
Body Diameter, ft	2.90026
Propellant Type	4.0 (IRFNA/RP-1)
Equivalence Ratio (Ratio of Actual Fuel/Air to Stoichiometric Fuel/Air)	2.0
Chamber Pressure, Psi	1108.15
Nose Diameter to Body Diameter Ratio	1.000
Nose Length to Body Diameter Ratio	3.25
Nozzle Throat Diameter to Body Diameter Ratio	0.1408147
Nozzle Expansion Ratio (Nozzle Exit Area to Nozzle Throat Area)	10.32
Fractional Nozzle Length**	0.66
Burn Time, Seconds	62.00
Tail Root Chord to Body Diameter Ratio	1.50905
Tail Taper Ratio (Tip Chord to Root Chord Ratio)	0.42879
Tail Semi-Span to Body Diameter Ratio	0.51697
Tail Leading Angle, Degrees (Measured Normal to Missile Body)	60.00
Tail Trailing Edge X-Location to Body Length Ratio	1.000
Initial Launch Angle, Degrees**	90.00

**Values Approximated from [62-64].

The information below is related to SCUD-B.

1. Length: 11.2-11.4 meters depending on the warhead.
2. Diameter: 884 millimeters.
3. Span: 1.8 meters.
4. Launch Weight: 5562-5950 kg.
5. Empty Weight: 2076 kg.
6. Fuel Weight: 3771 kg (852 kg Fuel & 2919 kg Oxidizer)
7. Payload Weight: 1016 kg.
8. Range: 50-300 km
9. Speed: Maximum of 1.5 km/sec, 1.13 km/sec at Apogee, 1.4 km/sec at Impact.
10. Accuracy: 450 meters.

The SCUD-B is shown on Fig. 10 on a transport erector vehicle [64]. This SCUD-B missile can be modeled using the AULRC tool and is compared against the OMEGA [65] tool, which is a Government-Off-The-Shelf software used to generate liquid propellant missile 6-DOF simulations similar to the AULRC. To ensure our model is valid, we can plot the mass, thrust, range, and altitude over time for both the AULRC and OMEGA tools. Figure 11 shows how mass changes over time and we can see that the OMEGA & AULRC models agree very well. Notice that the once the propellant completely finishes, the mass is constant. There is a slight noticeable mismatch at the end of the propellant burning and could be due to slight errors in how mass properties were defined in the AULRC. Figure 12 shows the thrust over time for both tools and we can see that they agree. Notice the only difference is that OMEGA allows the chamber pressure to throttle up, whereas the AULRC starts at the prescribed chamber pressure. Figure 13 shows the range over time and we can see that both models agree. Finally, on Fig. 14, the altitude over time is plotted for both models and can see they match very well. Notice that OMEGA flies a bit higher compared to the AULRC, but both arrive at the same range since the AULRC will let the missile glide.

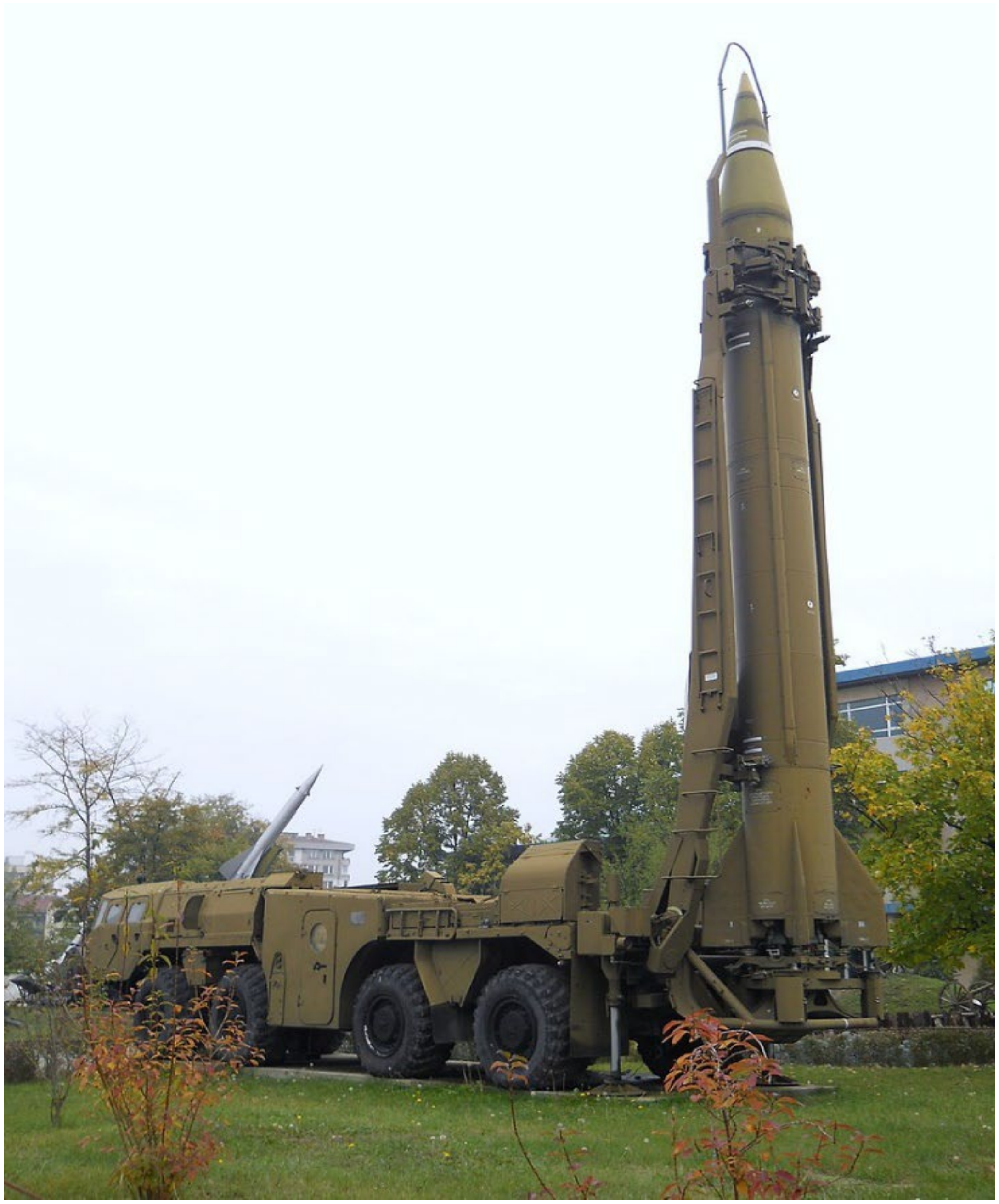


Fig. 9 SCUD-B on Transport Erector [64].

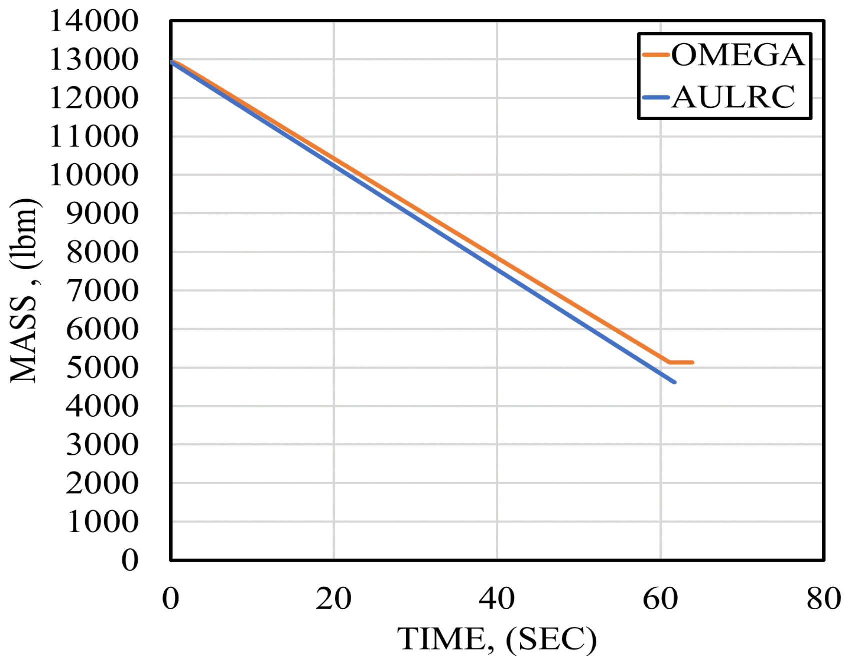


Fig. 10 Omega vs AULRC: Mass Over Time.

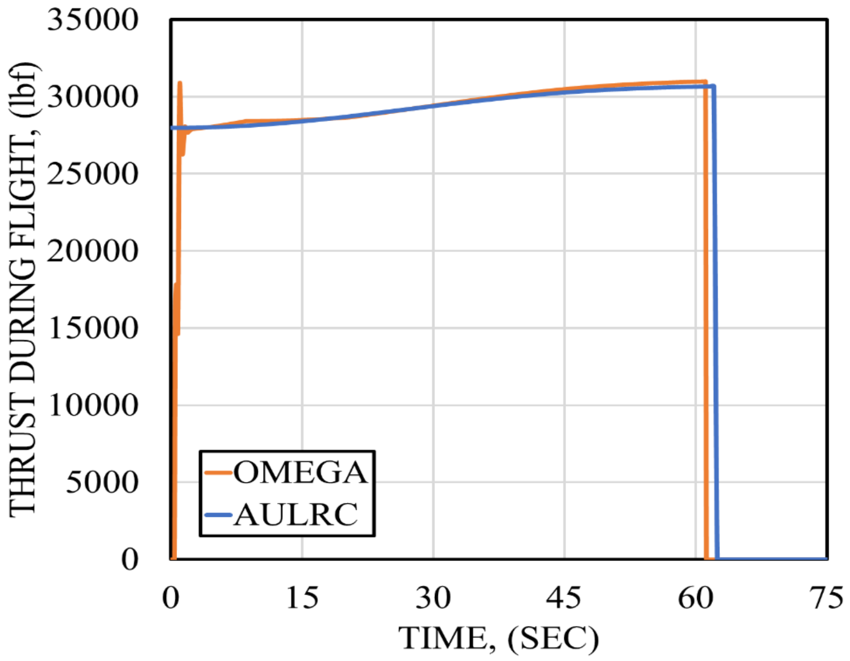


Fig. 11 Omega vs AULRC: Thrust Over Time.

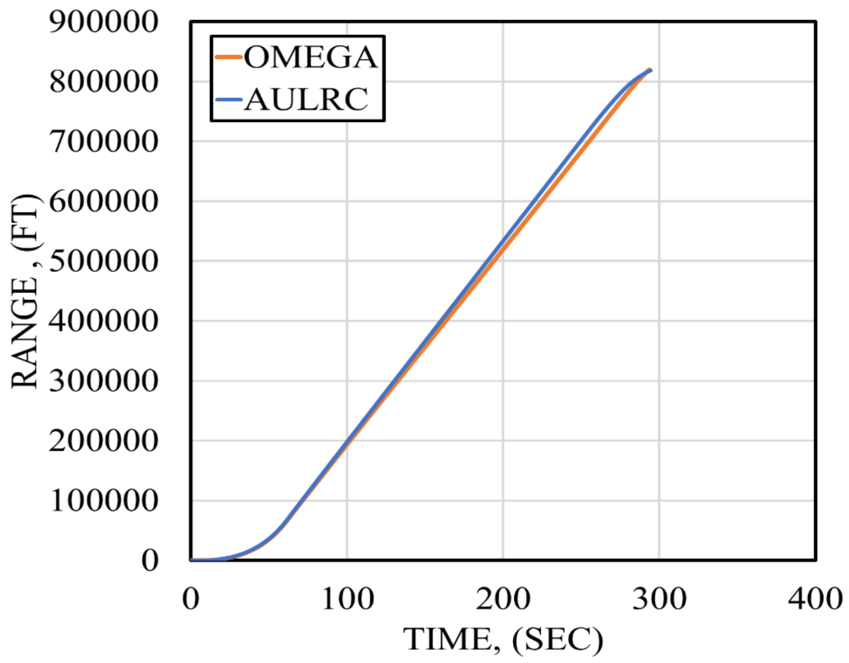


Fig. 12 Omega vs AULRC: Range Over Time.

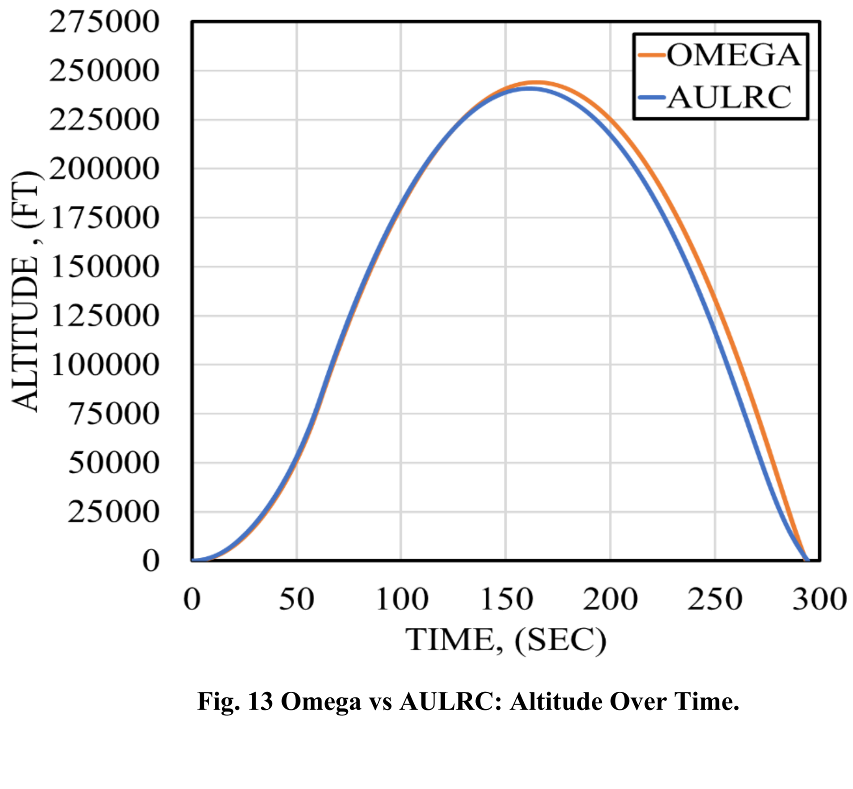


Fig. 13 Omega vs AULRC: Altitude Over Time.

II.IX AULRC Input Data

Table 4 AULRC SCUD-B Input

Parameter	Parameter Value
Body Diameter, ft	2.90026
Propellant Type	4.0 (IRFNA/RP-1)
Equivalence Ratio (Ratio of Actual Fuel/Air to Stoichiometric Fuel/Air)	2.0
Chamber Pressure, Psi	1108.15
Nose Diameter to Body Diameter Ratio	1.000
Nose Length to Body Diameter Ratio	3.25
Nozzle Throat Diameter to Body Diameter Ratio	0.1408147
Nozzle Expansion Ratio (Nozzle Exit Area to Nozzle Throat Area)	10.32
Fractional Nozzle Length**	0.66
Burn Time, Seconds	62.00
Wing Root Chord to Body Diameter Ratio	0.0
Wing Taper Ratio (Tip Chord to Root Chord Ratio)	0.0
Wing Semi-Span to Body Diameter Ratio	0.0
Wing Leading Angle, Degrees (Measured Normal to Missile Body)	0.0
Wing Leading Edge X-Location to Body Length Ratio	0.0
Tail Root Chord to Body Diameter Ratio	1.50905
Tail Taper Ratio (Tip Chord to Root Chord Ratio)	0.42879
Tail Semi-Span to Body Diameter Ratio	0.51697
Tail Leading Angle, Degrees (Measured Normal to Missile Body)	60.00
Tail Trailing Edge X-Location to Body Length Ratio	0.88
Autopilot Time Delay, Seconds***	231.80
Autopilot Time Constant (τ)***	0.4044
Autopilot Damping Coefficient (ζ)***	0.9140
Autopilot Crossover Frequency, (rad/sec)***	34.4028
Autopilot Effective Navigation Ratio***	3.741
Initial Launch Angle, Degrees**	86.241

**Values Approximated

***Values Generated from Genetic Algorithm

Using the information from Table 3, we can then generate a set of example inputs for the AULRC and is shown on Table 4. Notice the addition of the wing parameters but are set to zero since the SCUD-B has no wings. The tail trailing edge location has been moved up, because during simulations it appeared that missile was more stable when moving the tail closer to the nose. The initial launch angle is set to 86 degrees since the datasets that we focus on in this work do not use nozzle vanes to pitch so setting the launch angle at 86 degrees acts as a way to pitch through the use of gravity. Next, notice that the autopilot parameters are generated using the genetic algorithm (GA), which is available in both Auburn tools. The GA was set to match a range and apogee of 250 km and 75 km, respectively.

Figure 15 shows an example rocket using the input data from Table 4. Notice that the nose is blunt due to the nose diameter ratio equal to one. The SCUD-B does not actually have a blunt nose but was set to be blunt for this study. Figure 16 shows the nozzle that was produced using the input data on Table 4.

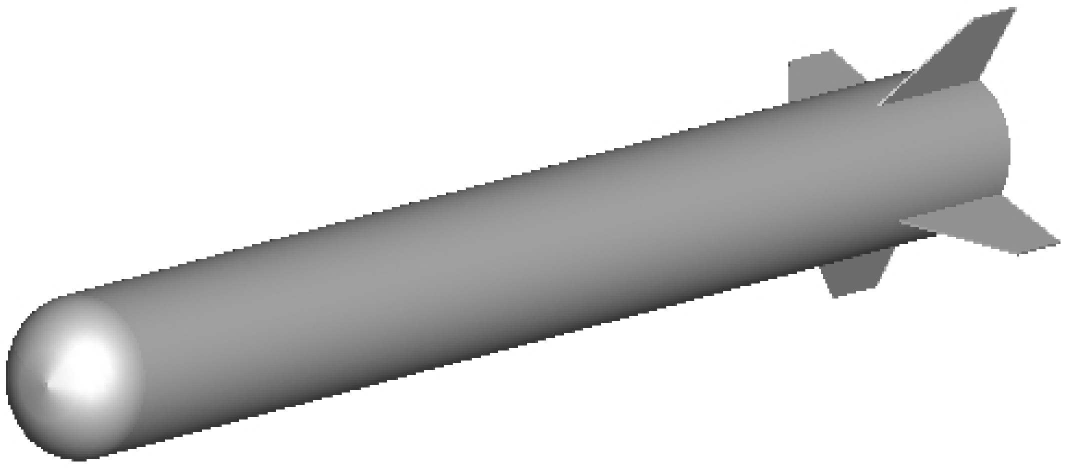


Fig. 14 AULRC Example Rocket.

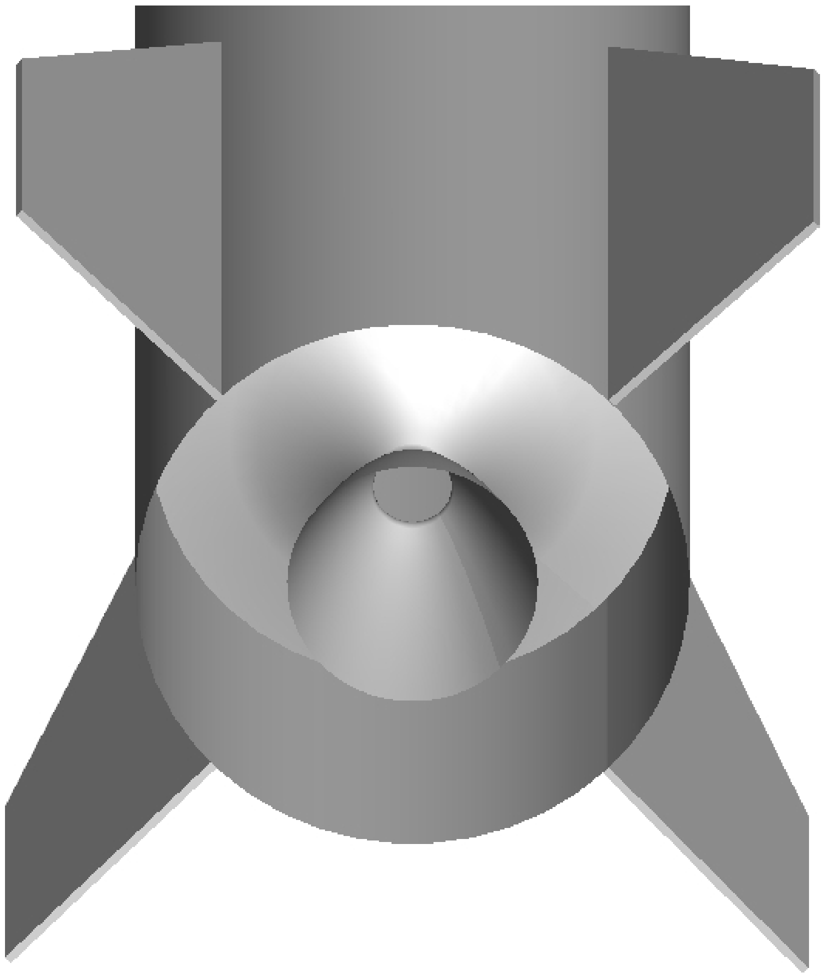


Fig. 15 Nozzle End for AULRC Example.

II.X AUSRC Input Data

Again, the information from Table 3 is also used to generate a baseline set of input for the AUSRC database. Note that the SCUD-B is a liquid rocket missile so generating a SRM version will require generating geometries for the solid propellant using the GA. The GA was set to match a range and apogee of 250 km and 75 km, respectively.

Table 5 AUSRC SCUD-B Input

Parameter	Parameter Value
Nose Diameter to Body Diameter Ratio***	0.1936
Nose Length to Body Diameter Ratio	3.25
Propellant Type	3 (PVC/AP/AL)
Outer Propellant Radius Plus Fillet Radius to Body Diameter Ratio***	0.6095
Inner Propellant Radius to Outer Propellant Radius Ratio***	0.2165
Number of Star/Wagon-Wheel Points***	7
Fillet Radius to Outer Propellant Radius Ratio***	0.089
Epsilon Width***	0.91821
Star/Wagon-Wheel Point Angle, Degrees***	18.4816
Fractional Nozzle Length***	0.7573
Nozzle Throat Diameter to Body Diameter Ratio	0.1426
Fineness Ratio	9.119
Body Diameter, meters	0.8840
Wing Semi-Span to Body Diameter Ratio	0.0
Wing Root Chord to Body Diameter Ratio	0.0
Wing Taper Ratio (Tip Chord to Root Chord Ratio)	0.0
Wing Leading Angle, Degrees (Measured Normal to Missile Body)	0.0
Wing Leading Edge X-Location to Body Length Ratio	0.0
Tail Semi-Span to Body Diameter Ratio	0.51697
Tail Root Chord to Body Diameter Ratio	1.50905
Tail Taper Ratio (Tip Chord to Root Chord Ratio)	0.42879
Tail Leading Angle, Degrees (Measured Normal to Missile Body)	60.00
Tail Trailing Edge X-Location to Body Length Ratio	0.88
Autopilot Time Delay, Seconds***	231.80
Autopilot Time Constant (τ)***	0.4044
Autopilot Damping Coefficient (ζ)***	0.9140
Autopilot Crossover Frequency, (rad/sec)***	34.4028
Autopilot Effective Navigation Ratio***	3.741
Initial Launch Angle, Degrees***	53.63

***Values Generated from Genetic Algorithm

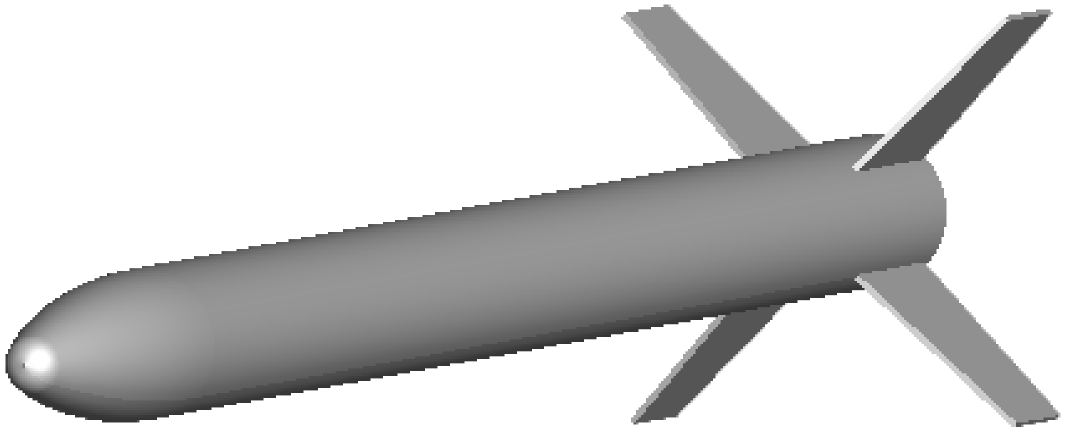


Fig. 16 AUSRC Input Example.

II.XI Program Errors & Filtering

Due to settings in the Auburn tools, unsatisfactory conditions can occur during simulation time. Some of the conditions are geometrical constraints, thrust constraints, and the majority are due to conditions during the 6-DOF subroutine. Below the current errors and filters are listed. Errors 1-7 will stop the simulation immediately when detected, errors 8-11 will still output data until the error occurs. For the databases, we wish to use missiles which do not experience failure of any kind, so when filtering the missiles, errors 1-11 do get reported. Filter 12 shows that there can still be thrust when the missile lands, which means the missile did not go far enough so it gets filtered. Filter 13 is not an error in the program, it just shows the missile did not go farther than 10 miles, so they get filtered out of the database. Filter 14 is also not an error, it is an effect due to errors 8-11 since the simulation stops tracking the missile, with the database we want full complete trajectories, and so if errors 8-11 occur, they get filtered out of the database.

1. Grain Type Unspecified (AUSRC), this parameter is $kctr = 1$ and checks many geometric constraints for the propellant grain. Examples include: $\left(\mathrm{H}_1 = \mathrm{R_P}\sin\left(\frac{\pi\epsilon}{\mathrm{N}}\right)\right) <$ fillet radius; wagon wheel spokes are too long (they are geometrically intersecting, for no spoke interference $\delta < \frac{\pi}{N}$), grain perimeters are less than zero (cannot be negative geometry length).
2. For AUSRC & AULRC: Expansion ratio should be within 2-50; Fractional nozzle length should be greater than 0.6 and less than 1.0.
3. For AUSRC & AULRC: Chamber pressure exceeded max limit, which is currently set to 9000 psi.
4. For AUSRC: Port area less than 1.2 times nozzle throat area.
5. For AUSRC & AULRC: Thrust is too small.
6. For AUSRC: Propellant grain size too small.
7. For AUSRC: Exit Mach number less than 1.0 (Should be supersonic nozzle designs).
8. For AUSRC & AULRC: Missile is tumbling or rolling out of control.
9. For AUSRC & AULRC: Max G-limit exceeded.
10. For AUSRC & AULRC:: Max Mach number limit exceeded.
11. For AUSRC & AULRC: Wing/Tail fin sheared off.
12. For AUSRC & AULRC: Thrust still occurring, means rocket did not go far enough to finish the entire propellant.
13. For AUSRC & AULRC: Missile did not go further than 10 miles.
14. For AUSRC & AULRC: Final altitude is not zero, means missile did not land.

III. Data Generation

Both regression and classification tasks require extensive databases for training models. Databases can be built multiple ways. Unrelated to the AULRC and AUSRC, an example would be taking data from the population and digitizing data into large spreadsheets. Data could also be gathered from the internet using surveys. This work uses the AULRC and AUSRC to build the databases for both regression and classification. Both programs use a subroutine constructed to randomly generate inputs that will automatically be simulated in the AULRC or AUSRC. There are many ways of randomly generated numbers, and the Auburn tools have several programmed. These numbers are generated randomly by sampling from normal or uniform distributions. Since the inputs to the Auburn tools are continuous, the distributions are also continuous. A good resource for random number generation can be found in Ripley [66].

There are five available options in the Auburn tools for sampling: normal distribution, uniform distribution, Latin Hypercube (LHC) sampling, LHC with edge sampling, LHC with centered sampling. In both Auburn tools, the data is input into a file called "gannlDIST.dat" and is used for every variation of the sampling algorithms unless the user changes the source code to read in a different file. Data will also be described for the regression and classification tasks and the summary statistics will also be shown to introduce the preprocessing tasks that are done to ensure data is acceptable for model generation.

III.I Uniform Distribution Sampling

The first and simplest sampling method available is uniform sampling [67-68], and the probability distribution function (pdf) is shown on Eq. (3.1), which is constant line between (a) and (b). Programmatically, (a) is the minimum value for the design parameter and (b) is the maximum value for the design parameter.

$$f(x) = \frac{1}{b-a}, \quad \mathrm{a} \le \mathrm{x} \le \mathrm{b} \tag{3.1}$$

To programmatically obtain uniform random values (Z_U), values are not directly calculated from uniform distribution on interval [a,b]; instead, values are calculated from a uniform distribution on interval [0,1] and are then scaled on [a,b] shown on Eq. (3.2). The uniform values on [0,1] are calculated using the Marsaglia-Zaman subtract with borrow random generator [69]. Generally, some form of uniform distribution is preferred in relation to the regression and classification tasks to ensure that the models are not biased to regions with higher sampling densities. Uniform sampling is more desirable so that models overall are training to entire regions of data and are not preferring specific regions. Essentially trying to reduce errors by minimizing the number of regions that may have reduced sampling density. Table 6 shows an example of input data and how it would look for uniform sampling.

$$Z_U = a + (b-a)\,Unif(0,1) \tag{3.2}$$

Table 6 Uniform Data Example for "gannlDIST.dat"

Maximum	Minimum	Parameter
2.0	0.5	DBODY, (meters)
15	10	Fineness Ratio

III.II Normal Distribution Sampling

The second sampling method is to sample from the pdf of the normal distribution, shown in Eq. (3.3). The main parameters of this function are the mean value (μ) and the variance (σ^2). The mean value, shown on Eq. (3.4), is just the average value or the expected value of *X,* where *X* is continuous random variable with a pdf of $f(x)$. The variance or standard deviation squared, shown on Eq. (3.5), is the expected value of *X* minus the mean value squared which shows the amount of spread from the average value. For more information on the normal distribution, see references [67] & [68].

$$f\left(x\right)=\frac{1}{\sigma\sqrt{2\pi}}e^{\left[\frac{-\left(x-\mu\right)^2}{2\sigma^2}\right]}, \quad -\infty<x<\infty \tag{3.3}$$

$$\mu=E\left(X\right)=\int_{-\infty}^{\infty}xf\left(x\right)dx \tag{3.4}$$

$$\sigma^2=Var\left(X\right)=E\left[\left(X-\mu\right)^2\right]=\int_{-\infty}^{\infty}\left(x-\mu\right)^2 f\left(x\right)dx \tag{3.5}$$

To programmatically sample from the normal distribution, the Box-Muller (BM) algorithm is used to efficiently generate random values [67-68]. Other variations of sampling can be done as well, for example the central limit theorem can be used to sample for normal distributions. BM uses two independent uniform distributions and converts them to polar coordinates (R, Θ), which are then multiplied and get used to calculate a random normal value (Z_N). This algorithm requires the user input the mean and standard deviation, and instead of directly sampling from the normal distribution, the uniform distribution can be used to generate random values. The uniform distribution uses the same algorithm mentioned in Eq. (3.2).

$$\begin{aligned}
\Theta &= 2\pi Unif\left(0,1\right) \\
R &= \sqrt{-2\ln\left(Unif\left(0,1\right)\right)} \\
Z_{BM} &= R\cos\left(\Theta\right) \\
Z_N &= \mu + \sigma Z_{BM}
\end{aligned} \tag{3.6}$$

Typically, in relation to work shown here, the normal distribution sampling is not used because normal distribution generates a bell-shaped curve density. Therefore, near the tail regions the sampling density decreases. Regions with low sampling density can increase error because models may not be able to fit the data near the tails where sampling density is low. Table 7 does show an example of what the user would input for the normal distribution. Only the mean and standard deviation are required.

Table 7 Normal Distribution Example for "gannlDIST.dat"

Mean	Standard Deviation	Parameter
14.33367	0.1433367	Fineness Ratio
0.75	0.0075	DBODY, (meters)

III.III Latin Hypercube Sampling

Uniform sampling in general does produce a uniform distribution, however it can develop regions where there is lower sampling. To better ensure that the dataset will have uniform sampling over every region, the Latin Hypercube (LHC) sampling algorithm can be used, which was developed from McKay [70]. LHC itself is a stratified sampling technique which divides the interval (like [a,b] in uniform distribution) into subregions or *strata*. Uniform distributions are then randomly sampled from each stratum, and this significantly improves the overall uniform distribution. Stratified sampling is an area of research which attempts to approve space filling designs. This work will only focus on LHC sampling for space filling designs and readers are encouraged to review [70-74] for more information on space filling designs and sampling methods. Other designs have further attempted to improve upon LHC by developing quasi-random (quasi-Monte Carlo) sequences and include Halton [71], Sobol sequences [76], and Niederreiter [77]. Another popular sampling method is based on the maximum entropy, which seeks to use probability distribution that has the largest entropy [78]. Future work in data generation will need to consider the methods from references [70-78].

Derivations of LHC sampling are shown in [70], [71], and [75]. This work will follow the derivation from Santner [71]. The inputs to the Auburn tools are $\mathbf{X} = (X_1, \dots, X_d)$ where (d) is the number of inputs to the Auburn tools, such that $1 \leq k \leq d$. There are a total number of samples (N_S), with individual samples denoted by (j), such that $1 \leq j \leq N_S$. There are then (N by d) or (j by k) LHC values, which is a matrix. The LHC algorithm defined by Eq. (3.7) first calculates a matrix of (N_S by d) uniform samples on interval [0,1] denoted by $\left(U_{jk}\right)$. Next, a matrix of (N_S by d) random permutations on interval [1, N_S]. Programmatically, the random permutation algorithm follows a shuffling algorithm described by Knuth [79] and the algorithm is essentially equivalent to sampling without replacement. The shuffling algorithm also uses an in-house pseudo-random integer generator developed from Wichmann & Hill [80]. The matrix of (N_S x d) LHC values can then be calculated and since $\mathbf{X} = (X_1, \dots, X_d)$ are uniformly sampled over $[0,1]^d$ then $F_k^{-1}(x) = x$.

LHC_{jk}

LHC_{jk} are scaled between 0-1 and need to be rescaled over the interval $[a_k, b_k]$ for each k input, which are denoted by $SLHC_{jk}$.

$$\begin{aligned}
U_{jk} &= Unif\left(0,1\right) \\
\Pi_{jk} &= RandPerm\left(1, N_S\right) \\
LHC_{jk} &= F_k^{-1}\left(\frac{\Pi_{jk} - 1 + U_{jk}}{N_S}\right) = \frac{\Pi_{jk} - 1 + U_{jk}}{N_S} \\
SLHC_{jk} &= a_k + \left(b_k - a_k\right) LHC_{jk}
\end{aligned} \tag{3.7}$$

The algorithm described in Eq. (3.7) is for the original LHC algorithm and is what is mostly used in this work. There are many ways that the LHC algorithm has been modified, but only two others are included, and they are LHC-Center and LHC-Edge algorithms. Both of these algorithms are slight variations with the LHC-Center is more likely to generate random values in the center of each stratum as opposed to LHC-Edge which is more likely to generate random values near the edges of each stratum. Typically, the edge and center version are not used because as long as N is large (greater than 1000, which it usually is) then the sampling is uniform over each stratum regardless of which version is used.

$$\begin{aligned}
LHC_{Center} &= \frac{2\Pi_{jk} - 1 + U_{jk}}{2N} \\
LHC_{Edge} &= \frac{\Pi_{jk} - 1 + U_{jk}}{N - 1}
\end{aligned} \tag{3.8}$$

Special LHC designs include orthogonal arrays, symmetric LHC, cascade/nested/sliced LHC, orthogonal LHC (not the same as orthogonal arrays), symmetric LHC, and LHC designs that use Euclidean Distance [71]. Programmatically, Eq. (3.7) is the simplest method to follow since it only relies on the user inputting the maximum and minimum values for each input [*a, b*] similar to the inputs on Table 6.

III.IV Regression & Classification Data

This section will describe the inputs and outputs for the classification and regression tasks for both the AULRC and AUSRC datasets. Readers should understand that the datasets were based on the SCUD-B from Table 3. Summary statistics will be introduced and shown for various parameters. Summary statistics are extremely useful since it is a part of preprocessing data. Looking at the summary statistics is mostly an observation task, and it is important to ensure that the summary statistics are what the user expects. The summary statistics are also used to ensure that there are no issues in the data that was generated, and these issues could be outliers or problems in the code that generated the data. Also, the information shown in the following sections are developed in Python [1] using Pandas [81] & [82], Matplotlib [83], seaborn [84], and Plotnine based on 'ggplot2' in R [85].

The first set of summary statistics will describe the count, mean, standard deviation, minimum value, 25th Percentile, 50th Percentile, 75th Percentile, and maximum value, like what is shown on Table 8 excluding the count. Table 8 can easily be generated in Python using Pandas "describe" function. The count is just the number of samples in the dataset. The mean would be the average value for the parameter. Standard deviation is the square root of the variance of the parameter. Min and Max are just the minimum and maximum values of the parameter, respectively. The quartiles for 25%, 50% (Median), and 75% show percentage of where data lies, so 25% shows 25 percent of data lies below the value shown. Readers should note that the minimum and maximum values are what was input to the Auburn tools and should match relatively close when data is output.

The “describe” function provides a great set of summary statistics; however, the distribution sampling must be confirmed and is best done by using a density histogram [67]. Histograms group data into intervals by either defining number of bins or interval range. The density of each interval is displayed using a bar and the height is equal to the number of samples in each interval. To confirm the LHC or uniform sampling mode is providing a uniform distribution then the bars should be of equal density and would show equal height on a figure. Despite all efforts that have been introduced to produce uniform distributions, the upcoming sections will show some of the histograms not having uniform distribution at all. Mainly this is due to filtering from the Auburn tools because not every design configuration gives a conceivable trajectory.

III.IV.I AULRC Regression Data

For the regression tasks, the output data is to be modeled as a function of the inputs. First, the database must be described to show what inputs are used and why other inputs were excluded. Table 8 shows the summary statistics of both input and output developed using the LHC sampling algorithm from Eq. (3.7). Displayed on Table 8 is the parameter name, the mean, standard deviation, minimum value, 25th Percentile, 50th Percentile (median), 75th Percentile, and maximum value. The number of samples or count that was generated for this dataset was 500,000 samples and takes approximately 75.5 hours to execute. Since the wing parameters were excluded, they are all zero. Because the autopilot was turned off for both the vanes and tail fins, they are excluded as well. Figure 97 in Appendix D: shows how the input file is setup for the AULRC with no autopilots turned. Appendix E: & Appendix F: show datasets using fin autopilot and fin/vane autopilot, respectively. The data describes a range of ballistic missiles.

Table 8 AULRC Regression Data

Parameter	Mean	STD DEV	MIN	25%	50%	75%	MAX
DBODY	3.02	0.29	2.50	2.77	3.02	3.27	3.50
EQRATIO	2.00	0.12	1.80	1.90	2.00	2.10	2.20
PC	1093.63	247.56	600.00	892.28	1108.03	1306.45	1500.00
DNOSE	0.85	0.09	0.70	0.77	0.85	0.92	1.00
LNOSE	3.25	0.19	2.93	3.09	3.25	3.41	3.58
THROAT	0.15	0.03	0.10	0.13	0.16	0.18	0.20
EXPR	16.35	4.89	8.00	12.10	16.27	20.56	25.00
FNL	0.80	0.12	0.60	0.70	0.80	0.90	1.00
TBURN	61.23	9.82	45.00	52.76	60.69	69.34	80.00
TRCR	1.26	0.43	0.50	0.89	1.26	1.63	2.00
TTR	0.66	0.18	0.35	0.50	0.65	0.81	1.00
TAILB2	0.97	0.29	0.50	0.72	0.95	1.21	1.50
TLE	18.56	11.07	0.00	9.08	18.16	27.61	40.00
TXLERATIO	0.88	0.07	0.75	0.81	0.88	0.94	1.00
ILAUNCH	84.91	2.66	80.00	82.73	85.01	87.15	89.99
THRSEA	40.31	18.38	7.32	26.37	36.41	50.64	126.91
MAXTHR	45.58	20.57	7.82	29.85	41.51	57.44	140.75
MAXDIST	841.19	739.48	52.80	231.87	610.78	1268.77	6713.27
APOGEE	298.69	318.13	5.44	50.15	187.71	448.04	3573.74
TOF	280.07	149.47	46.42	151.60	263.21	383.68	1133.21
WEIGHT	16.56	6.45	5.17	11.65	15.23	20.19	51.05

Box and whisker plots (boxplots) show the bottom whisker and top whisker, which are the minimum and maximum value, respectively, and the box display the 25th, 50th (median), and 75th percentiles (quartiles). Figure 18 shows the box plots for DBODY, PC, TAILB2, and ILAUNCH. We can see that DBODY displays the quartiles at relatively uniform distribution and PC shows the quartiles shifted up (skewed left). TAILB2 appears to be slightly skewed right and ILAUNCH seems very uniform.

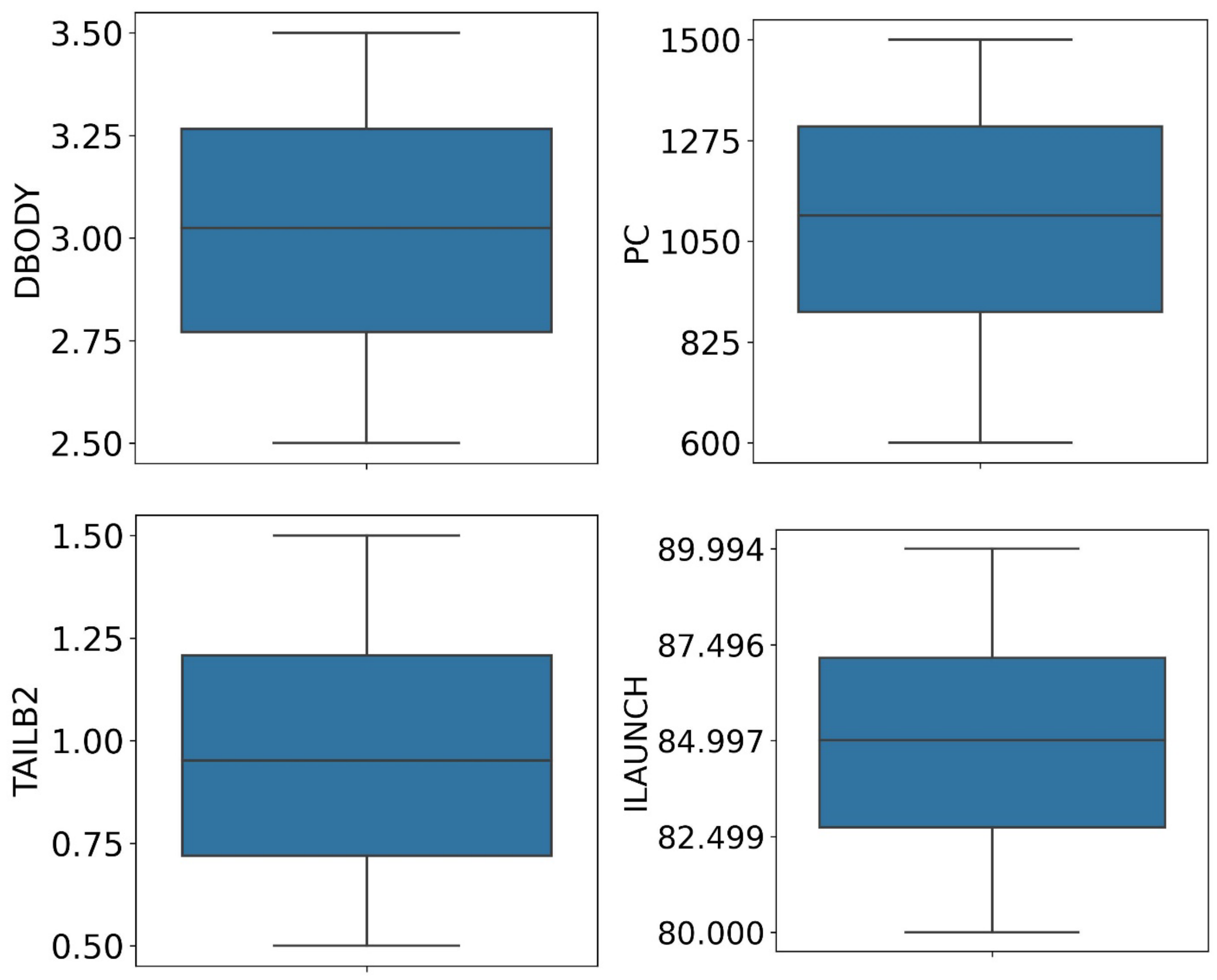


Fig. 17 Box & Whisker Plot for DBODY (Top-Left), PC (Top-Right), TAILB2 (Bottom-Left), & ILAUNCH (Bottom-Right).

The whole reason for using LHC sampling was to get better uniform distribution. To better display the sampling distribution, histograms can be used to show the frequency of values using bins and show an approximate view of the distribution. So, if the values are approximately constant then the distribution can be said to be approximately uniform. Figure 19 displays the histograms for DBODY with count being the number of samples with the specified bin width. The count increases as DBODY increases from 2.5 to 3.5 but seems uniform overall and it could also be said that higher DBODY values are preferred and lower DBODY are less likely to occur. Due to filtering, higher DBODY may be more stable during flight. If we plot the max distance (MAXDIST) versus DBODY, shown on Fig. 98 in Appendix G:, shows two groupings of data. The first group seems to start at MAXDIST equal to 2000 (DBODY=2.5) and increases linearly as DBODY goes to 3.5. The second group starts at MAXDIST equal to 2000 (DBODY=2.5) and can see that the MAXDIST goes up to 5000 and increases to 6500 as DBODY increases but the frequency starts to decrease drastically once DBODY gets to 3.0. This trend is also noticeable on APOGEE and TOF versus DBODY.

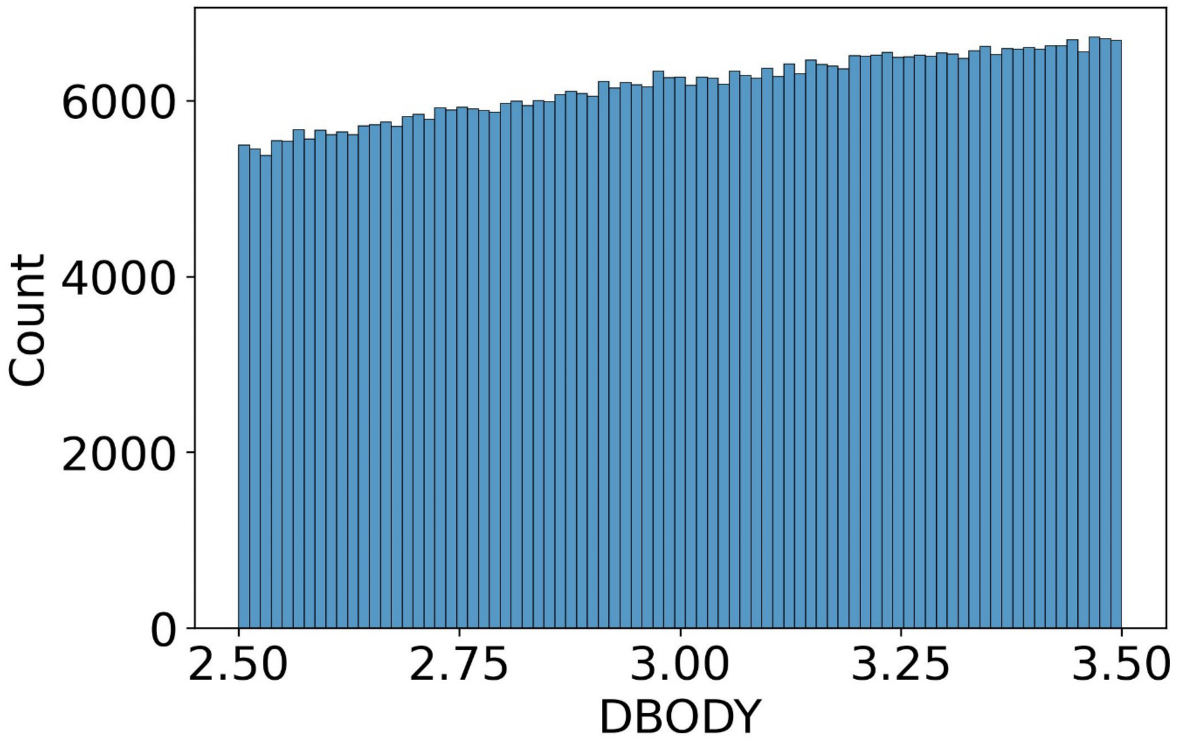


Fig. 18 DBODY Histogram.

PC count increases quite drastically as PC increases and is not uniform in sample distribution. Higher values of PC are apparently preferred. Since PC directly results thrust, lower values of thrust are less likely to launch far and may not launch at all if the weight is too much, therefore larger values of PC are more desirable. When plotting the outputs versus PC on Fig. 99 in Appendix H: can see that there are not multiple groupings due to PC. Instead, we see that the outputs and their range increase as PC increases. Notice that as PC increases the minimum value is still constant so it shows that missiles with PC can still have low MAXDIST or TOF. Other parameters are still affecting the missile causing the minimum value to be constant, even as PC increases.

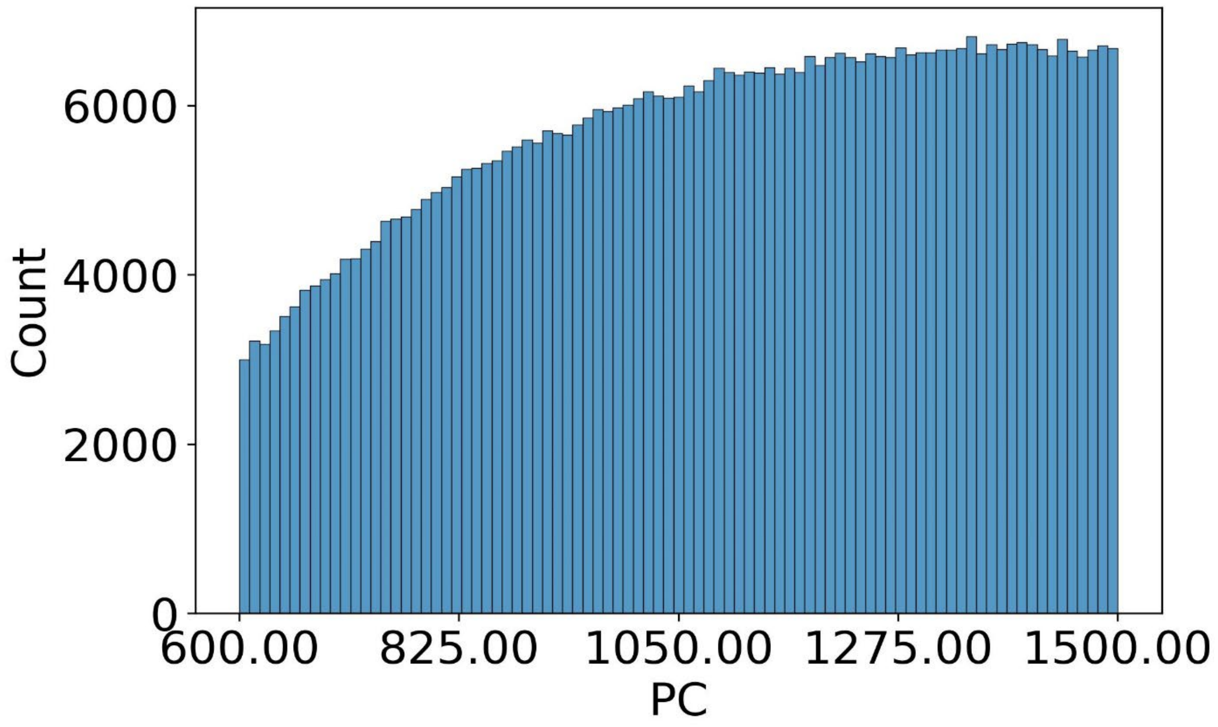


Fig. 19 PC Histogram.

Next, looking at Fig. 21, which displays the histograms for TAILB2, the number of missiles decreases as TAILB2 increases and smaller TAILB2 values are more likely to occur. Larger values of TAILB2 can contribute greatly applied momentum forces on the fins and fin shearing can occur at high speeds, so large fins are more likely to break resulting in unstable trajectory and would then be filtered out. Larger fins could also be causing more drag to influence the trajectory and causing large stability changes. When plotting the outputs versus TAILB2 in Appendix I:, the outputs are approximately uniform (constant) for every output, so it is hard to prove exactly why larger TAILB2 are less frequent except for hypothesizing that the fins are shearing as they get larger.

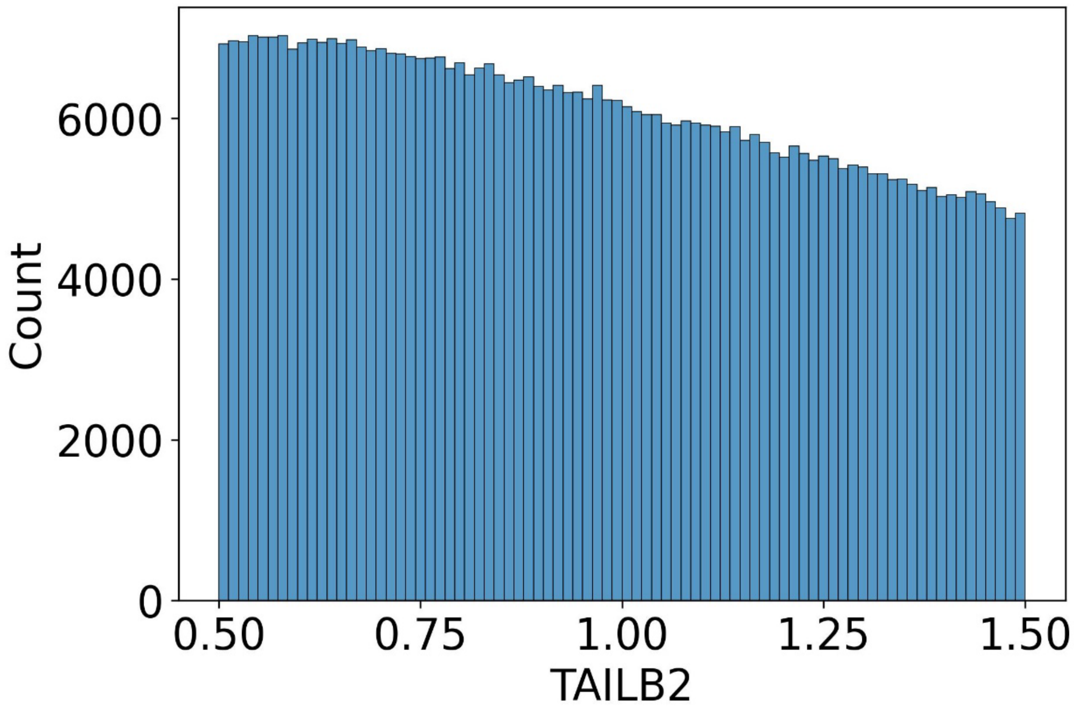


Fig. 20 TAILB2 Histogram.

Looking at Fig. 22, ILAUNCH count decreases as ILAUNCH increases. and ILAUNCH ranging from 85.00-87.50 degrees seems to be preferred. A huge reason why the frequency drops when ILAUNCH approaches 90 degrees is because the missile has no vane autopilot to pitch it over and so many vertical launches will fail because they are less likely to go further and may get filtered out of the database. The pitching angle should not be set to too low otherwise the missile may not go as far. Liquid missiles do not generate the same amount of acceleration as solid propellant rockets, which is why they are usually launched vertically and then pitched over. Future datasets will pitch vertically, then the vane autopilot will be used to pitch over.

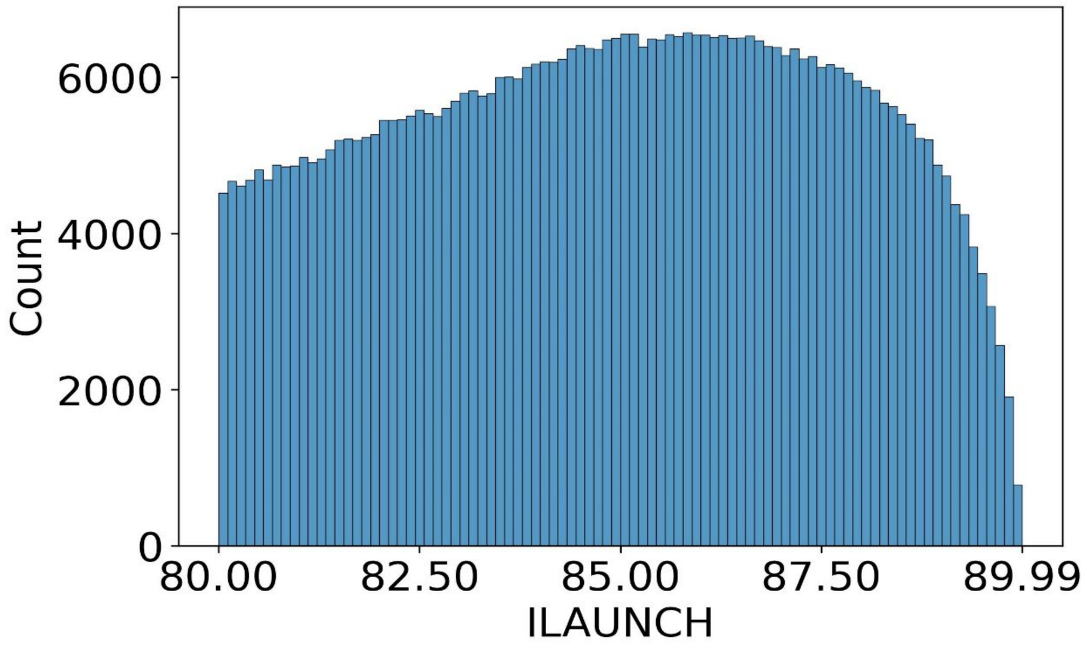


Fig. 21 ILAUNCH Histogram.

When plotting the outputs versus ILAUNCH on Fig. 101 in Appendix J: shows that APOGEE increases as ILAUNCH increases, which makes sense 90-degree vertical launches are most likely to go highest. However, MAXDIST drastically decreases as ILAUNCH approaches 90 degrees because there is no nozzle vane pitching allowing MAXDIST to increase, so instead APOGEE increases as ILAUNCH approaches 90 degrees. MAXTHR and THRSEA are uniformly distributed as ILAUNCH goes from 80 to 90 degrees. TOF increases as ILAUNCH increases, but we know from Fig. 22 that the frequency decreases greatly as ILAUNCH approaches 90 degrees. ILAUNCH has little to no effect on WEIGHT, which makes sense because WEIGHT itself is constructed from geometric terms and has such a low range compared to TOF. ILAUNCH only practically affects WEIGHT if ILAUNCH becomes too small, then the missiles may not be able to launch due to WEIGHT.

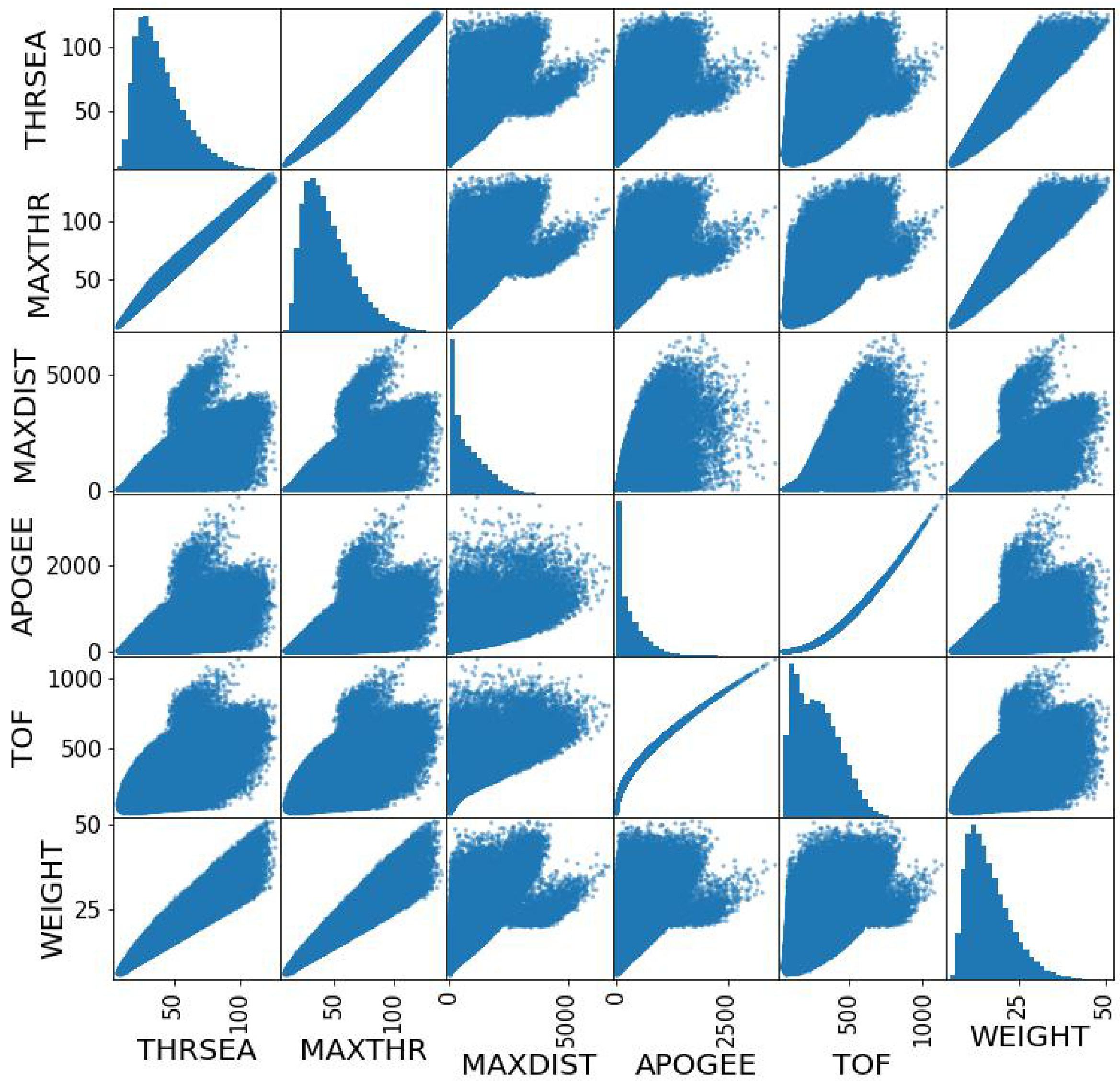


Fig. 22 AULRC Output Data Scatter Matrix.

Next, the outputs need to be evaluated using a scatter matrix to compare the histograms of the outputs, and also compare the scatter plots of the outputs versus each other using pandas “scatter matrix”. Figure 23 shows the scatter matrix of the output and will see that the diagonal are the histograms, and the off diagonals show the scatter plots. The histograms appear skewed right, meaning that the max values in the range are less frequently to occur. For example, one reason why longer TOF is less frequent to occur is that longer flights are more likely to experience an error such as tumbling or fin shear. The scatter plots are useful to see if there are any relationships between the output variables. These relationships can help understand how separable the data will be and can be used to see which parameters will do well in the regression analysis.

For example, the relationship between MAXTHR versus THRSEA is very linear and means that the two are related, which is obvious since they are both thrust terms. We cannot say how easy it will be to train models to MAXTHR & THRSEA, but we can hypothesize that they will be reproducible. Looking at TOF versus APOGEE shows a curvilinear relationship and suggests that as TOF increases, as does APOGEE and so they are directly related to each other for rocket propelled lofted ballistic trajectories are subject to the modeling in this dissertation.

We now put focus on APOGEE & TOF versus MAXDIST and can see that both have a wide distribution and can say that APOGEE & TOF may have a more nonlinear relationship; thus, may be more difficult to reproduce using regression methods. It can be seen that certain scatter plots appear to have multiple distributions such as MAXDIST versus WEIGHT and suggest they may be more challenging to reproduce. When looking at MAXTHR, THRSEA, & WEIGHT histograms, they are normal with a right skew. Also notice that their range is small compared to the other outputs. So, they should be relatively easy to reproduce. TOF histogram is also normal with right skew but has a much larger range so it might be more difficult to reproduce. MAXDIST

& APOGEE are very right skewed but appear to be exponentially distributed so they may be more challenging to reproduce.

Some of the sample thrust curves are plotted on Fig. 24 and can see how thrust changes as time increases during flight. The reason why thrust increases is because during flight the atmospheric pressure decreases as altitude increases. It will also be seen that some of the thrust curves decrease because some of the flights are shorter duration. The flights are decreasing in altitude causing thrust to be slightly less due in part to increases in atmospheric pressure on average, and more prominently scaled.

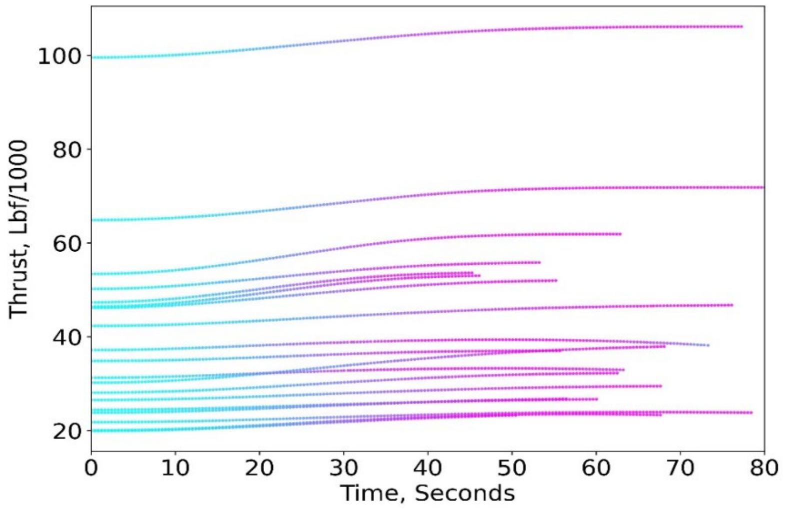


Fig. 23 Sample Thrust Curves.

When plotting the sample altitude versus time curves on Fig. 25 can see how the altitude varies greatly with time. Can see that the higher altitude trajectories have longer flight times and vice versa. If we look at close-up of the sample altitude versus time curves on Fig. 26, can see a few of the samples do not travel that long and so the thrust starts to decrease as altitude decreases and was shown on Fig. 24. The sample altitude versus range is shown on Fig. 27 and we can see the range that the missiles can have and will notice that the samples that go up more vertically are less likely to travel further compared to the missiles that are pitched over. A close-up of Fig. 27

shown on Fig. 28 shows that missiles pitched too low do does not go as far and will notice that they have lower burn times and lower thrust.

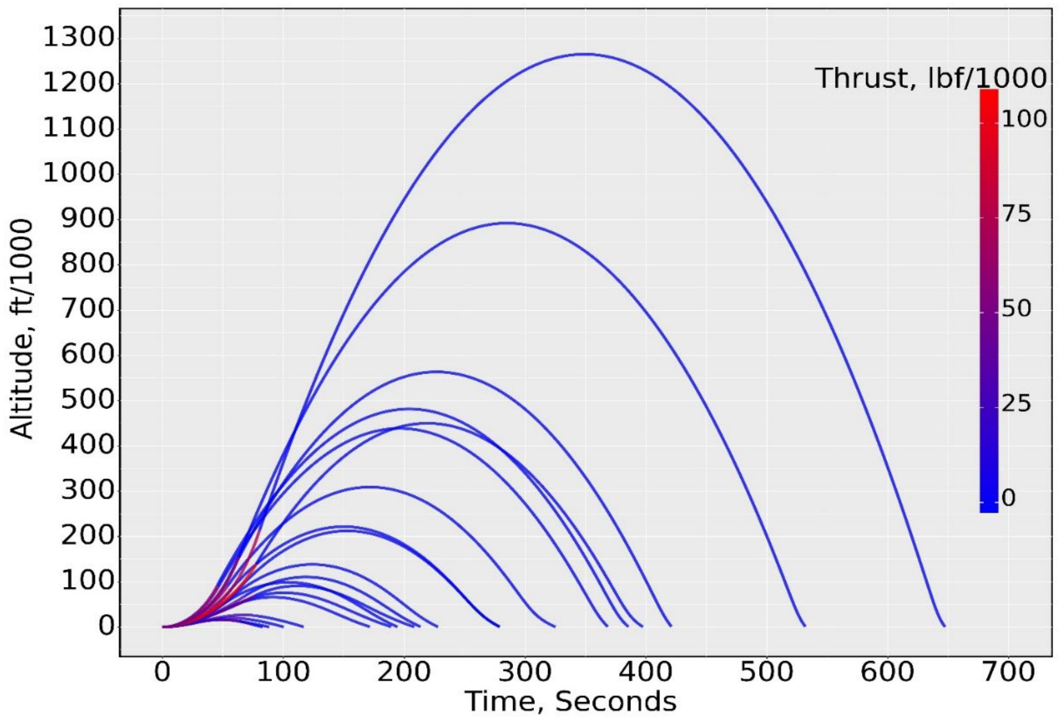


Fig. 24 Sample Altitude vs. Time Curves.

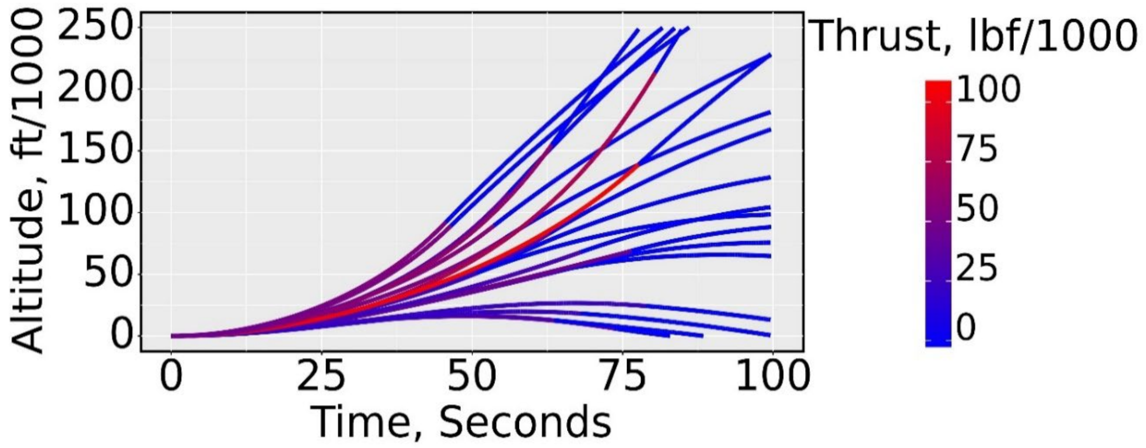


Fig. 25 Close-up Sample Altitude vs. Time Curves.

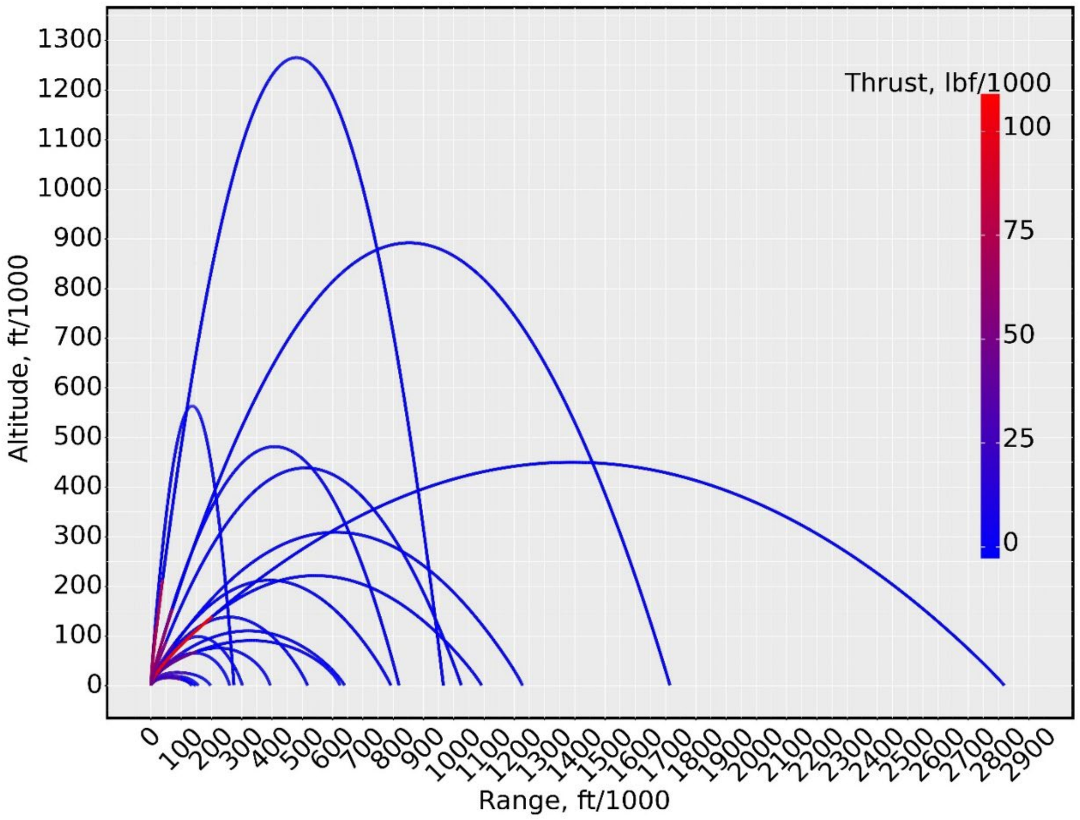


Fig. 26 Sample Altitude vs. Range Curves.

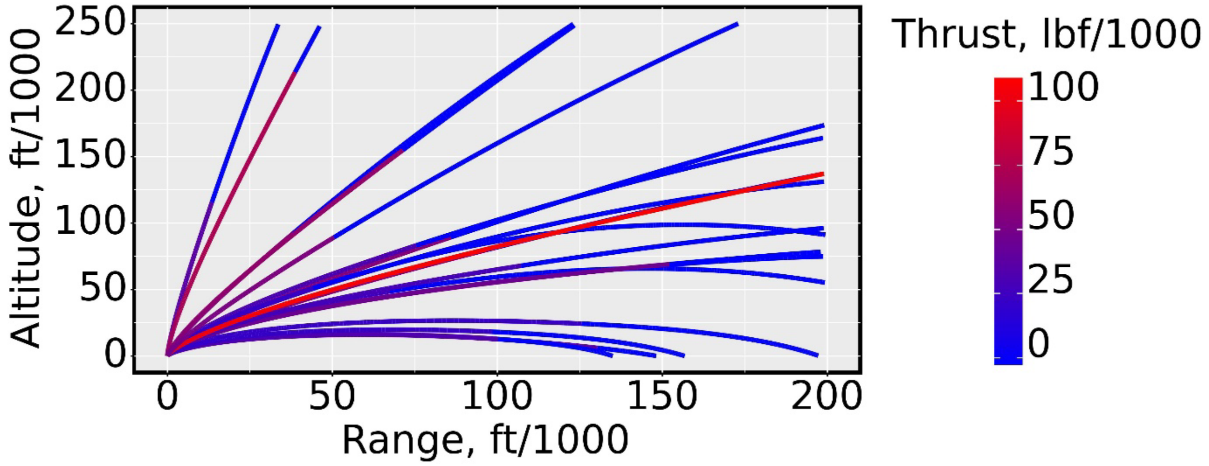


Fig. 27 Close-up Sample Altitude vs. Range Curves.

III.IV.II AUSRC Classification Data

For the classification tasks, the output data from the AUSRC is used as the inputs in the classification model and outputs a class value, which refers to the group (class) the missile belongs to. One reason for using classification models is to reverse engineer parameters. Originally, before the Auburn group tried to classify missiles, reverse engineering methods were used to try and develop the missile parameters using regression methods. However, NNETs were not able to accomplish this goal because there are too many missile parameters to model. For example, using output data such as BURNTIME and MAXTHR, could we rebuild the missile geometries such as DBODY (Body Diameter) and FINENESS (fineness ratio). So, instead of trying to rebuild the geometry parameters, we would give different groups of missiles a unique class value. This class value, in a way, tells us what geometry the missile has without actually needing to know what the geometry is.

To rebuild the geometry using output data, a more advanced network called Generative Adversarial Networks (GANs) may need to be used but is outside the scope of this work. Instead, this work started out with classification models. Again, we begin by describing how the classes were developed. For this work, we only utilize 7-star point grains. Originally, during the 2021-2022 MSIC contract we developed 72 classes by modifying number of star points from 5,7,9, and 11 points, nozzle throat diameter, fineness ratio, and body diameter. Due to excess amount of information, it is not easy to visualize every star grain configuration and so we only focus on a subgroup of the database with 7-star points for this work. Table 9 shows the summary statistics used to build the database for the classification models. The four outputs utilized for classification are BURNTIME, MAXTHR, MAXPC, and MAXPE.

Table 9 Classification Database Summary Statistics

Parameter	Mean	STD DEV	MIN	25%	50%	75%	MAX
BURNRATE	0.040	0.000	0.040	0.040	0.040	0.040	0.041
PREXP	0.350	0.002	0.347	0.348	0.350	0.352	0.354
DENSITY	0.064	0.000	0.063	0.064	0.064	0.064	0.065
CSTAR	5363.98	30.95	5310.36	5337.27	5364.15	5390.78	5417.64
RPVAR	0.610	0.006	0.600	0.605	0.610	0.615	0.620
RIVAR	0.216	0.006	0.206	0.211	0.216	0.221	0.226
FVAR	0.089	0.005	0.081	0.085	0.089	0.093	0.097
EPS	0.918	0.023	0.878	0.898	0.918	0.938	0.958
PTANG	18.480	4.040	11.482	14.989	18.477	21.982	25.481
FNL	0.757	0.006	0.747	0.752	0.757	0.762	0.767
THROAT	0.163	0.019	0.140	0.144	0.163	0.183	0.189
LBODY	9.119	0.163	8.909	8.924	9.119	9.314	9.329
DBODY	0.884	0.033	0.842	0.845	0.884	0.923	0.926
LGRAIN	2.379	0.170	2.078	2.220	2.379	2.541	2.683
BURNTIME	16.864	1.422	13.404	15.746	16.812	17.935	21.094
MAXTHR	108.24	20.26	65.63	92.94	106.47	121.59	176.44
MAXPC	2499.53	882.79	1196.97	1673.06	2330.49	3290.06	4657.67
MAXPE	4.636	0.571	3.447	4.175	4.611	5.064	6.377

Table 10 shows how the classes are differentiated from throat diameter, fineness ration, and body diameter. We will see that the first 9 classes have the same throat diameter. Each class changes in body diameter and each 3 classes the fineness ratio increases. To generate the complete database, the AUSRC is executed 18 times (72 times to include the other star point configurations). Appendix K: shows an example input file used to generate data using the AUSRC. Throat diameter has a +/- 0.006 for maximum and minimum, respectively. Fineness ratio has a +/- 0.01 for maximum and minimum, respectively. Body diameter has a +/- 0.002 for maximum and minimum, respectively. The rest of the inputs also have a small amount of noise to add variability to the output. Noise is also added because the regression models cannot use constant values. For each class, 2500 samples are generated for a total of 45,000 samples (180,000 samples including the extra star point configurations).

Table 10 Mean Value Class Differentiation

Class	Throat Diameter	Fineness Ratio	Body Diameter
1	0.14264	8.91878	0.844
2	0.14264	8.91878	0.884
3	0.14264	8.91878	0.924
4	0.14264	9.11878	0.844
5	0.14264	9.11878	0.884
6	0.14264	9.11878	0.924
7	0.14264	9.31878	0.844
8	0.14264	9.31878	0.884
9	0.14264	9.31878	0.924
10	0.18264	8.91878	0.844
11	0.18264	8.91878	0.884
12	0.18264	8.91878	0.924
13	0.18264	9.11878	0.844
14	0.18264	9.11878	0.884
15	0.18264	9.11878	0.924
16	0.18264	9.31878	0.844
17	0.18264	9.31878	0.884
18	0.18264	9.31878	0.924

Because there are many classes, looking at the histograms is not simply accomplished, since the distributions are overlayed. Instead, we utilize the box and whisker plots to see how the classes are differentiated by looking at DBODY, FINENESS, THROAT, BURNTIME, MAXTHR, MAXPE, and MAXPC. Can see DBODY for each class on Fig. 29 and see the small amount of noise that was added (+/- 0.002). FINENESS for each class is shown on Fig. 30 and can see the small amount of noise that was added (+/- 0.01). THROAT for each class is shown Fig. 31 and can see the small amount of noise that was added (+/- 0.006).

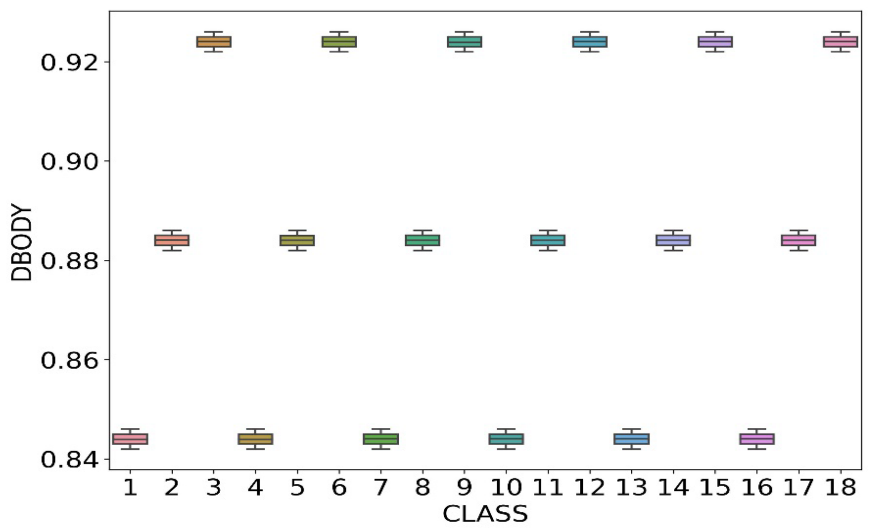


Fig. 28 DBODY Box & Whisker Plots of Each Class.

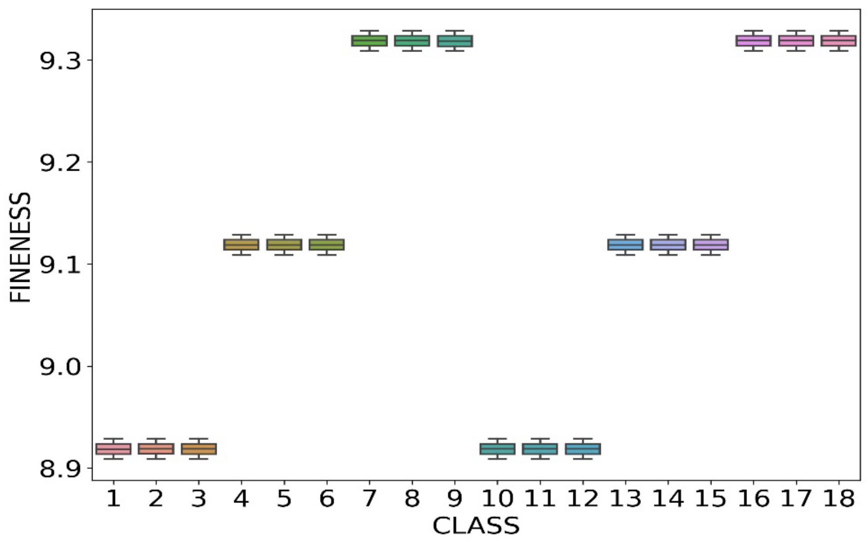


Fig. 29 FINENESS Box & Whisker Plot of Each Class.

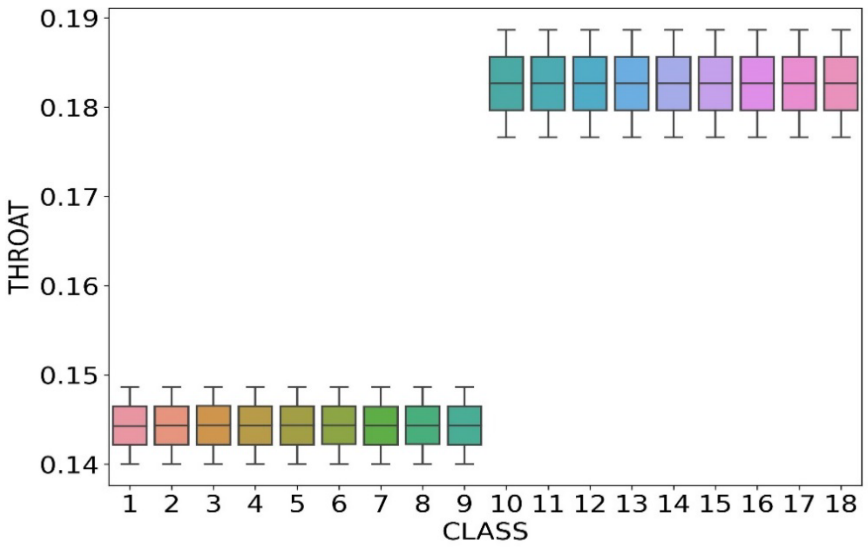


Fig. 30 THROAT Box & Whisker Plot of Each Class.

When looking at BURNTIME on Fig. 32, can see how BURNTIME increases with each DBODY increase, which makes sense because the wider the rocket the more fuel that can be carried. Notice BURNTIME drops as DBODY resets to lowest value. Can see that for each 3 class groups (first 9 classes) BURNTIME decreases as FINENESS ratio increases. As fineness ratio increases the grain length increases slightly and the max pressure increases more significantly. If we assume the burn time is steady, which it true for cylindrical grains, then we can approximate burn time as $t_b = \mathrm{LGRAIN}/(\mathrm{a}P_O^N)$. For demonstration purposes only, if LGRAIN=2.1 with MAXPC=3000 psi gives $t_b = 2.13$ seconds and LGRAIN=2.5 with MAXPC=3750 psi gives $t_b = 1.08$ seconds. Can directly see that an increase in fineness ratio causes slight increase in grain length but a VERY significant increase in chamber pressure and therefore a decrease in burn time.

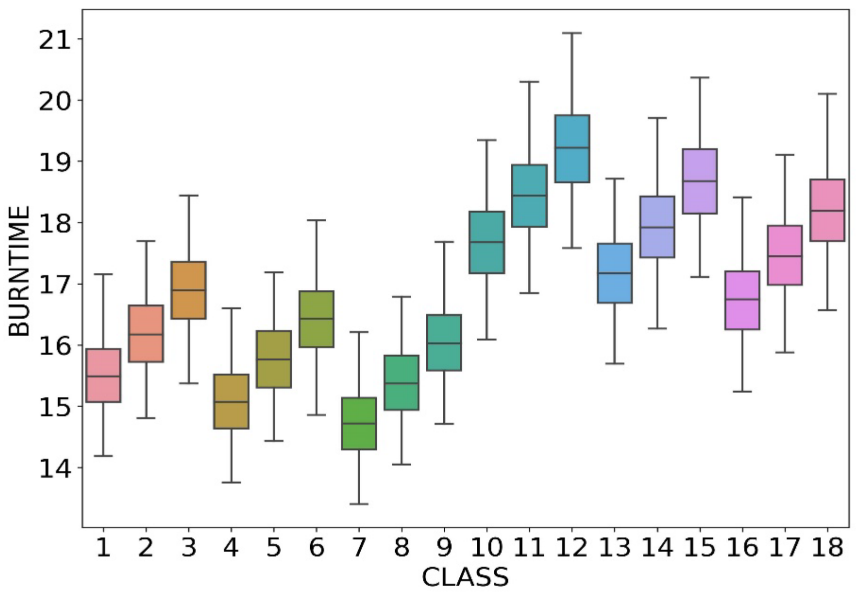


Fig. 31 BURNTIME Box & Whisker Plot of Each Class.

An increase the nozzle throat diameter causes the nozzle length to be shorter which then causes the propellant grain length to be longer for classes 10-18; therefore, the propellant burn time is longer since the grain is longer. If we compare classes 1 & 10 using Eq. (2.29), they have the same DBODY and FINENESS, but different THROAT. To see what is causing the difference in BURNTIME, we know that L_B is constant for both cases, the starting point of the grain x_{grain} is also constant, and the body radius is constant. The only parameter that changes is the nozzle length, if we then analyze Eq.'s (2.29) & (10.1)-(10.4) in Appendix B:, the only parameter causing any difference is THROAT. An increase in THROAT reduces the available nozzle length, which then increases the grain length (shown on Fig. 33) and vice versa. An increase in grain length then increases the BURNTIME, which makes sense because more available propellant the longer it takes to burn propellant.

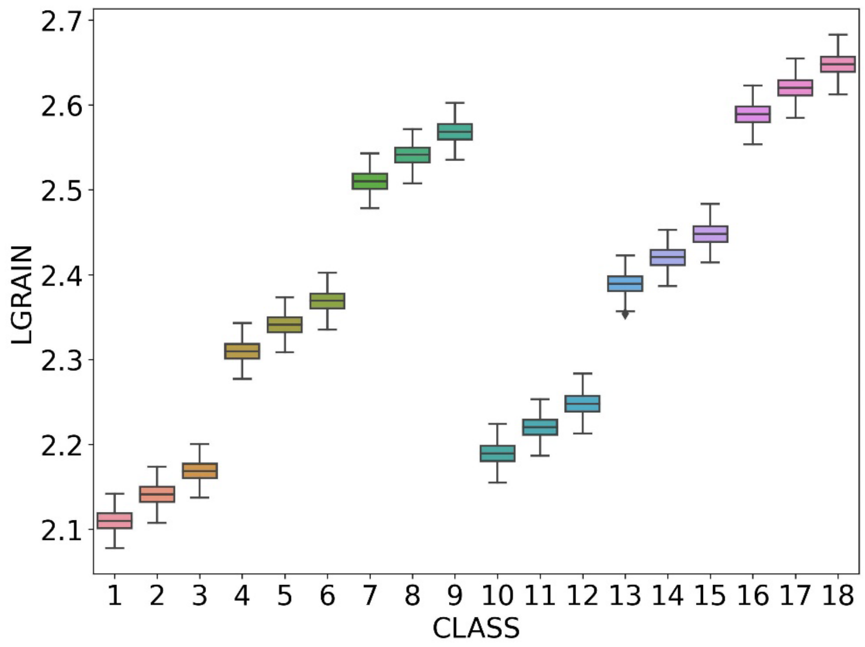


Fig. 32 LGRAIN Box & Whisker Plot of Each Class

Next, we can look at Fig. 34, which shows MAXTHR box plots for each class. Can see that as DBODY increases, MAXTHR increases, and as FINENESS increases, MAXTHR increases. Increasing THROAT decreases MAXTHR, analyzing Eq. (2.30) and comparing classes 1 and 10, which both have the same DBODY and FINENESS, burn area and the propellant properties are constant; the only difference is THROAT. Calculating Eq. (2.30) using THROAT from classes 1 and 10 will then show that class 10, which has higher THROAT, yields a significant decrease in chamber pressure compared to the chamber pressure for class 1. This significant decrease in chamber pressure causes a huge decrease in MAXTHR (See Eq. (2.31)). Notice Fig. 35 and shows that the first 9 classes, which have the same THROAT, have a higher MAXPC, then classes 10-18, which have higher THROAT, have lower MAXPC. This trend is also seen on Fig. 36, which shows MAXPE, and can see that as DBODY & FINENESS increase, MAXPE increases, and when THROAT increases there is a reduction in MAXPE when comparing classes with same DBODY & FINENESS.

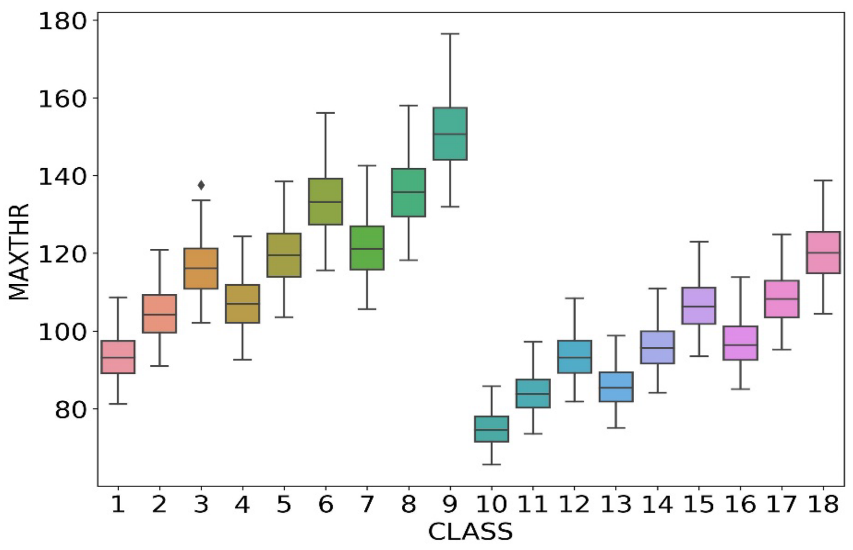


Fig. 33 MAXTHR Box & Whisker Plot of Each Class.

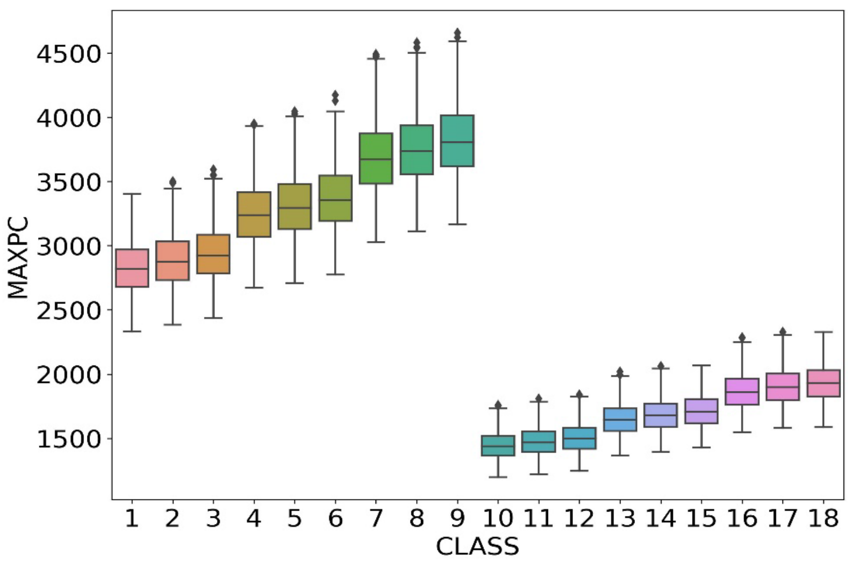


Fig. 34 MAXPC Box & Whisker Plot of Each Class.

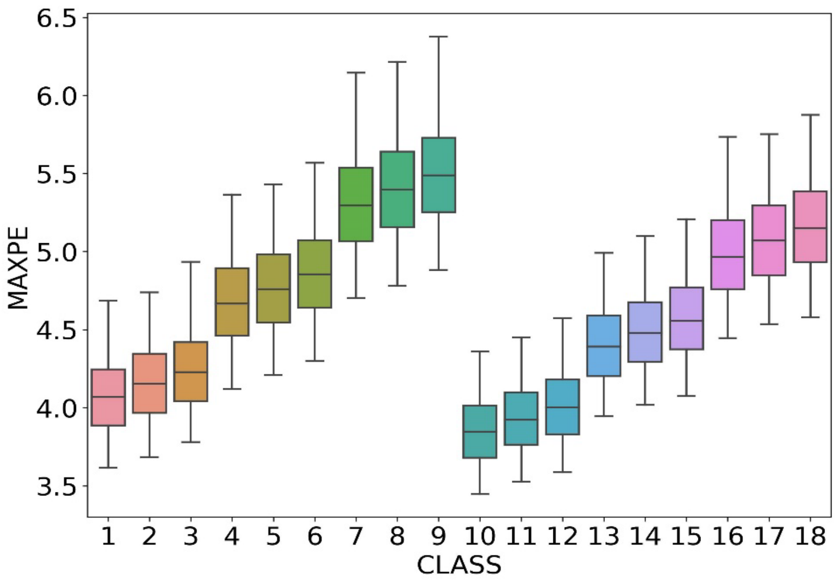


Fig. 35 MAXPE Box & Whisker Plot of Each Class.

IV. Model Generation: AULRC Regression

Now that the data has been introduced and described in Chapter III, we can begin the regression analysis. In this work, we first utilize traditional statistical learning methods such as linear regression, lasso regression, and ridge regression. We then recognize that linear regression cannot fully capture the entire design space and reproduce the performance required to replicate the highly nonlinear database. To improve performance, we then utilize NNETs.

IV.I Review of Regression in Missiles

Before the regression methods and their applications are discussed in detail, it is important to mention previous works produced with regards to regression in which we are predicting some quantitative response such as time of flight or normal force coefficient. One of the first regression projects undertaken was the calculation of missile aerodynamic coefficients using NNETs from Ritz [86] who used AEROModeler, which at the time was a prototype software to implement NNETs. For the time, these results did well and surprisingly efforts were taken to develop model explanations using descriptor sensitivities. Next, NNETs were developed using SAS to predict the aerodynamic coefficients of missiles with grid fins [87]. These models use features such as angle of attack and grid fin length to predict output features such as drag coefficient and static margin. Results shows that each output could be predicted with R^2 all greater than 0.9999. Next, SAS was used again to develop NNETs to extend the work from [86] to include extra output features such as pitch-rate pitching-moment effectiveness $\left(C_{M_q}\right)$ and roll-rate effectiveness $\left(C_{L_p}\right)$ [88]. Results show extremely good fits using SAS. Next, SAS was used to develop linear regression models to develop thrust as a function of time and classification NNETs were used to predict how many stars points the grains have and resulted in 95% accuracy [89]. Finally, we arrive at the work done in this dissertation from Cervantes [90], which is work based off this dissertation. NNETs were developed using TensorFlow to predict time of flight, max thrust during flight, max distance travelled, max altitude, and max thrust at sea level. Results showed mean absolute percentage errors all than 0.5%. The work in this paper did not include the prediction of max weight, nor did it train long enough to further improve results. Results will show in the coming sections as to why NNETs are necessary and why enough training time is necessary.

IV.II Linear Regression Methods

To reiterate the basic idea of regression, a quantitative response is to be modeled using an input feature ([67],[68],[91]). Equation (4.1) shows the relationship between the output and the input, where $\left(f(X)\right)$ is an unknown function and $(\mathcal{E})$ is the random error from a normal distribution with a mean of zero. If the unknown function $\left(f(X)\right)$ is assumed to be a linear function, then $\left(f(X)\right)$ can be approximated using coefficients (β_0, β_1), where (β_0) is the intercept term and (β_1) is the slope. For example, we may be trying to model time of flight of the missile as function of launch angle and this model would be a simple linear regression model [92] shown on Eq. (4.2). The only thing we need to identify are the regression model coefficients (β_0, β_1). To solve for the coefficients, we first generalize Eq. (4.2) into Eq. (4.3). $\hat{\beta}_0$ is the estimated intercept and is the bias term, and the coefficients are combined into $\hat{\beta}$, which is vector of size $(P + 1)$ to include the bias term. The vector of input features, which are DBODY, ILAUNCH, etc., are $X = (X_1, X_2, \ldots, X_P)^T$ (where P equals 15 for this database) used to predict the output responses Y like TOF are modeled using Eq. (4.3). However, if we are modeling multiple outputs like TOF, MAXTHR, MAXDIST, APOGEE, THRSEA, and WEIGHT, then we can further generalize Eq. (4.3) into Eq. (4.4), which includes the K-vector of output responses Y to be modeled using the (PxK) matrix of coefficients $\hat{\beta}_K$.

$$Y = f(X) + \varepsilon$$
$$Y = \beta_0 + \beta_1 X + \varepsilon \tag{4.1}$$

$$TOF = \beta_0 + \beta_1 \cdot ILAUNCH \tag{4.2}$$

$$\hat{Y} = \hat{\beta}_0 + \sum_{j=1}^{P} X_j \hat{\beta}_j = X^T \hat{\beta} \tag{4.3}$$

$$\hat{Y}_K = X^T \hat{\beta}_K \tag{4.4}$$

To calculate the regression coefficients, we can fit the model to training data that was described in III.IV.I AULRC Regression Data. To train the model, the method of least squares can used to minimize the residual sum of squares [93], shown on Eq. (4.5). $RSS(\beta)$ is then rewritten to matrix notation on Eq. (4.6) where y is a N-vector of training outputs and X is an (NxP) matrix of inputs (each row refers to inputs like BDODY, ILAUNCH, etc.). Differentiating Eq. (4.5) with respect to β will allow estimating β which minimize Eq. (4.5) by setting Eq. (4.7) to zero gives the estimated coefficients $\hat{\beta}$ on Eq. (4.8). Since we are evaluating K=6 features there are again $\hat{\beta}_K$ $(15x6)$ matrix of estimated coefficients.

$$RSS(\beta)=\sum_{i=1}^{N}(y_i-x_i\beta)^2 \tag{4.5}$$

$$RSS(\beta)=(y-X\beta)^T(y-X\beta) \tag{4.6}$$

$$\frac{dRSS(\beta)}{d\beta}=\sum_{i=1}^{N}2\left(-x_i^T\right)(y_i-x_i\beta)=X^T(y-X\beta)=0 \tag{4.7}$$

$$\hat{\beta}=\left(X^TX\right)^{-1}X^Ty \tag{4.8}$$

Before implementation and results are shown for the linear regression method, it is important to discuss the bias-variance trade-off [92]. Every model is affected by bias, variance, and error ($\mathcal{E}$). When evaluating the test mean squared error (MSE) about a single sample (x_0), we can measure the variance of $\hat{Y}(x_0)$, the bias of $\hat{Y}(x_0)$, and the variance of the error ($\mathcal{E}$), shown on Eq. (4.9). $\text{var}(\mathcal{E})$ is the irreducible error, we then try to produce a model with low bias and low variance. Variance in a model refers to the amount the prediction of the model would change if it were trained using different data. A model with large variance would change drastically with small changes in the training data. Imagine one model with a specific set of training data, it produces a model with some output. New training samples are then added to the training and the model is refit, the model now predicts drastically different results; this is a model with large variance. Bias then refers to the error from modeling a highly nonlinear problem with a low order type model, such as linear regression. It may be that relationship between the responses and features are highly nonlinear, so no amount of training data can be used to produce an accurate model; thus, the model would have large bias. Typically, traditional linear regression methods suffer from large bias and low variance (underfitting).

$$E\left(y_0 - \hat{Y}\left(x_0\right)\right)^2 = \mathrm{var}\left[\hat{f}\left(x_0\right)\right] + \left[bias\left(\hat{f}\left(x_0\right)\right)\right]^2 + \mathrm{var}\left(\varepsilon\right) \tag{4.9}$$

Another issue we must guard against is overfitting to the training data, which is when the model has low bias and high variance. So, this is opposite to the previous statement just made. One easy way to generate overfitting is by increasing the number of parameters, which is useful to improve model performance. What happens is that certain parameters build a large coefficient value but are cancelled out by another parameter with a similar large negative coefficient. To reduce variance and ensure low bias, a penalty can be applied to the size of the coefficients. The first method is called "Ridge Regression" and works by shrinking the coefficients to be smaller and can approach zero. Instead of determining $\hat{\beta}_K$, we now determine the ridge coefficients $\left(\hat{\beta}_K^{Ridge}\right)$ such that they minimize a penalized residual sum of squares, shown on Eq. (4.10) where λ is the regularization strength or shrinkage term. Basically, larger λ makes shrinkage greater so coefficients are driven towards zero. Due to the scaling of the inputs, the inputs are typically standardized. Writing the arg-min portion of Eq. (4.10) in matrix form on Eq. (4.11) and taking the derivative with respect to β and setting equal to zero results in the estimated ridge coefficients $\left(\hat{\beta}_K^{Ridge}\right)$ on Eq. (4.12) where *I* is the (PxP) identity matrix.

$$\hat{\beta}^{Ridge} = \arg\min_{\beta}\left\{\sum_{i=1}^{N}\left(y_i - \beta_0 - \sum_{j=1}^{P} x_{ij}\beta_j\right)^2 + \lambda\sum_{j=1}^{P}\beta_j^2\right\} \tag{4.10}$$

$$RSS(\lambda) = (y - X\beta)^T (y - X\beta) + \lambda\beta^T \beta \tag{4.11}$$

$$\frac{dRSS(\lambda)}{d\beta} = -2\sum_{i=1}^{N}(x_{ij})\left(y_i - \sum_{j=1}^{P} x_{ij}\beta_j\right) + 2\lambda\sum_{j=1}^{P}\beta_j = 0$$

$$\frac{dRSS(\lambda)}{d\beta} = -X^T(y - X\beta) + \lambda\beta = -X^T y + X^T X\beta + \lambda\beta$$

$$X^T y = X^T X\beta + \lambda\beta = \left(X^T X + \lambda I\right)\beta \tag{4.12}$$

$$\hat{\beta}^{Ridge} = \left(X^T X + \lambda I\right)^{-1} X^T y$$

Another widely used regularization method used is "Lasso Regression". Lasso instead makes coefficients exactly zero, whereas ridge shrinks towards zero. The penalty function in ridge is considered L_2, however, the penalty applied to lasso is L_1 which makes the solution nonlinear and so there is no general closed form solution. Programmatically, the lasso coefficients $\left(\hat{\beta}^{Lasso}\right)$ can be found using Eq. (4.13). Similar to ridge, the inputs should also be standardized as well. Both ridge and lasso are used to feature selection to reduce the number of parameters in the model. Generally, parameters with coefficients near zero are removed and the model is refit using reduced data.

$$\hat{\beta}^{Lasso} = \arg\min_{\beta}\left\{\frac{1}{2}\sum_{i=1}^{N}\left(y_i - \beta_0 - \sum_{j=1}^{P} x_{ij}\beta_j\right)^2 + \lambda\sum_{j=1}^{P}\left|\beta_j\right|\right\} \tag{4.13}$$

To implement linear, ridge, and lasso regression models, we use can use the Python package called Scikit-Learn [2] to call the regression methods above using "sklearn.linear_model" functions “LinearRegression”, “RidgeCV”, and “MultiTaskLassoCV”. “LinearRegression” uses the ordinary least squares regression to solve for coefficients. “RidgeCV” is a cross-validation implementation of ridge to calculate the optimal λ and automatically fits using the optimal λ. Similarly, “MultiTaskLassoCV” is also a cross-validation implementation of lasso to calculate and fit with the optimal λ.

IV.II.I Linear Regression Methodology

The implementation of the Python script for the linear regression is as follows and a very good reference that this initial was based on is from Geron [94], who outlines methodologies for using Scikit-Learn in great detail. It is highly recommended that users with little python experience start with Geron [94]. An extra reference for developing a lot of the residuals and figures were developed from Bruce [95].

1. Read in data using Pandas “read_csv” function to read data.
2. Use Pandas to separate data into input data and output data.
3. Use Sklearn “train_test_split” to randomly split data into training and testing data.
4. Develop pipeline to use “PolynomialFeatures” to add higher order terms and standardize the data using “StandardScaler”.
5. Use the pipeline to transform the data.
6. Train the models using “fit”.
7. Predict the output data using the test data.
8. Calculate performance of models such as R^2, MAPE, MSE, and WAPE.

The data is first read in and split into the input and output data. It is recommended that data is split into training and testing data, so that the model does not overfit to the training data and do poorly on the test data or new data. We will also evaluate higher order models of linear regression which utilize higher terms like (DBODY*ILAUNCH, DBODY*THROAT*ILAUNCH, etc.). The data is standardized then trained. The performance of each of the models is then obtained. First, the R^2 statistic is shown on Eq. (4.14) and is a proportion of the variance explained by the model and is typically between 0-1, however if it is negative then a model which draws a line through the mean value of the data would have a better fit ([92], [93]). R^2 is a measure of the proportion of variability in the output that is explained by the inputs. Another way of thinking about R^2 is by saying the amount of variability that is explained by the regression model $(TSS - RSS)$. One issue with R^2 is that it can be challenging to assess what value of R^2 is acceptable.

$$\overline{y} = \frac{1}{N}\sum_{i=1}^{N} y_i$$

$$R^2 = \frac{TSS - RSS}{TSS} = 1 - \frac{RSS}{TSS} = 1 - \frac{\sum_{i=1}^{N}\left(y_i - f\left(x_i\right)\right)^2}{\sum_{i=1}^{N}\left(y_i - \overline{y}\right)^2} \tag{4.14}$$

One of the most popular methods for estimating model quality is through the use of mean squared error (MSE), shown on Eq. (4.15). The closer the predicted responses $\left(f(x_i)\right)$ are to the true values (y_i), the lower MSE will be. A lower is MSE is desired in a model output. One issue with MSE is that the outputs is in squared units, so if the output measure were TOF then the MSE would be $(second^2)$ which is an odd way to measure the error.

$$MSE = \frac{1}{N}\sum_{i=1}^{N}\left(y_i - f\left(x_i\right)\right)^2 \tag{4.15}$$

Often a percentage error is preferred and so the mean absolute percentage error (MAPE) is used [96]. It is a proportion of the residual error to the actual value so in a sense is the relative error of the model. MAPE is much easier to understand because it is percentage based. A negative of MAPE is that when true values are near zero, MAPE will become very large and can approach infinity. MAPE is also biased towards negative values and MAPE places higher penalty on negative values [97].

$$MAPE = \frac{100}{N} \frac{\sum_{i=1}^{N} \left| y_i - f\left(x_i\right) \right|}{y_i} \tag{4.16}$$

To deal with the issues of MAPE, a modified version of MAPE called MMAPE (Modified Mean Absolute Percentage Error) is also used to calculate performance, which is shown on Eq. (4.17), and can see that instead of dividing by the true value (y_i), the residual error is divided by the difference between the maximum and minimum of *Y*. If values are near zero then the error will not approach infinity and errors only grow large when the residual is great. An issue with MMAPE is that errors which are significant for a specific sample may get reduced too low. Caution should be taken when interpreting MMAPE due to the scaling of the residual error using the range because they can vastly differ. For example, the ranges for the outputs are 119.59 Lbf/1000 for THRSEA, 132.93 Lbf/1000 for MAXTHR, 6660.47 ft/1000 for MAXDIST, 3568.3 ft/1000 for APOGEE, 1086.79 seconds for TOF, and 45.88 Lbm/1000 for WEIGHT.

$$MMAPE = \frac{100}{N} \frac{\sum_{i=1}^{N} \left| y_i - f\left(x_i\right) \right|}{\max\left(Y\right) - \min\left(Y\right)} \tag{4.17}$$

IV.II.II Linear Regression: One-Way Results

When splitting the data, the training data contains 400,000 samples and the testing data contains 100,000 samples. The models are then fit using degree of one, so there are only one term parameters modeled (DBODY, ILAUNCH, etc.), and the overall metrics are calculated and shown for all three models on Table 11. Because this model only uses one-way terms in the model there are 15 coefficients to model. The regularizations strength for lasso was determined to be 0.1 which adds a small amount of error, and ridge allows regularization strength for each output are [0.8, 0.4, 5.3, 1.1, 0.4, 0.9] for TOF, MAXTHR, MAXDIST, APOGEE, THRSEA, & WEIGHT, respectively. Can see that based on R^2 the models are all equivalent and only differ when evaluating MSE and MAPE. Linear model seems to perform best when evaluating MSE, Lasso does best when evaluating MAPE and MMAPE. If we assess the individual output metrics, we can then learn how well the model is performing for each individual output. Table 12 shows the R^2 and see that all models do about the same. However, when assessing the individual outputs, can see that none of the models can highly replicate MAXDIST, next is APOGEE, TOF, WEIGHT, THRSEA, and finally MAXTHR seems to be the most replicable. To reassure the previous statements, MSE is shown for each output on Table 13. Again, each model has approximately the same MSE for each output. The model metrics are listed in order from highest MSE to least MSE: MAXDIST, APOGEE, TOF, MAXTHR, THRSEA, and WEIGHT. In general, will see that MAXTHR, THRSEA, and WEIGHT will have the lowest errors and APOGEE & MAXDIST will have the most error. TOF will typically have the mid-range error. With R^2 and MSE it is hard to understand what the error looks like, so we need to look at percentage error.

Next, we assess MAPE on Table 14 and each model does about the same overall; except now we can gauge how bad the model is performing. Note the APOGEE has 183% error, which is eye opening to see how bad the model is reproducing APOGEE; MAXDIST has 88% error and suggests that the model is not reproducing MAXDIST at all. TOF also a high pretty high error as well with 20% and shows that on average the model will have 20% relative error to the true value. MAXTHR, THRSEA, & WEIGHT have about 8-11% relative error and show that they are well reproduced. Finally, looking at MMAPE on Table 15, all the models show the same performance. However, notice that the percent errors show less than 5% for each output. Caution should be taken when using MMAPE because the residual error is scaled by the range, so it is a percentage of the range. So MAXDIST has largest error of 4.7% meaning that the residual error is 4.7% relative to the range of MAXDIST, which is approximately 6660 ft/1000 meaning MAXDIST could be off by about 313,020 feet assuming 4.7% error and would be a very large error. Even the 2.7 % MMAPE for WEIGHT suggests that the result could be off by 1240 lbm when making predictions. Even for the small error with MMAPE, there is still a very large residual error.

Table 11 Overall Model Performance for Degree =1

Model	R^2	MSE	MAPE	MMAPE
Linear	0.8581	35918.4609	53.3316	3.1827
Ridge	0.8581	35918.975	53.3311	3.1827
Lasso	0.8581	35918.0508	53.3036	3.1825

Table 12 R^2 for Degree = 1

Model	TOF	MAXTHR	MAXDIST	APOGEE	THRSEA	WEIGHT
Linear	0.8825	0.9463	0.6454	0.8023	0.9404	0.9319
Ridge	0.8825	0.9463	0.6454	0.8023	0.9404	0.9319
Lasso	0.8825	0.9463	0.6454	0.8023	0.9404	0.9319

Table 13 MSE for Degree = 1

Model	TOF	MAXTHR	MAXDIST	APOGEE	THRSEA	WEIGHT
Linear	2624.9968	22.673	192767.5625	20072.6484	20.067	2.8277
Ridge	2625.0365	22.6734	192770.2539	20072.9912	20.0673	2.8277
Lasso	2624.887	22.6728	192765.5156	20072.332	20.067	2.8277

Table 14 MAPE for Degree = 1

Model	TOF	MAXTHR	MAXDIST	APOGEE	THRSEA	WEIGHT
Linear	19.6549	9.8245	88.127	183.4058	10.4442	8.5334
Ridge	19.655	9.8245	88.1262	183.4037	10.4441	8.5334
Lasso	19.6599	9.8183	88.1071	183.2694	10.4381	8.5288

Table 15 MMAPE for Degree = 1

Model	TOF	MAXTHR	MAXDIST	APOGEE	THRSEA	WEIGHT
Linear	3.5573	2.6389	4.6893	2.7812	2.7501	2.6795
Ridge	3.5573	2.6389	4.6893	2.7812	2.7501	2.6795
Lasso	3.5576	2.6385	4.6892	2.7809	2.7497	2.6791

Figure 37 shows the predicted versus actual plots to compare the predicted output using the linear model to the actual test data. If the model was perfectly replicating the data, the seaborn "regplot" would show the predicted versus actual data as a linear line and would follow the black line on the figure. However, can see that predicted versus actual data for TOF that the model partially fits the data and there is still wide dispersion of error. Can even see that for earlier TOF, the model is predicted negative TOF, which is substantially different from actual TOF (over 500 seconds), then the model is greatly underpredicting and apparently there is a portion of data that is not getting accurately trained by the model. When looking at MAXTHR, can see that the model is much more accurately predicting the actual data, at least from 25-75 lbf/1000. The model is having trouble predicting near the minimum and is predicting negative value, which is very wrong, and the model is underpredicting near the maximum thrust values. Both outputs suggest that a higher order model is necessary to improve model error.

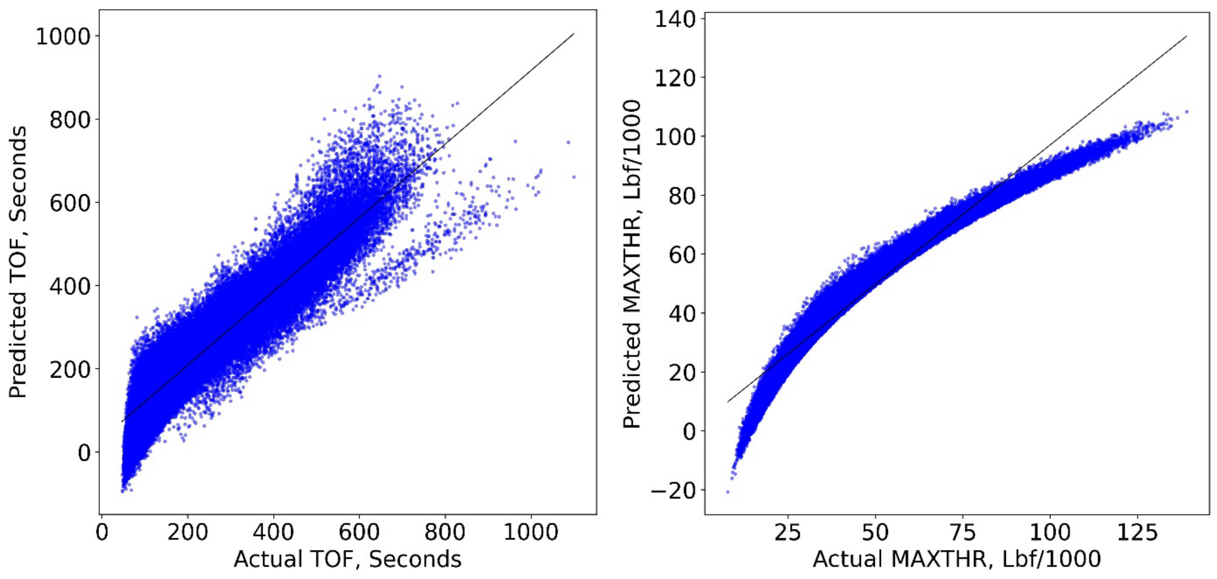


Fig. 36 Linear One-Way Models: Predicted vs. Actual for TOF (Left) & MAXTHR (Right).

Figure 38 shows the predicted versus actual plot for MAXDIST & APOGEE. We see a similar trend on both outputs where the model is not accurately predicting the output. The model is vastly underpredicting near the minimum values, MAXDIST is also greatly overpredicting near the minimum value. Both models seem to have a sub-portion of data that is not being modeled correctly as the output value increases. The linear model is obviously capable of replicating the data and a higher order model is needed.

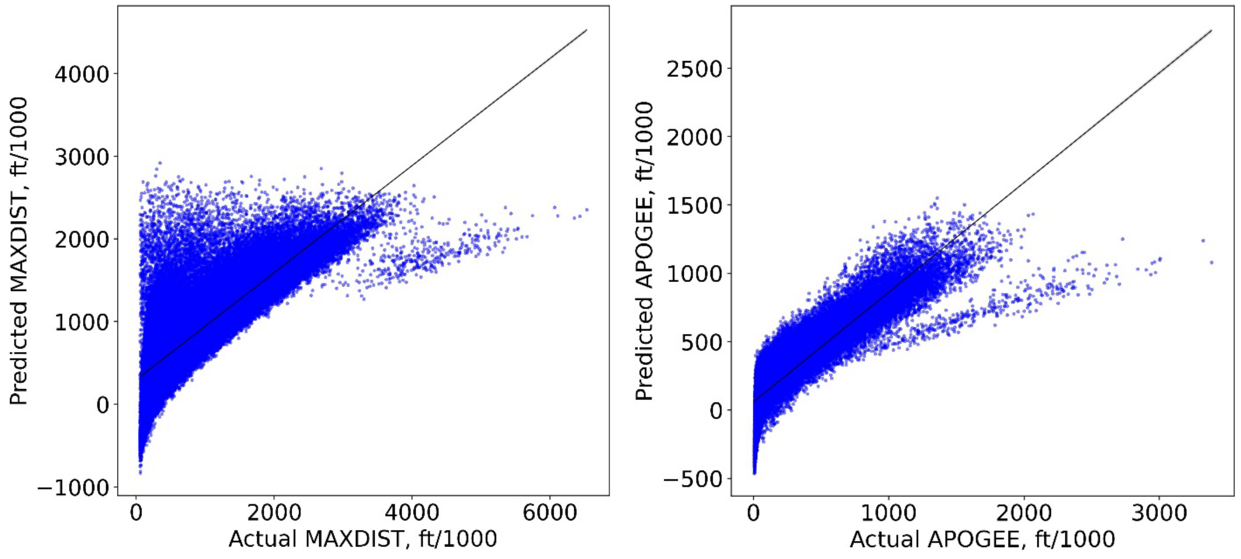


Fig. 37 Linear One-Way Models: Predicted vs. Actual for MAXDIST (Left) & APOGEE (Right).

Figure 39 shows the predicted versus actual plot for THRSEA & WEIGHT. The models are replicating the data similar to that of MAXTHR on Fig. 37, where only the center region is being modeled correctly and the minimum and maximum areas are being under predicted. Both of these outputs suggest that there is more nonlinear relationship not being captured by the one-way parameters and that a higher order model is needed.

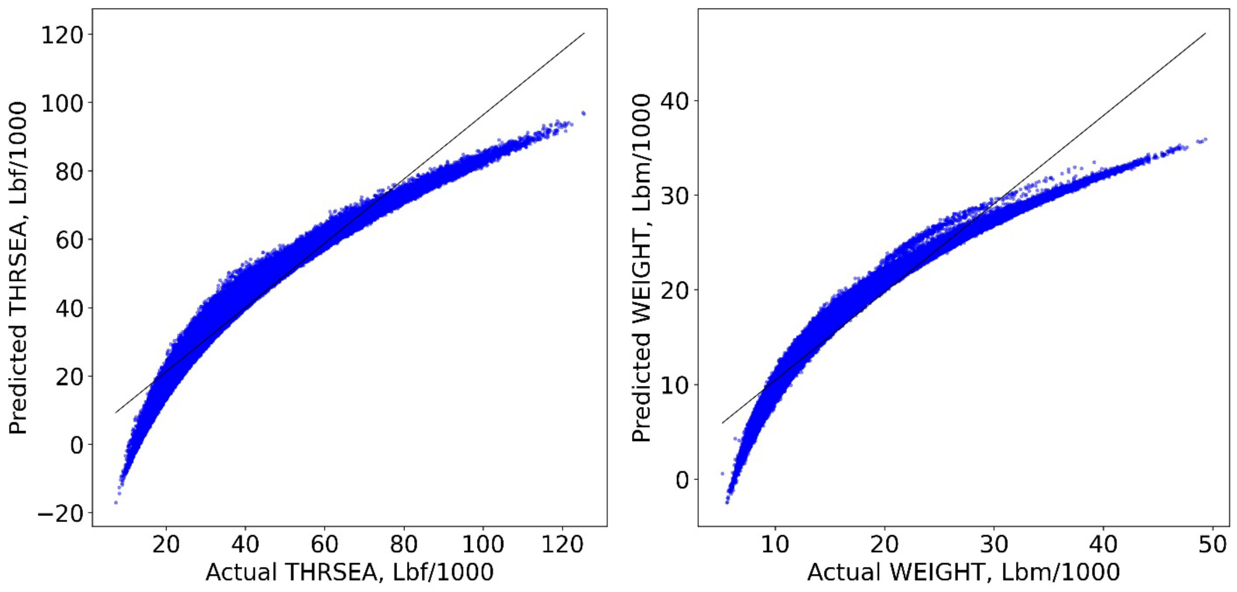


Fig. 38 Linear One-Way Models: Predicted vs. Actual for THRSEA (Left) & WEIGHT (Right).

IV.II.III Linear Regression: Two-Way Results

To make the model a higher order model using linear regression, the "PolynomialFeatures" parameter "degree" is set to two and will includes multiplicative parameters like on Eq. (4.18). We include terms of higher degree such as $DBODY^2$ to ensure that the model is accurately capturing nonlinear relationships in the data. It may be that part of the output in actuality may be linearly modeled with a parameter say $DBODY$, but another portion of the output is actually nonlinearly modeled with $DBODY^2$. Regardless of including nondistinctive parameters, it will be shown that the linear regression models even up to 3rd order are not capable of replicating every output. Because the two-way and one-way parameters are included, there are a total of 230 coefficients to obtain.

$$
\begin{aligned}
TOF &= \beta_0 + \beta_1 DBODY + \beta_2 ILAUNCH + \cdots \\
&\beta_3 DBODY^2 + \beta_4 DBODY \cdot ILAUNCH + \beta_5 ILAUNCH^2
\end{aligned}
\tag{4.18}
$$

This data used in this model is the same as that was used in the previous model in section IV.II.II, which used one-way parameters, so it has 400,000 training samples and 100,000 testing samples as well. The regularizations strength for lasso was set to 0.1 because lasso will not converge with too many parameters, and ridge allows regularization strength for each output are [0.1, 0.1, 0.1, 0.1, 0.1, 0.1] for TOF, MAXTHR, MAXDIST, APOGEE, THRSEA, & WEIGHT, respectively. Can immediately see from Table 16 that increasing the polynomial order to two that the performance increases and notice that the linear and ridge models are quite similar in metrics and are outperforming lasso. See that R^2 increases from 0.85 to 0.96, so overall the model is performing better and is verified by the decrease in MSE from 36k to 13.6k but MSE is still not near zero and should be. There is still considerable error with MAPE, even though it is halved, and MMAPE is also halved. Table 17 shows the individual R^2 values for each output and can see the model improvements. MAXTHR, THRSEA, and WEIGHT have R^2 all greater than 0.99 which suggests that the model is fitting extremely well. Can see that APOGEE & TOF have greatly increased as well but did not increase as high so there is still some error. MAXDIST did increase from 0.65 but not as much as the other parameters so there is obviously error in the regression model. Table 18 shows the MSE and we can see that MAXTHR, THRSEA, and WEIGHT have greatly reduced in error and no need to say there doing well with the other parameter. TOF also has very low error but not near zero yet and APOGEE & MAXDIST still have quite large errors. Table 19 shows individual output MAPE values and can see TOF has less than 10% relative error which is quite good, and MAXDIST & APOGEE are obviously not replicated well. Table 20 shows the MMAPE and shows low error for every parameter. Nevertheless, it should be recognized that the scaling of the outputs shows that MAXDIST can be off by approximately 180,000 feet. MAXTHR, THRSEA, and WEIGHT are modeled well using the linear regression models, but modeling TOF, MAXDIST, & APOGEE requires a higher order model.

Table 16 Overall Model Performance for Degree = 2

Model	R^2	MSE	MAPE	MMAPE
Linear	0.9598	13671.4414	24.0601	1.1556
Ridge	0.9598	13672.0328	24.0445	1.1554
Lasso	0.9197	22442.1855	33.4708	2.2206

Table 17 R^2 for Degree = 2

Model	TOF	MAXTHR	MAXDIST	APOGEE	THRSEA	WEIGHT
Linear	0.9682	0.9982	0.8628	0.934	0.9985	0.997
Ridge	0.9682	0.9982	0.8628	0.934	0.9985	0.997
Lasso	0.9415	0.9751	0.7736	0.8988	0.9715	0.9578

Table 18 MSE for Degree = 2

Model	TOF	MAXTHR	MAXDIST	APOGEE	THRSEA	WEIGHT
Linear	709.9409	0.7759	74613.3672	6703.9424	0.4911	0.1264
Ridge	709.9576	0.7759	74616.8835	6703.9625	0.4911	0.1264
Lasso	1307.1835	10.5091	123054.3359	10269.7432	9.5782	1.7513

Table 19 MAPE for Degree = 2

Model	TOF	MAXTHR	MAXDIST	APOGEE	THRSEA	WEIGHT
Linear	8.5808	1.7767	53.2036	77.7362	1.5718	1.4915
Ridge	8.5789	1.7765	53.112	77.7368	1.5714	1.4912
Lasso	12.6237	6.4154	51.0828	117.4359	6.9218	6.345

Table 20 MMAPE for Degree = 2

Model	TOF	MAXTHR	MAXDIST	APOGEE	THRSEA	WEIGHT
Linear	1.546	0.4783	2.6869	1.3608	0.3976	0.4638
Ridge	1.5458	0.4783	2.686	1.3607	0.3976	0.4638
Lasso	2.3214	1.7738	3.4338	1.8393	1.8849	2.0702

Figures 40-42 show the predicted versus actual and can see how well the model is replicating MAXTHR, THRSEA, & WEIGHT. MAXTHR & THRSEA do suffer a bit at the maximum end and can see the model slight underpredicting. WEIGHT is slightly underpredicting at the maximum end but appears to have another distribution being modeled in the mid-range which suggests a higher order model or nonlinear model is required to capture the effects of this distribution. It is obvious to see that there are two distributions that cannot be captured by the model and is far worse on MAXDIST because there is also a higher density of samples on the minimum. A higher order model may improve performance; however, there is apparently a second distribution that cannot be accurately modeled using linear regression so increasing the model order may improve performance of the other parameters but will still not capture the second distribution. See Appendix L: 3rd Order Linear Regression Model for some performance metrics and predicted versus actual plots using the linear model only because it proves that the linear regression model cannot capture the second distribution.

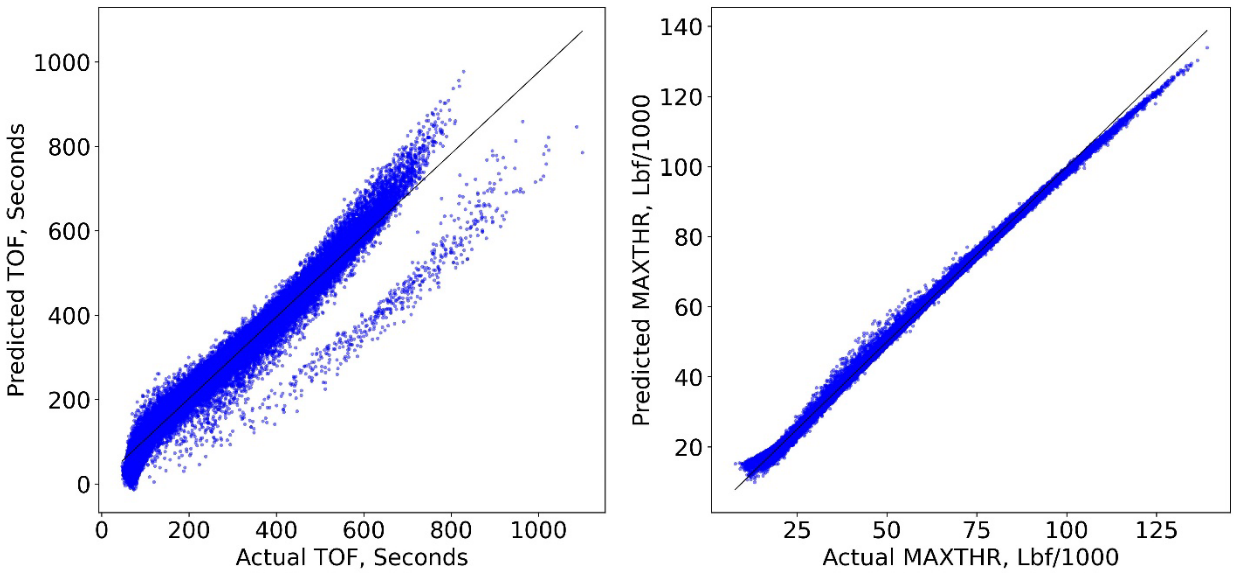


Fig. 39 Linear Two-Way Models: Predicted vs. Actual for TOF (Left) & MAXTHR (Right).

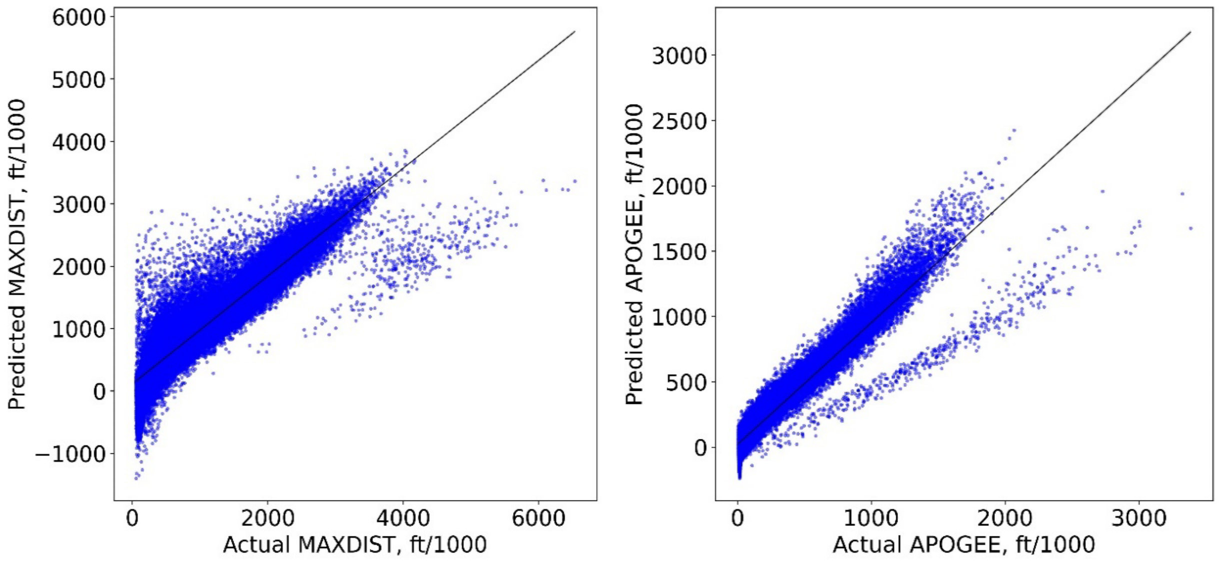
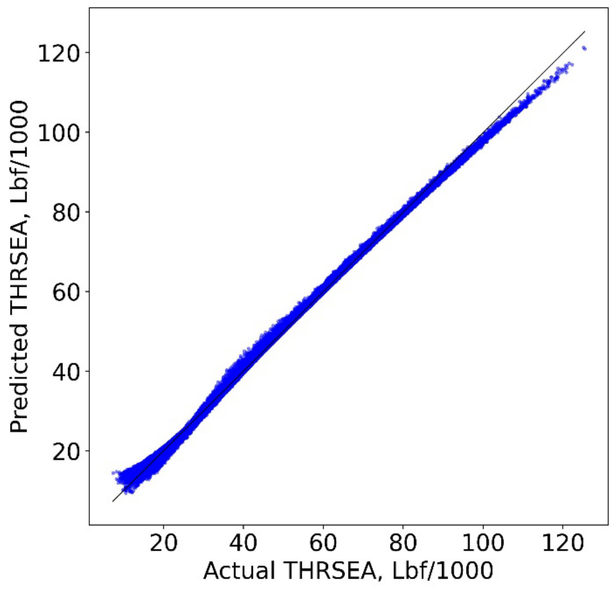


Fig. 40 Linear Two-Way Models: Predicted vs. Actual for MAXDIST (Left) & APOGEE (Right).



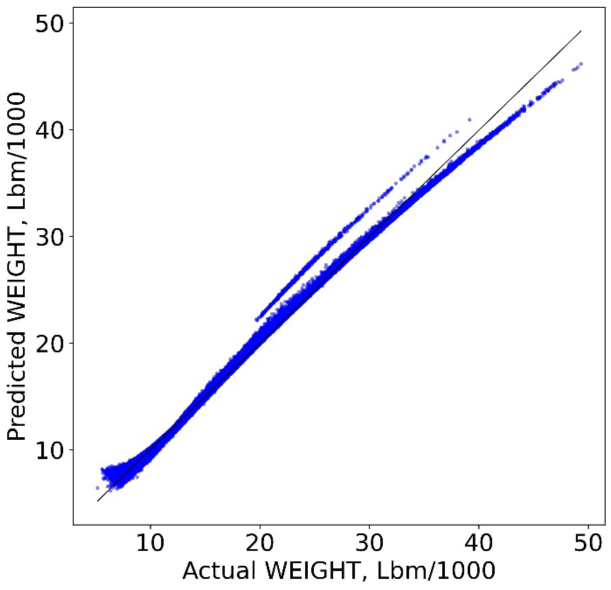


Fig. 41 Linear Two-Way Models: Predicted vs. Actual for THRSEA (Left) & WEIGHT (Right).

IV.III Neural Networks: Regression

In the previous section, linear regression methods were developed, and results shown that none of the methods were fully capable of replicating the data. A higher degree regression model is only going to slightly increase performance, but never fully replicate the data due to the two distributions of data. Therefore, to increase performance a nonlinear model is needed to capture the two distributions. There are multiple nonlinear methods and include nonlinear regression [91], nearest neighbors [93], decision trees [92][93], ensemble methods [92][93], and NNETs [92-94][98-103]. Before, NNETs are discussed in mathematical terms, a brief timeline of NNETs is presented to show how NNETs came to such prominence today. It should be noted that NNETs are not new, in fact they first came to light in 1943 from McCulloch and Pitts [104] developing a computational framework for biological neurons using logic functions such as AND, OR, and NOT

[94]. These logic functions work on the premise of whether inputs in the neuron are active or not. Frank Rosenblatt then developed the perceptron in 1958 [105], which takes a sum of the inputs and weights, and a function is applied that would output zero if the value was less than a threshold value and would output one if greater than the threshold value. Figure 43 shows the perceptron and can see the layout of how a perceptron works. The perceptron sum is a linear regression formula $z = X_1W_1 + X_2W_2 + \cdots + X_PW_P$ where W are the weights to be determined like the coefficients in regression, a threshold function (ϕ) is then applied to the perceptron sum to output zero or one based on a threshold value like on Eq. (4.19) and is essentially a step function.

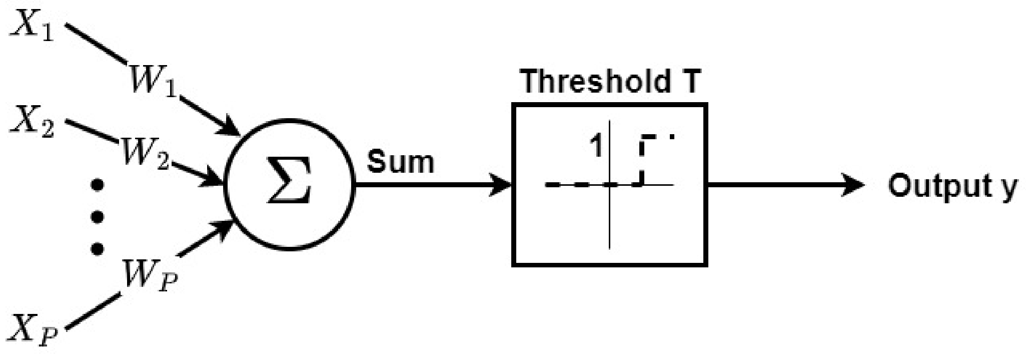


Fig. 42 Threshold Logic Unit Perceptron.

$$\phi(z) = \begin{cases} 0 & \text{if } z < threshold \\ 1 & \text{if } z \geq threshold \end{cases} \tag{4.19}$$

If we then combine multiple perceptrons to generate a fully connected layer the output can be generalized using basic linear algebra on Eq. (4.20). X is a matrix of inputs with each row being the sample and each column is a feature, say X_1. W is a matrix of all the weights where each row refers to weights for input neuron and each column refers to an artificial neuron, say z_1. The bias vector $\boldsymbol{b}$ contains all the weights for a bias neuron and has one bias per artificial neuron z. Output is then the input matrix multiplied by the weight matrix plus the bias matrix and then the function (ϕ), which was referred as "threshold", which is the activation function, and for perceptron learning was just the step function.

$$h(X) = \phi(XW + b) \tag{4.20}$$

NNETs are fundamentally similar to regression methods in that an algorithm is required for obtaining the weights. Rosenblatt adapted the training algorithm from Hebbian learning, which famously states that weights increase when two neurons fire simultaneously [106]. Perceptrons use an adaptation of Hebb's rule and accounts for error in the output. Equation (4.21) is the perceptron learning rule used to update the weights, where $w_{i,j}$ is the weight between the i^{th} input neuron and the j^{th} output neuron, η is the learning rate, y_j is the actual output of the j^{th} output neuron for the training sample, $\hat{y}_j$ is the predicted output of the j^{th} output neuron, and x_i is the i^{th} input of the training sample. The *Perceptron convergence theorem* states that if the training samples are linearly separable then the solution will converge.

$$w_{i,j}^{(\text{Next Step})} = w_{i,j} + \eta\left(y_i - \hat{y}_j\right)x_i \tag{4.21}$$

There are limitations to perceptron learning, which if the data is too complex then perceptron learning will not be able to converge. More advanced perceptron's can be developed by add more perceptron layers (Multilayer Perceptron – MLP), which can then solve more complicated problems. A stronger learning algorithm is then needed to update the weights and in 1985 Rumelhart, Hinton, & Williams [107] adapted the backpropagation learning algorithm (developed in 60's & 70's) to train weights in the MLP. Backpropagation is actually just gradient descent which computes the error gradients with respect to the weights $\left(\frac{\partial \mathrm{E}}{\partial \mathrm{w}}\right)$ and will be shown to be similar to Eq. (4.21).

IV.III.I Feedforward NNETs & Backpropagation

If there are multiple layers, the error gradient can be quite deceptive to calculate, ref.'s [92-94] & [98-103] all provide some discussion on feedforward NNETs and backpropagation. A Feedforward network from [93] is shown here for the NNET shown on Fig. 44 which uses a single hidden layer. The neurons in the hidden layer are referred to as hidden units from now on. Let X $\in \mathcal{R}^n$ denote a real values random input vector of n independent variables, i.e., the inputs such as DBODY, and Y $\in \mathcal{R}^K$ denote a real valued random output vector of *K* dependent variables, i.e., the outputs such as TOF. The *Z* terms are the sum of the weight and inputs multiplied, shown on Eq. (4.22), where $W_{1,1}^1$ refers to weight times X_1 in the first hidden layer and $W_{B,1}^1$ is the bias vector

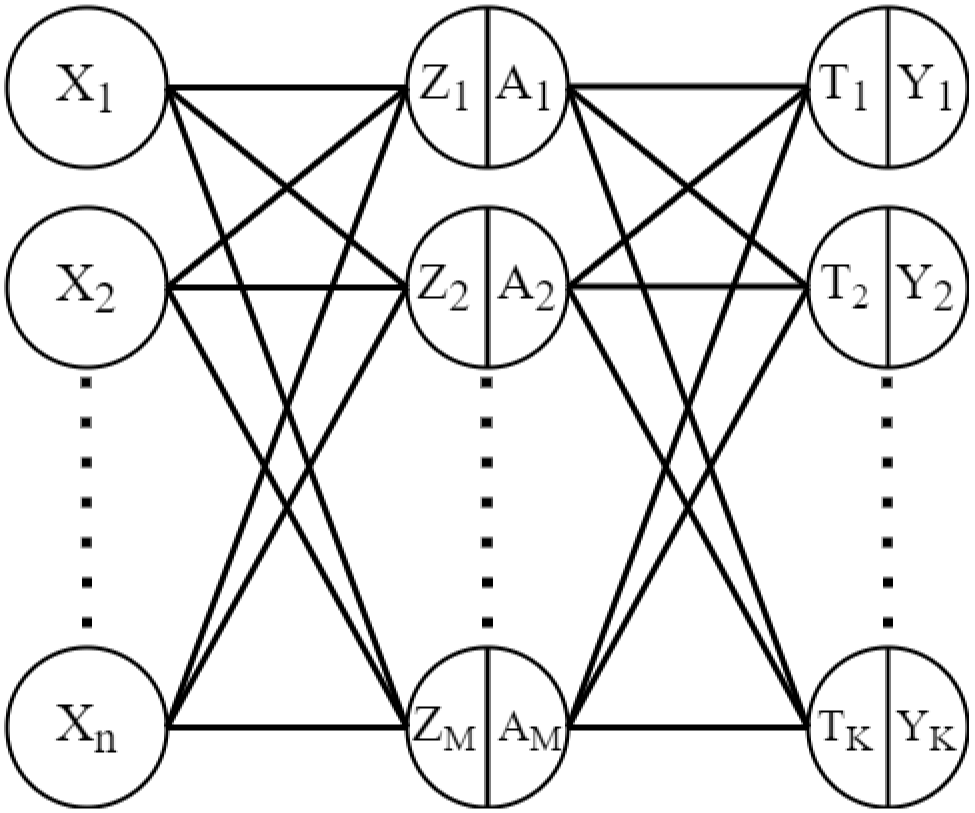


Fig. 43 NNET Representation.

for hidden unit 1 in hidden layer 1. Z derived features are then activated using function (σ), e.g., sigmoid function, shown on Eq. (4.23) to develop A activations for each hidden unit. The T terms are just like the Z terms in that they are the sum of weights in hidden layer 2 times the activations values A from hidden layer 1 and is displayed on Eq. (4.24), where $W_{1,1}^{2}$ refers to weight times A_1 in the first hidden layer for output unit 1in the output layer and $W_{B,1}^{2}$ is the bias vector for output unit 1 in the output layer. Because this NNET is used for regression analysis, $Y_K = T_K$ with no activation applied to T_K.

$$Z_m = W_{1,m}^{1} X_1 + \cdots + W_{n,m}^{1} X_n + W_{B,m}^{1} \ , \ m = 1, 2, \ldots, M \tag{4.22}$$

$$A_m = \sigma\left(Z_m\right) = \frac{1}{1 + e^{-Z_m}} \ , \ \ -\infty < Z_m < \infty \tag{4.23}$$

$$T_k = W_{1,k}^{2} A_1 + \cdots W_{M,k}^{2} A_M + W_{B,k}^{2} \ \ , \ k = 1, 2, \ldots, K \tag{4.24}$$

To fit the NNET there $M(n+1)$ weights (includes bias) to calculate for hidden layer 1 and $K(M+1)$ weights (includes bias) to determine for the output layer; so, there are a total of $\mathrm{M}(n+1) + K(M+1)$ weights to determine for the NNET. A learning algorithm similar to Eq. (4.21), called backpropagation is used to determine the error gradient to update the weights instead of using just the residual $(y - \hat{y})$. Equation (4.25) is the sum of squared errors such that the weights are minimized, similar to Eq. (4.5), and could be a different function such as mean squared error, mean absolute error, or mean absolute percentage error.

$$R(W) = \sum_{i=1}^{N} R_i = \sum_{i=1}^{N} \sum_{k=1}^{K} \left(y_{ik} - Y_k\left(x_i\right)\right)^2 \tag{4.25}$$

Equation (4.26) then shows how to use Eq. (4.25) to calculate the error gradient with respect to a weight for an output unit. Equation (4.27) shows how the error gradient reduces to calculate the bias weight for an output unit.

$$\begin{aligned}\frac{\partial R_i}{\partial W_{m,k}^2} &= \frac{\partial R_i}{\partial Y_k(x_i)}\frac{\partial Y_k(x_i)}{\partial T_k}\frac{\partial T_k}{\partial W_{m,k}^2} = \frac{\partial R_i}{\partial Y_k(x_i)}\frac{\partial Y_k(x_i)}{\partial T_k}A_m \\ \frac{\partial R_i}{\partial W_{m,k}^2} &= -2\left(y_{ik} - Y_k(x_i)\right)(1)\sigma(Z_m)\end{aligned} \tag{4.26}$$

$$\begin{aligned}\frac{\partial R_i}{\partial W_{B,k}^2} &= \frac{\partial R_i}{\partial Y_k(x_i)}\frac{\partial Y_k(x_i)}{\partial T_k}\frac{\partial T_k}{\partial W_{B,k}^2} = \frac{\partial R_i}{\partial Y_k(x_i)}\frac{\partial Y_k(x_i)}{\partial T_k}(1) \\ \frac{\partial R_i}{\partial W_{B,k}^2} &= -2\left(y_{ik} - Y_k(x_i)\right)(1)(1)\end{aligned} \tag{4.27}$$

Equation (4.28) then shows how to calculate the weights for the hidden layer and can see now that it includes the summation of error gradients of each output unit, significantly making the total error gradient more complex. A bias weight for a hidden unit is also shown for brevity on Eq. (4.29).

$$\begin{aligned}\frac{\partial R_i}{\partial W_{n,m}^1} &= \sum_{k=1}^{K}\frac{\partial R_i}{\partial Y_k(x_i)}\frac{\partial Y_k(x_i)}{\partial T_k}\frac{\partial T_k}{\partial A_m}\frac{\partial A_m}{\partial Z_m}\frac{\partial Z_m}{\partial W_{n,m}^1} \\ \frac{\partial R_i}{\partial W_{n,m}^1} &= \sum_{k=1}^{K}-2\left(y_{ik} - Y_k(x_i)\right)(1)W_{m,k}^2\sigma'(Z_m)X_{i,n}\end{aligned} \tag{4.28}$$

$$\frac{\partial R_i}{\partial W_{B,m}^1} = \sum_{k=1}^{K} \frac{\partial R_i}{\partial Y_k(x_i)} \frac{\partial Y_k(x_i)}{\partial T_k} \frac{\partial T_k}{\partial A_m} \frac{\partial A_m}{\partial Z_m} \frac{\partial Z_m}{\partial W_{B,m}^1}$$

$$\frac{\partial R_i}{\partial W_{B,m}^1} = \sum_{k=1}^{K} -2\left(y_{ik} - Y_k(x_i)\right)(1) W_{m,k}^2 \sigma'(Z_m)(1) \tag{4.29}$$

The error gradients from Eq.'s (4.26) and (4.28) can now be used in the gradient descent to update the $(j+1)$ weights for the both the hidden units and output units, shown on Eq. (4.30) where (γ) is the learning rate. This form of the gradient descent calculates the summation of the error gradients for the entire training set and is known as batch gradient descent (BGD). The main issues with batch gradient descent are that the quadratic error surface may be extremely complex, and the algorithm may get stuck in a local minimum [101]. Other concerns have to do with the amount of training time, since the training set can be massive it can take long computational times to calculate the entire dataset error gradient [94].

$$W_{m,k}^{2(j+1)} = W_{m,k}^{2(j)} - \gamma \sum_{i=1}^{N} \frac{\partial R_i}{\partial W_{m,k}^{2(j)}}$$

$$W_{n,m}^{1(j+1)} = W_{n,m}^{1(j)} - \gamma \sum_{i=1}^{N} \frac{\partial R_i}{\partial W_{n,m}^{1(j)}} \tag{4.30}$$

On the opposite spectrum of gradient descent is stochastic gradient descent (SGD) which only uses one sample $(N = 1)$ to approximate the error. Issues with SGD are that due to the approximation of error, the solution will jump randomly around the solution, but it may never reach the global minimum. Despite this randomness, it is usually more likely to find the global minimum compared to batch gradient descent. Similar to batch gradient descent, solely due to the amount of randomness the stochastic method can also take a very long time to converge; however, the stochastic method uses much less memory compared to batch gradient descent since the calculations do not require every training sample [94].

In the middle of SGD and batch gradient descent is mini-batch gradient descent (MBGD), which uses a random subset of training samples. There is less randomness in this algorithm so it should not jump around the global minimum too much. Similar to BGD, MBGD could have difficulty moving out of the local minimum. Typically, batch gradient descent takes the longest to train due to training size, followed by MBGD, and SGD is usually fastest; however, SGD could take much longer if there is too much randomness [94]. The NNETs developed in this work all use MBGD as it seems to provide the best compromise of SGD and batch gradient descent. For clarity with MBGD, each batch is fed through the network then the weights are updated, so if there are 100 batches in an epoch then there are 100 updates to the weights per epoch. For SGD, if there are 1000 samples, then there are 1000 weight updates per epoch. For BGD, then there is only one weight update per epoch.

IV.III.II Optimizer: Momentum, NAG, RMSProp, Adam, & Nadam

The backpropagation algorithm was defined using batch gradient descent because it was the original algorithm which was used to converge MLPs. NNETs have come a very long way and more advanced algorithms have come to fruition. When discussing machine learning, the backpropagation using gradient descent is commonly known as an optimizer called "SGD" and was modified to include a momentum term for speedup [108] at least in TensorFlow (TF). If we generalize the weights from Eq. (4.30) into (θ), we can develop a momentum vector (m) where (η) is the learning rate and $\left(\nabla_{\theta}J(\theta)\right)$ is the gradient of the cost function $\left(J(\theta)\right)$ with respect to the weights vector (θ). The weights vector (θ) are then updated by adding the momentum vector (m). A commonly used value for the momentum term (β) is 0.9 [108] and the momentum algorithm can easily be called in TF from function "TF.keras.optimizers.SGD(lr=0.001, momentum = 0.9)".

$$\begin{aligned} m &= \beta m + \eta \nabla_{\theta} J(\theta) \\ \theta &= \theta - m \end{aligned} \tag{4.31}$$

A variant of the momentum algorithm developed from Eq. (4.31) takes the gradient of the cost function in the direction of momentum [94]. This algorithm is known as *Nesterov Accelerated Gradient Algorithm* (NAG) [108], shown on Eq. (4.32), helps the momentum vector point in the right direction and usually helps the algorithm converge faster compared to the momentum algorithm. The NAG variant of the momentum algorithm can easily be implemented by setting "nesterov = true" in the "SGD" function from TF.

$$\begin{aligned} m &= \beta m + \eta \nabla_{\theta} J(\theta - \beta m) \\ \theta &= \theta - m \end{aligned} \tag{4.32}$$

Another algorithm called *AdaGrad* [109] is an algorithm that was developed to correct the gradient direction earlier in training. Because it is an adaptive gradient method, AdaGrad runs into an issue where it slows down too fast and does not reach the global minimum. To fix this issue, *RMSProp* was developed to only use gradients from recent iterations instead of using every gradient from the beginning of training [108]. Equation (4.33) shows RMSProp where (β) is the decay rate usually set to 0.9 (similar to momentum), (⊗) refers to element wise multiplication for the cost function gradients, (η) is the learning rate (usually set to 0.001), (∅) refers to element wise division, and (ε) is a small value to protect from dividing by zero. RMSProp can easily be called in TF by “TF.keras.optimizers.RMSprop(lr=0.001, rho = 0.9)”, where “lr” is the learning rate and “rho” is the decay rate.

$$s = \beta s + (1-\beta)\nabla_\theta J(\theta) \otimes \nabla_\theta J(\theta)$$
$$\theta = \theta - \eta \nabla_\theta J(\theta) \oslash \sqrt{s+\varepsilon} \tag{4.33}$$

A more advanced algorithm *Adam* [110] (adaptive moment estimation), was developed to combine the momentum and RMSProp algorithms. The momentum algorithm is an exponentially decaying average of past gradients, which is an estimation of the mean of the gradients; RMSProp is an exponentially decaying average of the past squared gradients, which is an estimation of the variance of the gradients [94]. So, there is a momentum vector (m), which uses a momentum term (β_1), and a variance vector (s), which uses a decay rate (β_2) in Eq. (4.34). (m) and (s) are biased towards zero, so they are then bias-corrected to give bias-corrected first moment (mean) ($\hat{m}$) and bias-corrected second moment (variance) ($\hat{s}$), and are then used to give the weights update rule for (θ). Adam optimization can be called in TF using “TF.keras.optimizers.Adam(lr=0.001, beta_1=0.9, beta_2=0.999)”, where “lr” is the learning rate, “beta_1” is the momentum term, and “beta_2” is the decay rate. An upside to using Adam is that the learning rate (η) does not need to be fine-tuned in most cases and can be kept constant in most cases.

$$m = \beta_1 m + (1-\beta_1)\nabla_\theta J(\theta)$$
$$s = \beta_2 s + (1-\beta_2)\nabla_\theta J(\theta) \otimes \nabla_\theta J(\theta)$$
$$\hat{m} = \frac{m}{1-\beta_1^t}$$
$$\hat{s} = \frac{s}{1-\beta_2^t} \tag{4.34}$$
$$\theta = \theta - \eta \hat{m} \oslash \sqrt{\hat{s}+\varepsilon}$$

Similar to Eq. (4.32), the *Nesterov Accelerated Gradient* (NAG) Algorithm can be applied to Adam [111], which is called Nadam, to improve convergence time [94]. Equation (4.35) shows Nadam and it is very similar Adam, except that the cost function gradient is bias-corrected using the momentum term. Because Nadam is often faster than Adam, it is used for the analysis in this work. Nadam optimization can be called in TF using "TF.keras.optimizers.Nadam(lr=0.001, beta_1=0.9, beta_2=0.999)", where "lr" is the learning rate, "beta_1" is the momentum term, and "beta_2" is the decay rate.

$$
\begin{aligned}
& g = \nabla_\theta J(\theta) \\
& \hat{g} = \frac{\nabla_\theta J(\theta)}{1-\beta_1^t} \\
& m = \beta_1 m + (1-\beta_1) g \\
& \hat{m} = \frac{m}{1-\beta_1^t} \\
& s = \beta_2 s + (1-\beta_2) g \otimes g \\
& \hat{s} = \frac{s}{1-\beta_2^t} \\
& \theta = \theta - \eta \left(\beta_1 \hat{m} + (1-\beta_1) \hat{g} \right) \varnothing \sqrt{\hat{s}+\varepsilon}
\end{aligned}
\tag{4.35}
$$

These algorithms are not the only optimizers available [94]. There are many others on the TF optimizers webpage. TF also allows users to build custom optimizer functions if the user is interested in optimization algorithms. Because this work is not an optimization study, we do not test multiple optimizers because they are very similar and are only concerned with getting minimal error. Nadam was chosen because it typically performs faster than most other optimizers; however, we are not concerned about proving that statement. If users are interested in optimizing the optimizer choice, then users should refer to the Keras-Tuner website [112] to see available methods for tuning NNETs.

IV.III.III Activation Functions: Sigmoid, RELU, and ELU

When backpropagation was developed for NNETs [107], an activation function was used which had continuous derivatives and was nonlinear. The first activation function was just a linear activation model in perceptron's, and to achieve qualities of a continuous nonlinear activation function, the sigmoid function (a.k.a. logistic function) was used and is shown on Eq. (4.36), including its derivative (see Fig. 45). The main issue with the sigmoid function is that when input values become large in positive or negative magnitude, the gradients become very small, and backpropagation barely changes the weights and as the gradient is backpropagated to the beginning layers there is no change. This effect is known as the vanishing gradients problem [94] and is referred to as saturation.

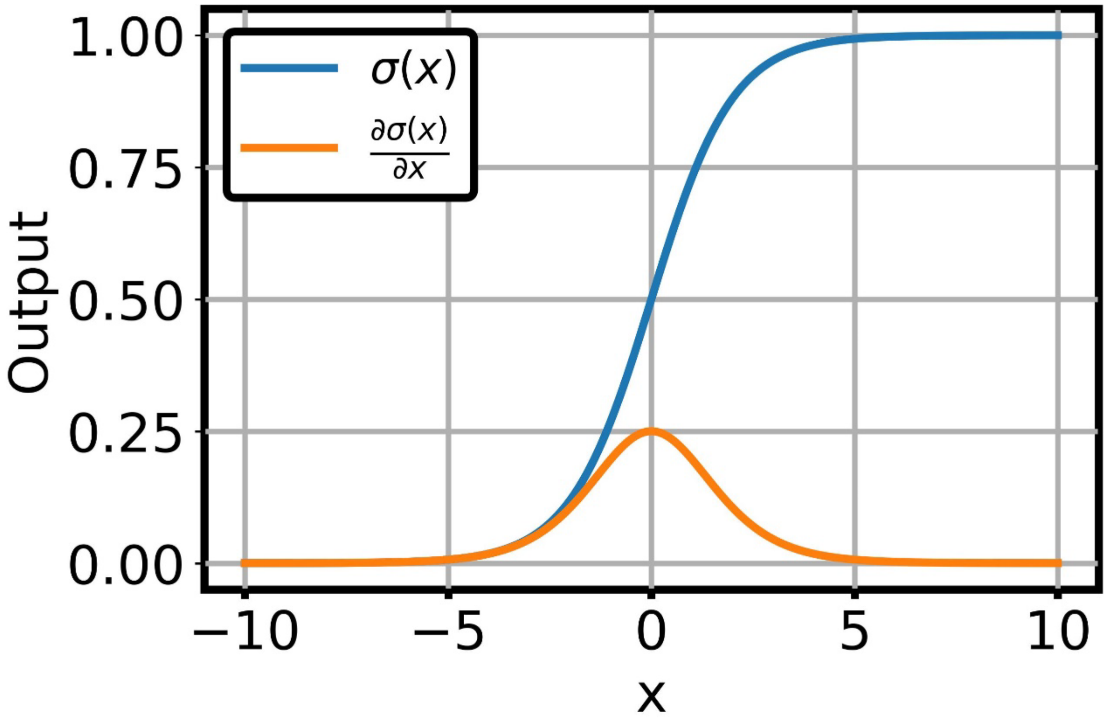


Fig. 44 Sigmoid and its Derivative.

$$\begin{aligned}\sigma(x) &= \frac{1}{1+e^{-x}} \\ \sigma'(x) &= \sigma(x)\left(1-\sigma(x)\right)\end{aligned} \tag{4.36}$$

Many activation functions have been developed over the years, but this work will only focus on two of the most common ones today and they are the *Rectified Linear Unit* Function (RELU) [113] and *Exponential Linear Unit* Function (ELU). RELU is heavily used in deep learning [114] because it avoids saturation for positive values, and it is simple to calculate. If the input is negative then it outputs zero and is linear for positive values. Its derivative is zero or one if the input value is negative or positive, respectively, and can see these properties on Fig. 46. Main issues with RELU are that it is not continuous at zero so the gradient descent can bounce around. Another issue is simply due to the fact that it outputs zero for negative values. RELU can cause units to become saturated for negative inputs and during network training, many units can start to only output zero. This is known as the dying RELU problem [94]. Other variants of RELU have been developed to avoid dying RELUs such as leaky RELU and randomized leaky RELU; most variants use $max(\alpha x, x)$ to avoid having zero for negative inputs [115].

$$\begin{aligned}RELU(x) &= \max(0,x) \\ RELU'(x) &= \max(0,1)\end{aligned} \tag{4.37}$$

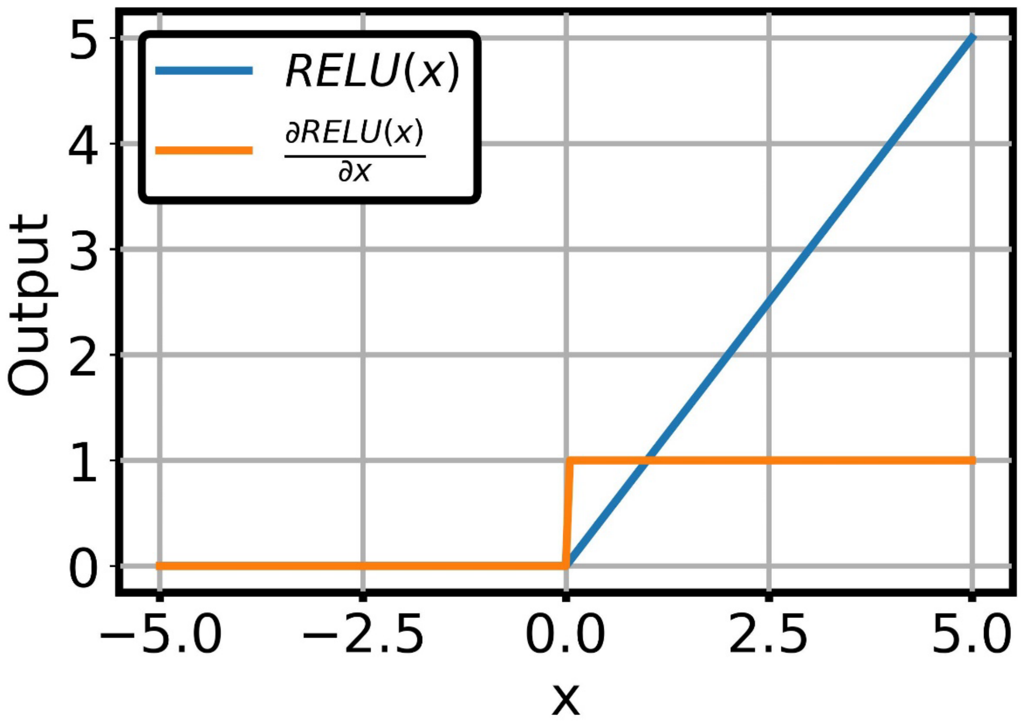


Fig. 45 RELU and its Derivative.

Finally, ELU was developed and shown to outperform RELU and all its variants [116]. ELU is shown on Eq. (4.38) and can see it avoids the dying RELU and vanishing gradients problem by allowing negative values to be output. Figure 47 shows ELU and derivative. An issue with ELU is that it can computationally take longer to calculate compared to RELU [94]. A variant of ELU called SELU was proposed because it allows self-normalization of the layers which solves the vanishing and exploding gradients problem, but there are many restrictions that must be satisfied to see performance gains [117].

$$ELU(x)=\begin{cases}\alpha\left(e^{x}-1\right) & \text{if } x<0\\ x & \text{if } x\geq 0\end{cases}$$
$$ELU'(x)=\begin{cases}\alpha e^{x} & \text{if } x<0\\ 1 & \text{if } x\geq 0\end{cases} \tag{4.38}$$

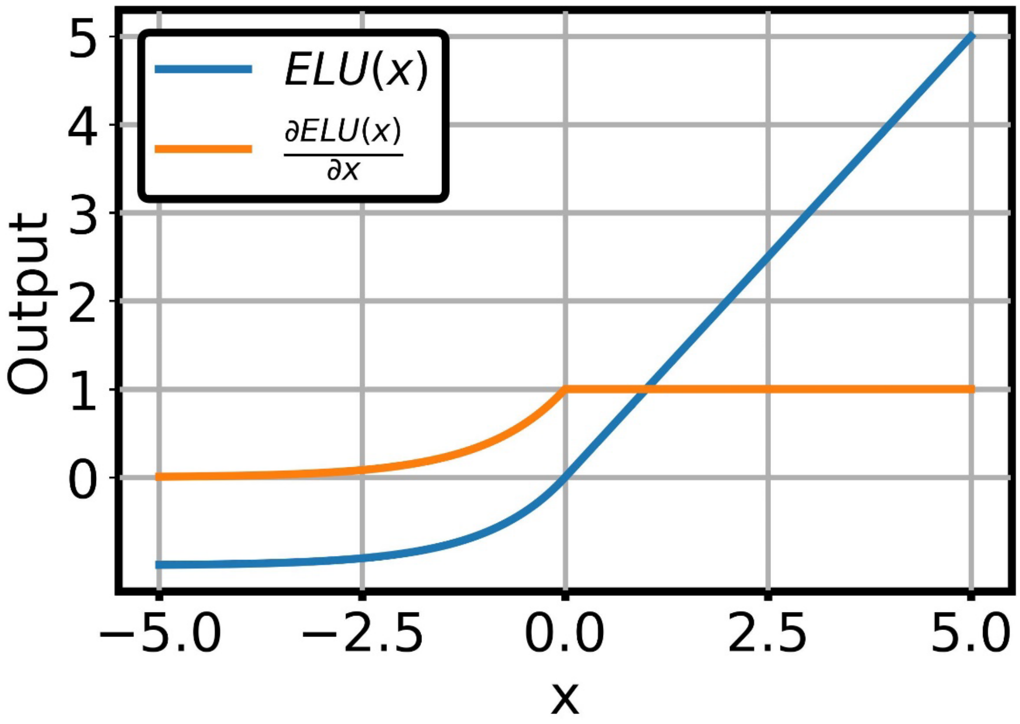


Fig. 46 ELU and its Derivative with $\alpha = 1$

To avoid issues with Sigmoid and RELU and constraints of SELU, ELU is used for this work with a default alpha of one. It is briefly mentioned from the Universal Approximation Theorem that the choice of activation function can lead to specific error bounds based on certain conditions. For example, Cybenko [118] derives a theorem which states for any sigmoidal function, then finite sums of the following form on Eq. (4.39) are dense in $\left(C(I_n)\right)$. Therefore, for any $f \in C(I_n)$ and $\mathcal{E} > 0$, there is a sum $G(x)$, i.e., Eq. (4.39), such that the residual error is less error is less than $\mathcal{E}$, shown on Eq. (4.39). Issues with this theorem is that it does not state how many units are the hidden layer nor does it state how to find the correct approximating function.

There are variations of this theorem, but they also have hard constraints and they suffer from not being able to tell the user what function is required nor do they state how many layers or units are required for convergence [119][120].

$$G(x) = \sum_{j=1}^{N} \alpha_j \sigma\left(y_j^T x + \theta_j\right) \tag{4.39}$$

$$\left|G(x) - f(x)\right| < \varepsilon \text{ for all } x \in I_n \tag{4.40}$$

IV.III.IV NNET Script Methodology

The NNET script methodology is similar to Section: IV.II.I Linear Regression Methodology.

1. Read in data using Pandas "read_csv" function to read data.
2. Use Pandas to separate data into input data and output data.
3. Use Sklearn "train_test_split" to randomly split data into training and testing data.
4. Develop normalization using "tf.keras.layers.Normalization" and use the adapt function to fit to the training data.
5. Develop the NNET model using "tf.keras.Sequential" and use "tf.keras.layers.Dense" to develop a dense feed forward network. The "Sequential" model includes an input layer, the "Normalization" function defined previously, hidden layers using "Dense", which include the number of hidden units and activation function, and output layer

“Dense” function with no activation. The input is the size of the inputs, which is 15 input units, and there are 6 output units.

6. Compile the model using “compile” and include a loss function and optimizer such as “mse” and “adam”, respectively.
7. Train the models “fit” and include the training input/output, validation input/output, and the number of epochs.
8. Predict the output data using the test data.
9. Calculate performance of models such as R^2, MAPE, MSE, and WAPE.

IV.III.V Results: Multi-Layer, Multi-Unit Matrix

The main issue with initially setting up NNETs is that there is no standard method which tells one how to define the NNET to get a suitable answer. Many databases will have a vastly different NNET because it depends on the problem itself. Many NNETs in development today boast a model with billions of parameters and one of the largest models developed by Google [121] has approximately 1.6 trillion parameters. That would mean a model with equal number of layers and equal number of hidden units per layer would have 12,000 layers with 12,000 hidden units per layer. This work requires nothing of this scale and complexity. So, we come back to how to define the model. There are some methods which attempt to optimize the models, but we have found that they are extremely slow due to the optimizer schemes used. Keras-Tuner [122] is a popular method that developed some schemes for optimizing NNET models and can be a great starting point if the user is not sure how to define parameters. A nice feature provided is that a user can try multiple activation function, weight optimizer, loss function, multiple hidden layers, and multiple hidden units per layer, etc.

Typically, we develop NNETs with a specific mindset, keep it as simple as possible. So, we try to keep the model as minimal as possible because a smaller model is easier and quicker to train and can make faster predictions at run time. Our somewhat brute force method will generate multiple NNETS with multiple hidden layers and multiple hidden units, but each layer has the same number of units. A script has been developed which iterate through each combination, so every other parameter is constant such as activation (ELU), loss (MAPE) and optimizer (Nadam). Each model is trained for 1000 epochs using a batch size of 1024 and includes training (400,000 samples) and validation (50,000 samples) data. During each training, the model with best validation loss is saved and used to prevent overfitting. We set an array of hidden layers to be $[1,2,3,4,5]$ and the number of hidden units to be $[10,20,30,40,50,60,70,80,90,100]$ where every layer has same number of hidden units (constant width model).

Table 21 shows the results of each combination of NNETs, where each row refers to a specific number of hidden layers and each column refers to number of hidden units per layer. Can see that for Layer = 1, increasing the number of hidden units does decrease error, and for layer = 2, there is a significant improvement over layer=1, with fewer hidden units per layer required. Will then notice that for layer = 3, the error is again decreased and notice that the minimum error is found at 90 hidden units per layer equal to 0.720 %. Can see that for layers 4 & 5, initially at units equal to 10-40, are doing better than layer 3 but start to increase in error at 50 units and never do as well at layer 3 at 90 hidden units per layer. So, we will then choose this setup configuration with 3 hidden layers and 90 hidden units per layer. Because we only trained each model for 1000 epochs, it is possible to further increase performance of model by training even longer. The next sections will develop model result for 10,000 and 50,000 epochs.

Table 21 NNET MAPE Results

Layer	10	20	30	40	50	60	70	80	90	100
1	9.203	6.579	5.015	4.173	3.790	3.592	3.257	2.991	2.997	2.769
2	3.864	2.193	1.365	1.096	0.988	0.940	0.827	0.902	0.821	0.821
3	3.114	1.577	1.077	0.934	0.834	0.775	0.832	0.746	**0.720**	0.740
4	2.567	1.282	1.007	0.918	0.900	0.817	0.800	0.822	0.824	1.048
5	2.352	1.220	0.986	0.937	0.905	1.008	1.038	0.980	1.060	1.060

IV.III.VI Results: 10,000 Epochs

We then train the model for 10,000 epochs using same model parameters as before. The model chosen has 3 hidden layers with 90 hidden units per layer. Each hidden layer uses ELU activation. The input layer has 15 inputs, and the output layer has 6 outputs. The model is compiled with Nadam optimization using MAPE as the loss function. The model is fit (trained) for 10,000 epochs using the training (400,000 samples) and validation (50,000 samples) data with a batch size of 1,024 samples, which takes approximately 2.2 hours. Because there are 400,000 training samples and mini-batch set of 1,024 samples there are a total of 400,000/1,024 = 390.625 batches or 391 complete batches. After each single batch the weights are updated, so for each epoch the weights are updated 391 times. For 10,000 epochs there will be 3,910,000 million updates to the weights. We train longer to ensure that we have reached the global minimum and if we plot the training loss and validation loss MAPE values over each epoch we should be able to see if the solution has converged.

Figure 48 shows the training and validation loss over each epoch and can see that the error starts to converge but does not plateau like it should. In fact, the best epoch where validation loss is minimum is taken at 9961, which is almost at final 10,000 epochs of training. Based on this information, the model may need to be trained longer, which is why the next section trains for 50,000 epochs. Can see that there is visible noise on the error as well, not so much on the training data, but the validation loss jumps around much more, which is expected since we use mini-batch training.

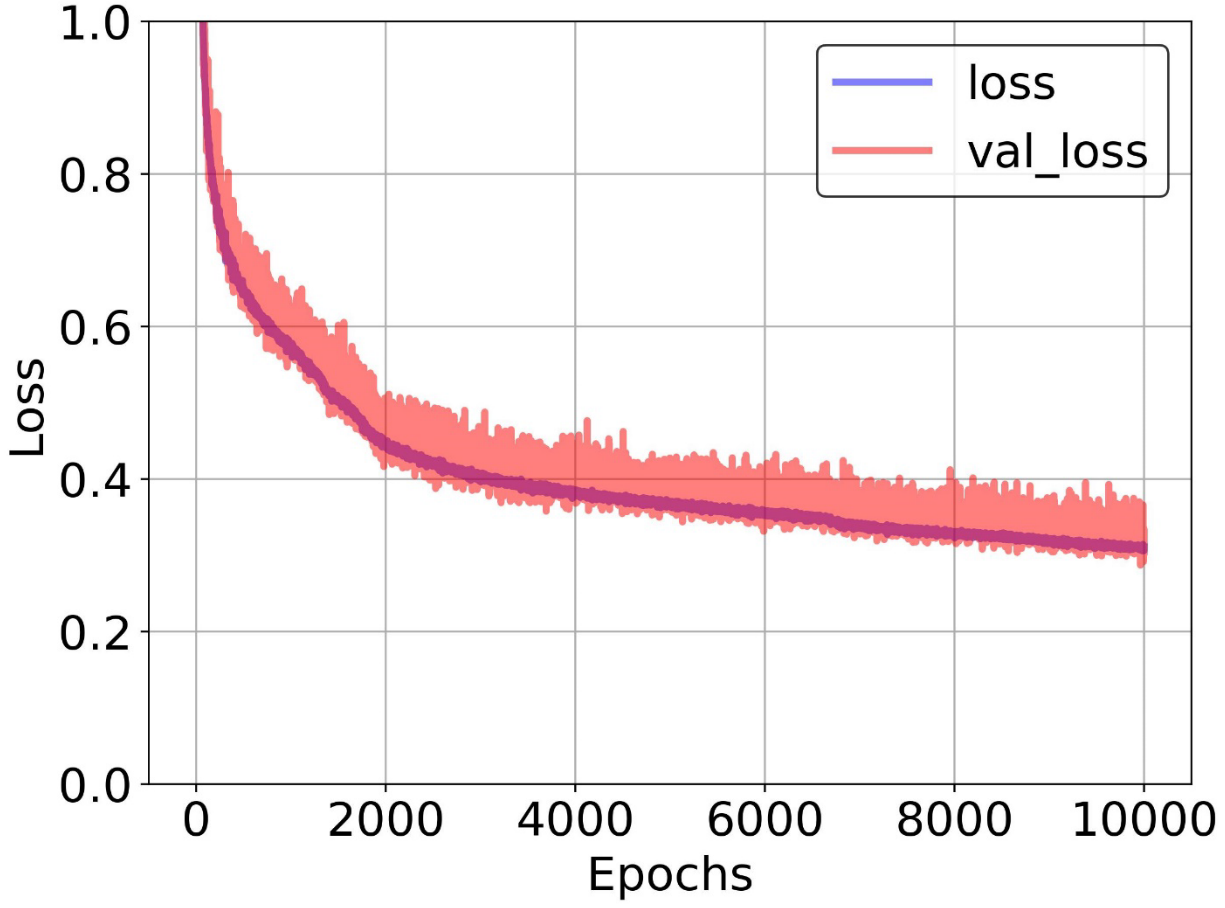


Fig. 47 Training and Validation MAPE History.

Can see that the model has improved for overall metrics compared to Table 11, Table 16, and Table 45 using the testing data. Can see that the R^2 for the NNET has increased above 0.99, which none of the linear regression methods were able to do. Every output metric is lowest for the NNET, which is not surprising.

Table 22 Overall Metrics for 10,000 Epochs Model

Model	Training	Validation	Testing
R^2	0.9988	0.9983	0.9987
MSE	287.9464	446.1618	305.5932
MAPE	0.2749	0.2869	0.2817
MMAPE	0.0479	0.0513	0.0491

When evaluating the individual output metrics on Table 23 can individually see that NNET is doing much better than even the 3rd order regression on Table 45. Can see that APOGEE does not have smallest R^2 but can see from its MSE and MAPE that it is doing much better than any of the other linear regression models developed. R^2 for MAXDIST is greater than 0.99 but its MSE is still quite large as we need it be close to zero as possible. The rest of the other parameters are all modeled quite well and would be very suitable for real life use.

Table 23 Individual Output Metrics

Metric	TOF	MAXTHR	MAXDIST	APOGEE	THRSEA	WEIGHT
R^2	0.99923	0.99999	0.99758	0.99532	0.99999	0.99995
MSE	17.21413	0.00251	1343.25842	473.07996	0.00195	0.00225
MAPE	0.2757	0.08668	0.57047	0.55055	0.09249	0.11435
MMAPE	0.06886	0.02618	0.06998	0.05969	0.02692	0.04325

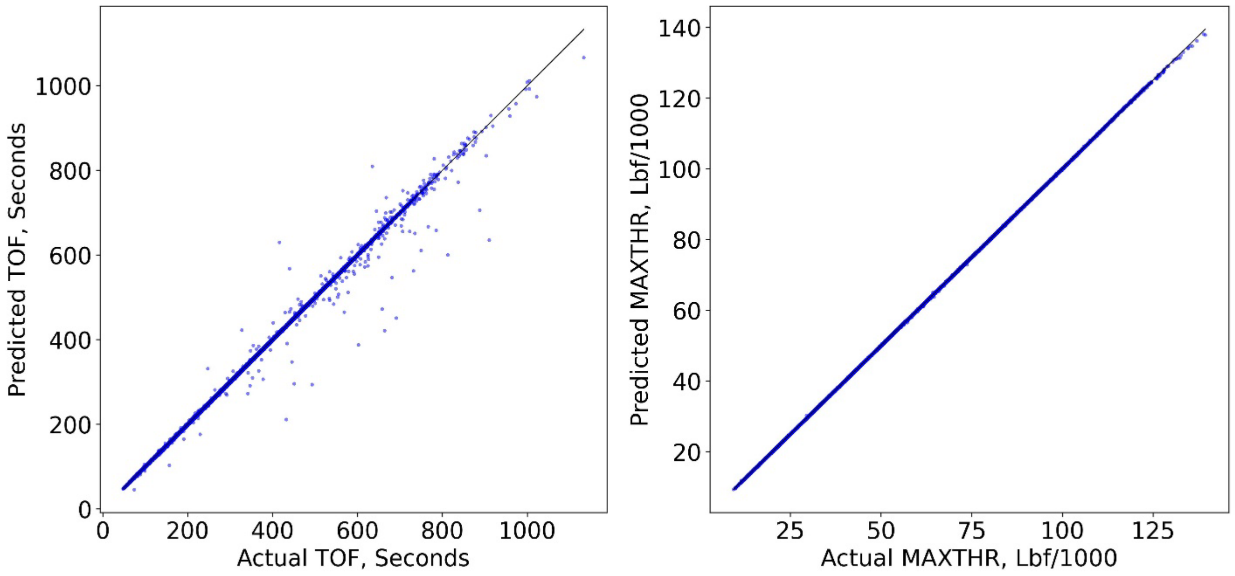
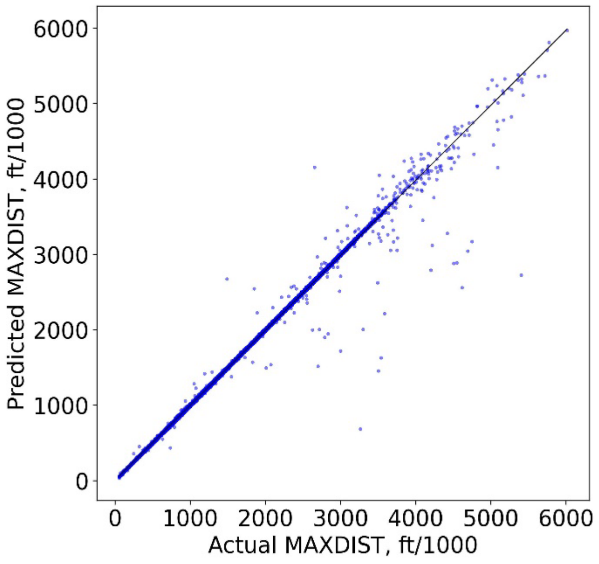


Fig. 48 NNET Model: Predicted vs. Actual for TOF (Left) & MAXTHR (Right).

Figure 49 shows the predicted versus for TOF & MAXTHR. Can see that TOF mostly follows the regression line with some under and over predicted points. MAXTHR is almost a perfect fit. Figure 50 shows the predicted versus actual for MAXDIST and APOGEE and can see both suffer the most in terms of prediction error. MAXDIST seems to have a constant error variance so the model may be able to be trained further to reduce this error. APOGEE still seems to have a second distribution which is not learnable but can see the error is reduced greatly with the NNET compared to Fig. 41, which had the two-way linear regression. Figure 51 shows predicted versus actual THRSEA & WEIGHT and can see they both are almost perfect, except for weight, which has a bit of error in the center.



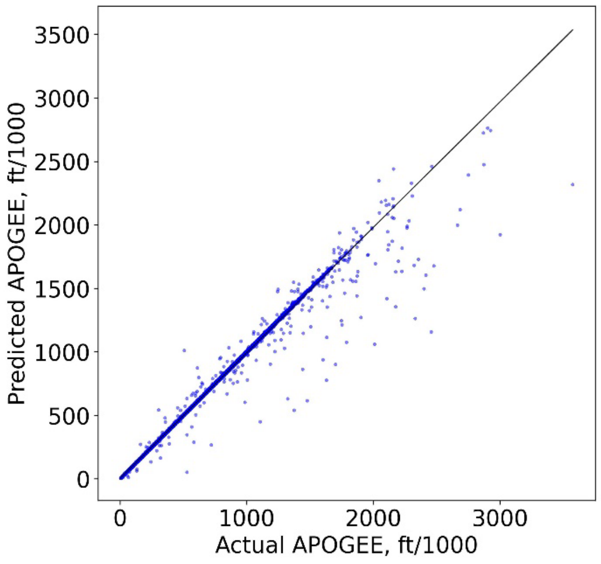
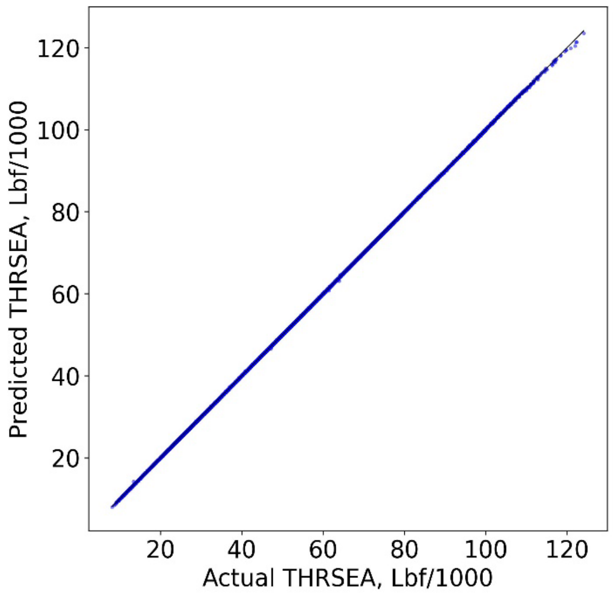


Fig. 49 NNET Model: Predicted vs. Actual for MAXDIST (Left) & APOGEE (Right).



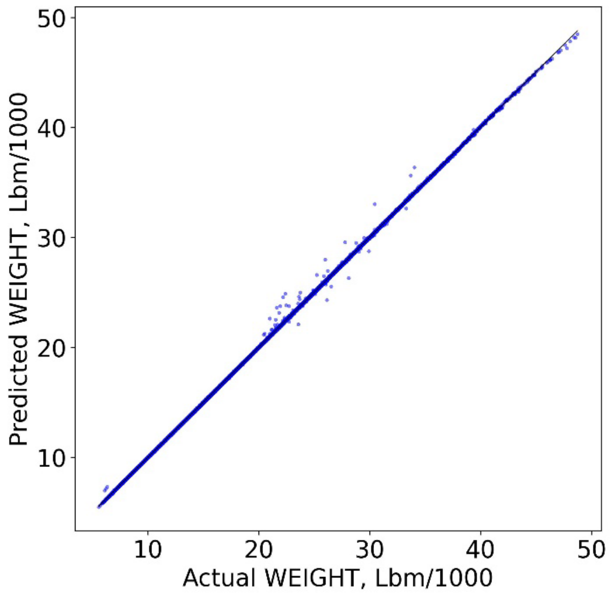


Fig. 50 NNET Model: Predicted vs. Actual for THRSEA (Left) & WEIGHT (Right).

IV.III.VII Results: 50,000 Epochs

This section uses the previously stated model is now trained for 50,000 epochs because it was shown on Fig. 48 that the training and validation loss appeared to be improving still, otherwise it would have been asymptotically approaching a minimum error. Can see on Fig. 52 that the training and validation loss appears to be asymptotically approaching the minimum error. The minimum error occurs at epoch 42,005 and takes approximately 20 hours. Notice that 10,000 epochs was not enough training and could see that the optimizer jumps out of local minimum.

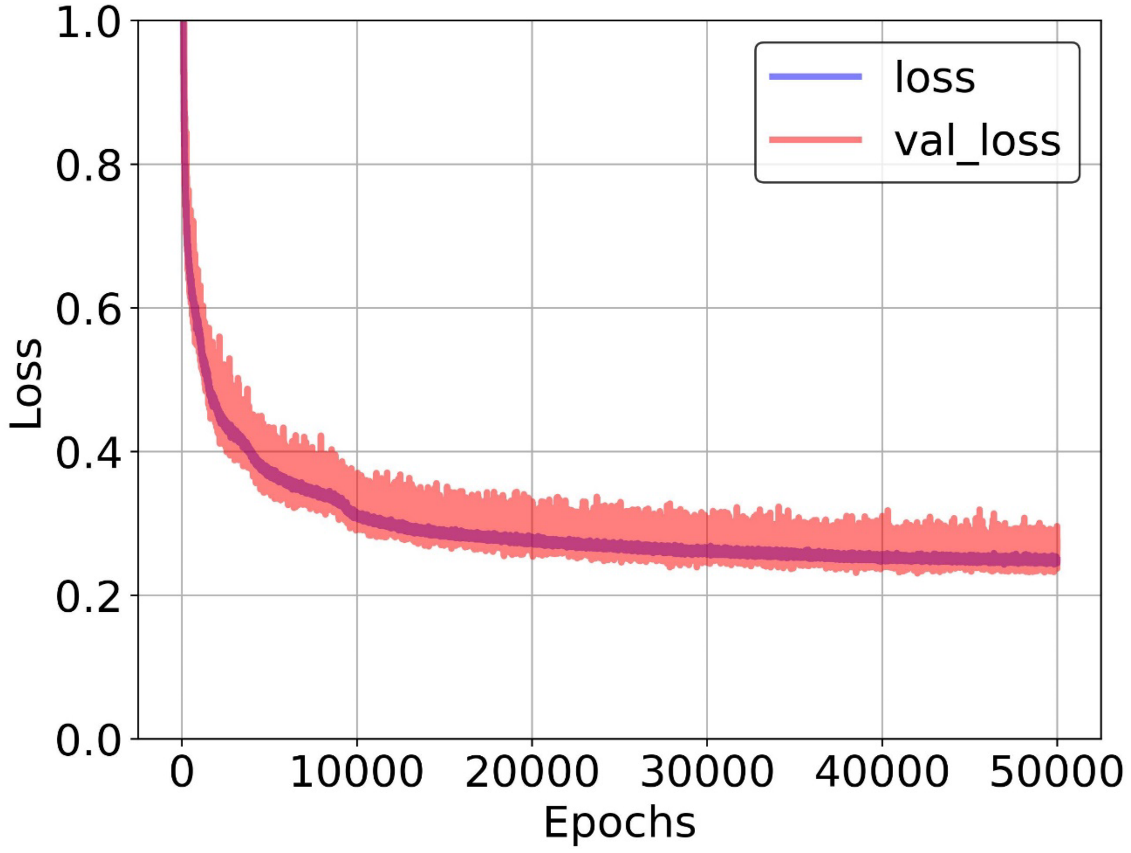


Fig. 51 Training and Validation MAPE History.

Can see that the model has improved from overall metrics on Table 11, Table 16, Table 45, and Table 22 comparing the testing data. Can see that the R^2 has for the NNET has increased above 0.999, which none of the linear regression methods were able to do and the previous NNET only achieve above 0.99. Can see the MSE has significantly decreased from the previous NNET.

Table 24 Overall Metrics for 50,000 Epochs Model

Model	Training	Validation	Testing
R^2	0.9998	0.9995	0.9995
MSE	62.8641	126.6180	144.2935
MAPE	0.2245	0.2302	0.2323
MMAPE	0.0363	0.0375	0.0380

When evaluating the individual output metrics on Table 25 can individually see that NNET is doing better than every parameter from Table 23. R^2 for MAXDIST is greater than 0.999 but its MSE is still the largest. The rest of the other parameters are all modeled well and would be very suitable for real life use.

Table 25 Individual Output Metrics

Metric	TOF	MAXTHR	MAXDIST	APOGEE	THRSEA	WEIGHT
R^2	0.99953	0.99999	0.99866	0.9988	1	0.99997
MSE	10.52652	0.00226	732.8645	122.36478	0.00136	0.00133
MAPE	0.21984	0.07198	0.47541	0.45567	0.07523	0.0958
MMAPE	0.05463	0.02342	0.05457	0.03786	0.02243	0.03506

Figure 53 shows the predicted versus actual for TOF and MAXTHR and has greatly reduced in error and can see there are approximately 6 points with significant error out of 50,000 samples. MAXTHR has almost perfect fit. MAXTHR is almost perfect.

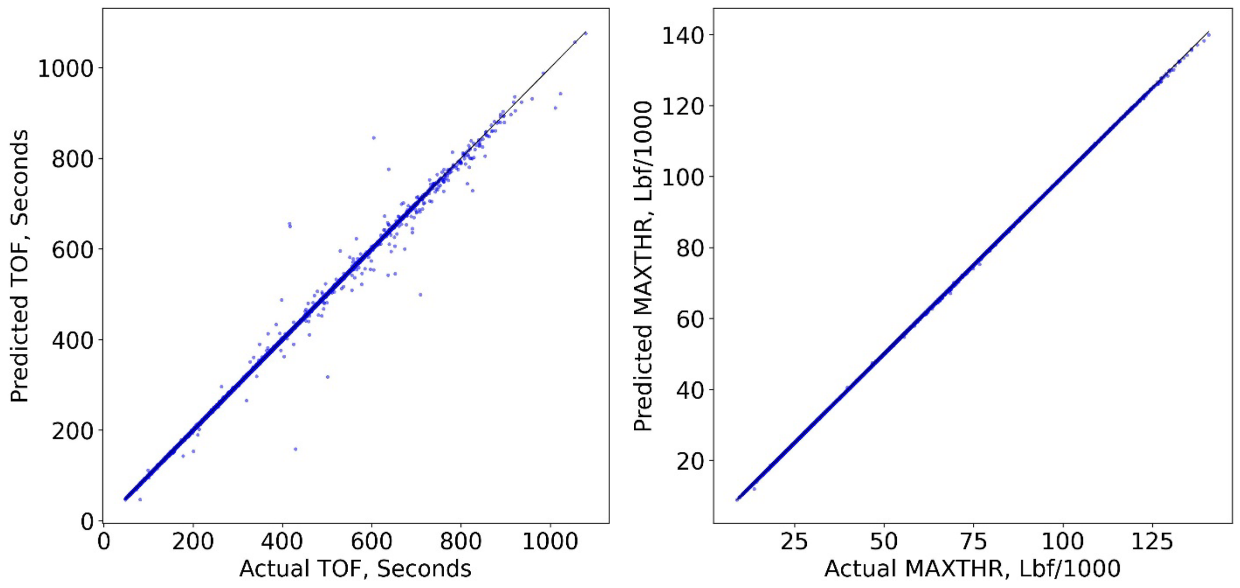


Fig. 52 NNET Model: Predicted vs. Actual for TOF (Left) & MAXTHR (Right).

Figure 54 shows the predicted versus actual for MAXDIST and APOGEE. The error on MAXDIST is greatly reduced, but there are still points with significant error (about 15 points with significant error). APOGEE has greatly reduced error and can now see that there is no obvious second distribution of data not being modeled correctly (about 10 points with significant error). The NNET trained for 10,000 epochs, shown on Fig. 50, was not even able to reduce the error to these metrics.

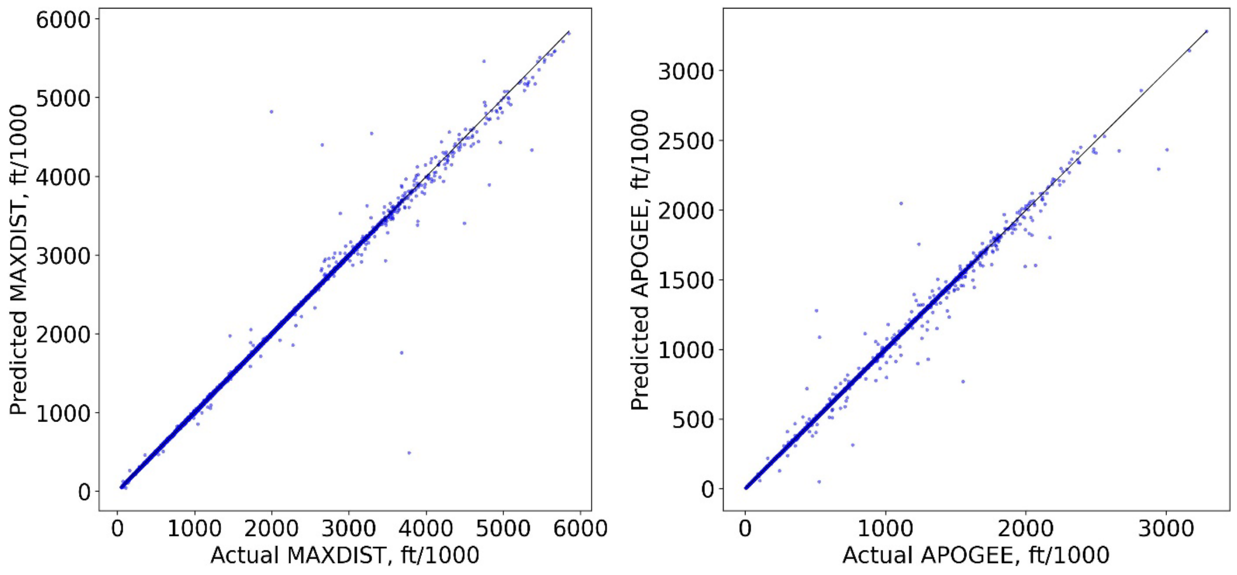


Fig. 53 NNET Model: Predicted vs. Actual for MAXDIST (Left) & APOGEE (Right).

Figure 55 shows the predicted versus actual for THRSEA and WEIGHT. THRSEA and WEIGHT are almost perfect. WEIGHT does seem to have a bit f error still but it is very minimal at least compared to the other parameters.

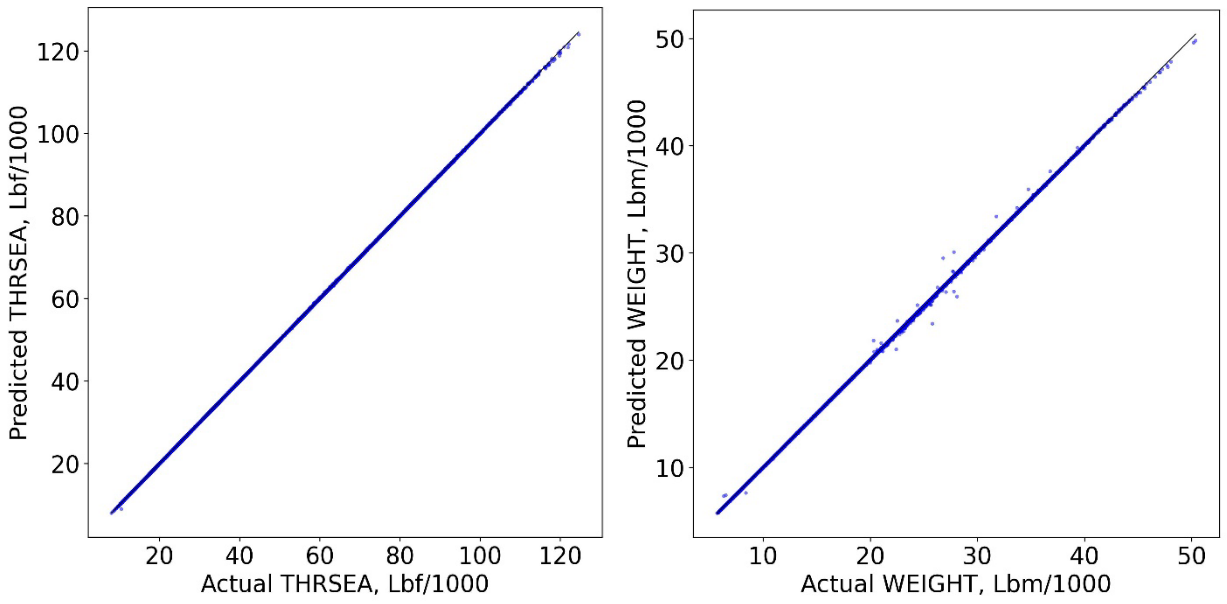


Fig. 54 NNET Model: Predicted vs. Actual for THRSEA (Left) & WEIGHT (Right).

The NNET trained for 50,000 epochs was shown to greatly improve error. It does seem that with enough training the optimizer has learned to improve performance of TOF, MAXDIST, and APOGEE and could see multiple samples with large error. Despite these samples, the model is a vast improvement over the NNET trained for 10,000 epochs and over the linear regression models which were not even capable of producing good models. NNETs are the method required to model such a large database with a nonlinear distribution.

V. Model Generation: AUSRC Classification

The previous chapter developed regression methods to predict outputs of interest. This chapter will then use the outputs of the Auburn tools as inputs as a way to predict which class the sample belongs to. Part of the reason why this is done is because originally we were trying to predict what the inputs would be required to make the outputs using regression. For example, if one knew TOF of a missile, could we then predict the geometry and launch parameters using a regression method. However, this was not successful and so, we then decided to model the problem as a classification problem. Doing this in a way allows us to predict what the inputs are without needing to know the inputs. For example, if one knew TOF of a missile, could we then predict its class value, say 1-12.

V.I Review of Classification in Missiles

Before the classification methods and their applications are discussed in detail, it is important to mention previous works the Hartfield lab has produced with regards to classification in which we are predicting some discrete response such as class value. Some of the first work was done by Albarado [123], which attempted to rapidly classify missiles using logistic regression and NNETs. Results showed that models could quickly classify missiles within 20 seconds of flight time. Next, NNETs were developed to rapidly classify munitions (artillery round and mortars) [124] and results showed that for the true data NNETs could 100% classify both the training and testing data but when noise was added the testing accuracy ranged between 85-99%. Carpenter [125] then extended from [124] to classify and predict long range missiles using NNETs, which show high accuracy and good predictions. Eckert [126] then applied NNETs to rapidly classify long range missiles. The classification models were developed on training data which was altered to simulate radar noise. The radar noise was based on the radar equation and based on the radars' position the noise could be higher or lower depending on how far the missiles' location was. Twelve classes were used for this work and analysis had shown that the even and odd classes could not be classified together but when separated the models had good classification. When the radar data included acceleration, the classification results improved to 92% testing accuracy. This work showed that LDA and NNETs work well, but NNETs came to be more powerful compared to LDA.

V.II Linear and Quadratic Discriminant Analysis

It is always important to develop a baseline of the results because if we can use a simple model over a higher order nonlinear method than we can drastically reduce training time and execution time. Many lower order models like in the linear regression cases provide direct interpretability of the model. In this work, linear and quadratic discriminant analysis (LDA/QDA) is used over logistic regression because logistic regression can be unstable to classes which are extremely different because it fits weights similarly to linear regression and so those weights become largely erratic [92]. Logistic regression also suffers from overfitting in large dimensional datasets and obviously is a linear method so it cannot differentiate nonlinear relationships. Despite these shortcomings of logistic regression, typically in practice logistic regression and LDA/QDA can give very similar results.

With LDA & QDA, it assumed that the samples are from a normal distribution and can be a downside to this method [93]. However, with missile classification there is no exact way to determine if a class of missiles follows a normal distribution, but we can make specific arguments that allow it to be normal. Consider missile A which has been developed for real life use. Missile A was probably developed to hit some target at some distance D. Designing and analysis of missile A would have determined the maximum and minimum distance D missile A is capable of traveling. Now missile A could technically have a zero distance if it was launch on the ground laying down; however, in real life practice this would be greatly avoided so we discount such scenarios. Most likely, Missile A would only ever be launched at specific launch angles to avoid failure. So, we can surmise that missile A has a specific range that is only ever used for max operability. Therefore, it seems safe to assume that the observations we are using for classification follow a normal distribution.

The derivation of LDA & QDA is not shown in entirety as there are many conditions, which can be described for discriminant analysis [127]. The Bayes Theorem is shown on Eq. (5.1) where $P(x|y=k)=f_k(x)$ is the density function of *X* for a sample that belongs to class *k* out of *K* classes. $P(y=k)=\pi_k$ is the prior probability that random samples belong to class *k*. $P(y=k|X=x)$ is then the posterior probability that a sample belongs to class *k*. The prior probability is often an assumption about the data and is often the proportion of class *k* samples out of the entire set of samples, so $\pi_k=N_k/N$ is used in the scikit-learn implementation. $f_k(x)$ is usually never available in real life and for LDA/QDA is assumed to be normal, shown on Eq. (5.2) , where *d* is the number of inputs.

$$\begin{gathered} P(y=k\mid X=x)=\frac{P(x\mid y=k)P(y=k)}{P(x)} \\ P(y=k\mid x)=\frac{P(x\mid y=k)P(y=k)}{\sum_{i=1}^{K}P(x\mid y=i)P(y=i)}=\frac{f_k(x)\pi_k}{\sum_{i=1}^{K}f_i(x)\pi_i} \end{gathered} \tag{5.1}$$

$$f_k(x) = \frac{1}{(2\pi)^{d/2} |\Sigma_k|^{1/2}} \exp\left[-\frac{1}{2}(x-\mu_k)^T \Sigma_k^{-1} (x-\mu_k) \right] \tag{5.2}$$

If the log of the posterior function is taken and reorganized, then we get Eq. (5.3). This equation is the log-posterior function for QDA because QDA assumes that the covariance matrices Σ_k are different for each class *k*. When calculating a prediction with a sample, there are then *K* predictions and the prediction with the largest value means it belongs to class *k*. The log-posterior is often written as $\delta_k(x)$ and can be thought of as a decision plane [92]. As the name suggests, QDA has a quadratic decision plane and fits to data using a quadratic function.

$$\begin{aligned}
&\log P(y=k \mid x) = \log P(x \mid y=k) + \log P(y=k) - \log \textstyle\sum_{i=1}^{K} f_i(x)\pi_i \\
&\log P(y=k \mid x) = \log f_k(x) + \log \pi_k - \log \textstyle\sum_{i=1}^{K} f_i(x)\pi_i \\
&\log P(y=k \mid x) = -\frac{1}{2}\log|\Sigma_k| - \frac{1}{2}(x-\mu_k)^T \Sigma_k^{-1} (x-\mu_k) + \log \pi_k + C \\
&C = -\log \textstyle\sum_{i=1}^{K} f_i(x)\pi_i - \frac{d}{2}\log(2\pi)
\end{aligned} \tag{5.3}$$

For LDA, the covariance matrices are all equal ($\Sigma_k = \Sigma$) for each class. Equation (5.3) can then be reduced to Eq. (5.4). As the name suggests, LDA is a linear decision plane that fits the data using a linear function. Equation (5.4) can be rewritten further into Eq. (5.5) and is written similarly to a linear regression function where w_k refer to the weights and w_0 refers to the intercept. Scikit-Learn sets the output coefficients ("coef_") and output intercept ("intercept_") from the fit model.

$$\begin{aligned}
&\log P(y=k \mid x) = -\frac{1}{2}(x-\mu_k)^T \Sigma^{-1} (x-\mu_k) + \log \pi_k + C \\
&C = -\log \textstyle\sum_{i=1}^{K} f_i(x)\pi_i - \frac{d}{2}\log(2\pi) - \frac{1}{2}\log|\Sigma|
\end{aligned} \tag{5.4}$$

$$\begin{aligned} \log P\left(y=k \mid x\right) &= w_k^t x + w_0 + C \\ w_k &= \Sigma^{-1} \mu_k \\ w_0 &= -\frac{1}{2} \mu_k^T \Sigma^{-1} \mu_k + \log \pi_k \end{aligned} \tag{5.5}$$

Unfortunately for QDA, it cannot be written as Eq. (5.5) so the coefficients and intercept cannot be used for model interpretation like LDA. Another feature of LDA allows linearly reducing the data to $(K-1)$ features. Reducing the number of dimensions allows for interpretation of the model visually. It will later be shown that if the data is reduced to 2 components, can visually observe how well the model is fitting and can see how much variance the reduced parameters model.

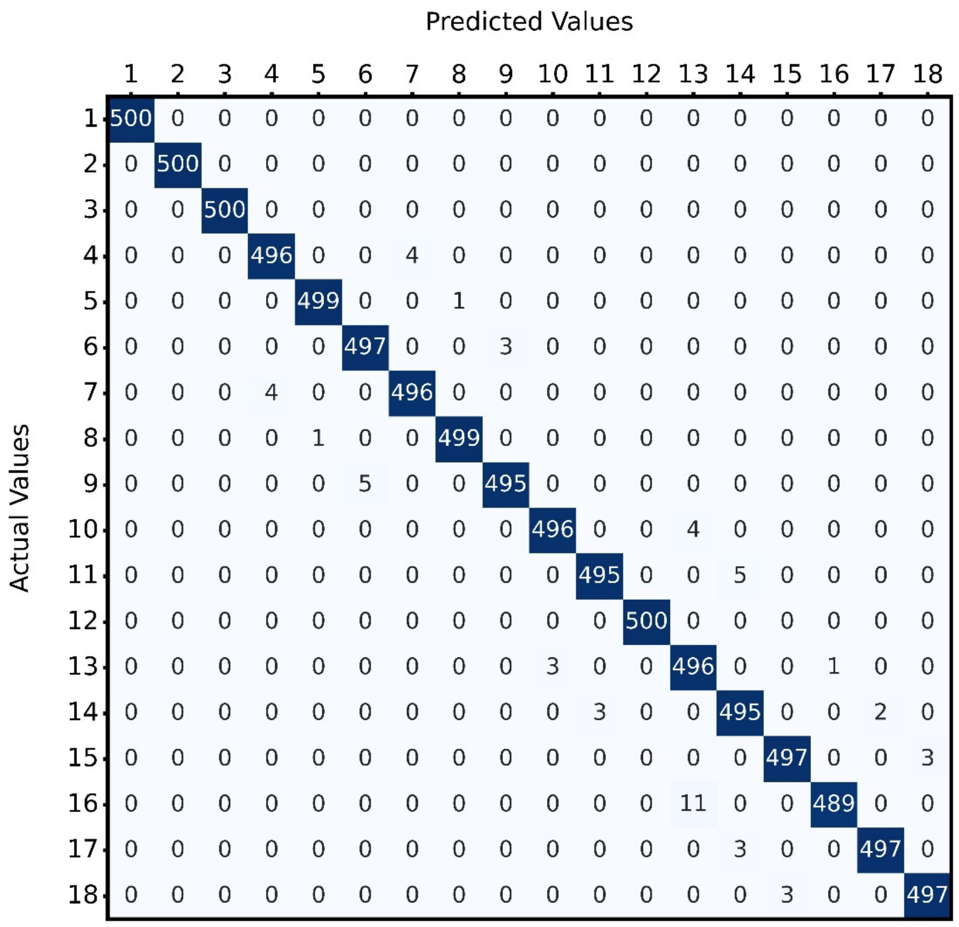
V.II.I Linear Discriminant Analysis Results

The methodology for using LDA and QDA is essentially the same as the linear regression methodology with exception of using the polynomial features and standardization of the input data. Everything else is the same such as using fit and predict. The data is split into 36,000 samples for training and 9,000 samples for testing data. The convenience of using LDA and QDA is that data does not to be standardized before training. Table 26 the accuracy of the model overall and can see that it is almost 100%. This data is obviously easily separable using LDA and it is not necessary to do QDA or even NNETs but is shown in this work because extensions of this classification database would show the LDA is not capable of separating the classes. For example, the extended version of this classification database includes 5-point, 9-point, and 11-point star grains for a total of 72 classes. The accuracy for that dataset is 78.17% for training data and 78.47% for testing data. We refrain from showing the 72 classes because it is too large of a dataset to fit in this work and would not be able to observe visuals which are extremely useful for classification.

Table 26 LDA Training & Testing Accuracy

Model	Train	Test
LDA	99.4694	99.3778

The confusion matrix is shown on Fig. 56 and shows the predicted versus actual of the testing data and is similar to Fig. 37 but for classification models. If the model were perfect then


Predicted Values
Actual Values

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
1	500	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	500	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	500	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	496	0	0	4	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	499	0	0	1	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	497	0	0	3	0	0	0	0	0	0	0	0	0
7	0	0	0	4	0	0	496	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	1	0	0	499	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	5	0	0	495	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	496	0	0	4	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	495	0	0	5	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0	500	0	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0	3	0	0	496	0	0	1	0	0
14	0	0	0	0	0	0	0	0	0	0	3	0	0	495	0	0	2	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	497	0	0	3
16	0	0	0	0	0	0	0	0	0	0	0	0	11	0	0	489	0	0
17	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0	0	497	0
18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0	0	497

Fig. 55 LDA Confusion Matrix for Testing Data.

the diagonal would report constant values of 500 samples per class. For example, can see that for the actual class 4, 496 samples are correctly predicted as class 4, but 4 samples are incorrectly predicted as class 7. It appears that incorrect prediction is being under and/or over predicted by 3 classes. Table 27 shows the precision, recall, and F1-score of the model using the testing data.

Precision is the number of true positives divided by the sum of true positives and false positives. For example, precision for class 4 is defined as $496/(496 + 4) = 0.992$ which is reported on Table 27. Precision is easily thought of as the accuracy along the predicted class k column on Fig. 56. Recall is the number of true positive divided by the sum of true positive and false negatives. For examples, recall for class 14 is defined as $495/(495 + 3 + 2) = 0.99$ which is report on Table 27. Recall can be thought of as accuracy along the actual class k row. F1-score is the harmonic mean of recall and precision and considers contribution of both parameters for minimization purposes. Often sacrifices must be made for either precision and recall and depends on a case by case purpose. So, to ensure maximization of both parameters, F1-score can be used instead.

Table 27 LDA Classification Report: Precision, Recall, & F1-Score

Class	Precision	Recall	F1-Score	Support
1	1	1	1	500
2	1	1	1	500
3	1	1	1	500
4	0.992	0.992	0.992	500
5	0.998	0.998	0.998	500
6	0.99004	0.994	0.99202	500
7	0.992	0.992	0.992	500
8	0.998	0.998	0.998	500
9	0.99398	0.99	0.99198	500
10	0.99399	0.992	0.99299	500
11	0.99398	0.99	0.99198	500
12	1	1	1	500
13	0.97065	0.992	0.98121	500
14	0.9841	0.99	0.98704	500
15	0.994	0.994	0.994	500
16	0.99796	0.978	0.98788	500
17	0.99599	0.994	0.99499	500
18	0.994	0.994	0.994	500

Because the model does extremely well, all three metrics are extremely close to one, which they should be. If we count the number of misclassified, can see that there are 191 misclassified training samples and 56 misclassified testing samples. Can see that the misclassification rates are less than 1 percent, which is in excellent agreement for this model. In general, such high fidelity results do not always occur and varies across different datasets.

Table 28 LDA Misclassification and MisRate

Metric	Train	Test
Misclassification	191	56
MisRate (%)	0.5306	0.6222

Traditionally for classification, the receiver operating characteristic (ROC) curve is used to assess a model at different classification thresholds. Each class true positive rate is plotted against the false positive rate. Can see from Fig. 57, ROC curve is plotted for LDA and can see that as the false positive rate increases (decreased threshold) increases the number of true positive for model. The primary function of this figure is to have each curve be on the left of the black dashed line and want the curve to go vertical at early false positive rate. The earlier the curve goes vertical shows that there will be less false positives and more true positives. In the next section, QDA is used and can see an improvement.

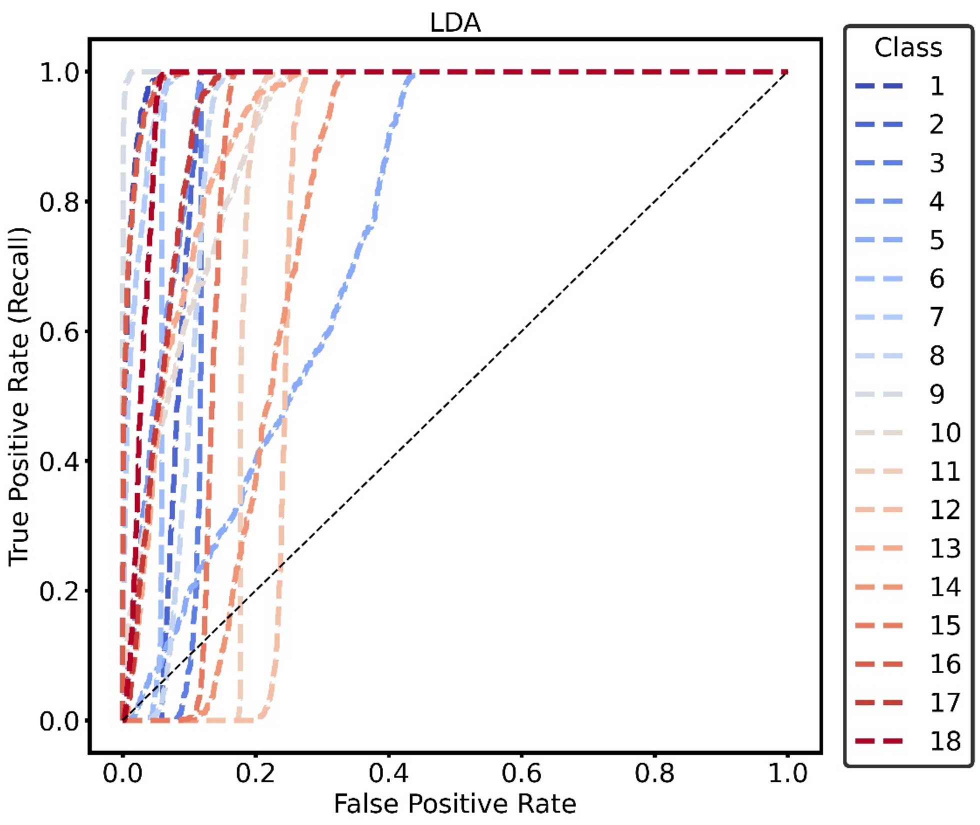


Fig. 56 LDA ROC Curve.

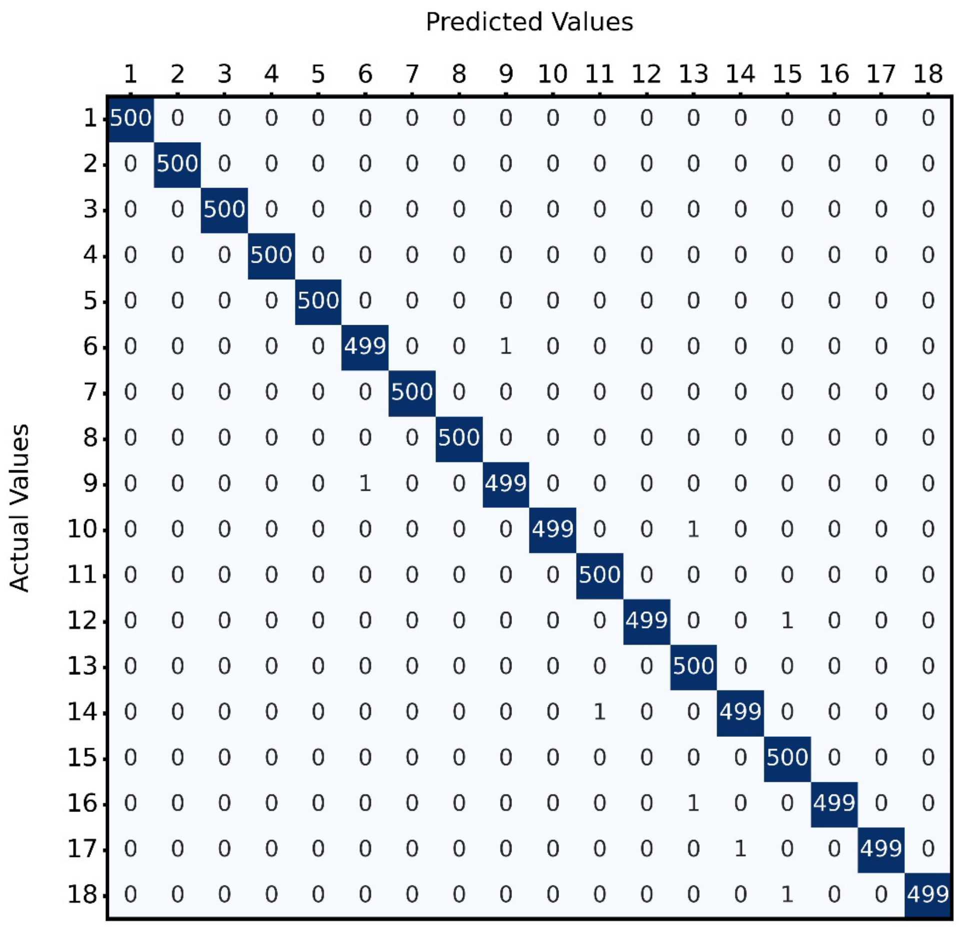
V.II.II Quadratic Discriminant Analysis Results

The same set of training and testing samples is used for QDA and follows same methodology as LDA, which are both similar to linear regression method in Scikit-Learn. Can see the training and testing accuracy for QDA on Table 29. Can see that QDA has achieved an increase in accuracy compared to LDA and both are still very good models.

Table 29 QDA Training & Testing Accuracy

Model	Train	Test
QDA	99.9444	99.9111

The confusion matrix for the model is shown on Fig. 58 and can see that there are only 8 misclassified test samples. Table 30 shows that the number of misclassified is only 20 for the training set and the misclassification rates are almost zero. Its obvious QDA improves accuracy.



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
1	500	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	500	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	500	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	500	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	500	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	499	0	0	1	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	500	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	500	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	1	0	0	499	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	499	0	0	1	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	500	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0	499	0	0	1	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	500	0	0	0	0	0
14	0	0	0	0	0	0	0	0	0	0	1	0	0	499	0	0	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	500	0	0	0
16	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	499	0	0
17	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	499	0
18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	499

Fig. 57 QDA Confusion Matrix for Testing Data.

Table 30 QDA Misclassification & MisRate

Metric	Train	Test
Misclassification	20	8
MisRate (%)	0.0556	0.0889

Table 31 shows the QDA classification report and can see that many of the classes have perfect precision, recall, and F1-score.

Table 31 QDA Classification Report: Precision, Recall, & F1-Score

Class	Precision	Recall	F1-Score	Support
1	1	1	1	500
2	1	1	1	500
3	1	1	1	500
4	1	1	1	500
5	1	1	1	500
6	0.998	0.998	0.998	500
7	1	1	1	500
8	1	1	1	500
9	0.998	0.998	0.998	500
10	1	0.998	0.999	500
11	0.998	1	0.999	500
12	1	0.998	0.999	500
13	0.996	1	0.998	500
14	0.998	0.998	0.998	500
15	0.996	1	0.998	500
16	1	0.998	0.999	500
17	1	0.998	0.999	500
18	1	0.998	0.999	500

We can see from Fig. 59 that the ROC curve shows that each class is being perfectly predicted and so it is obvious that this model performs extremely well. There are still false positives, but they are so few. This model and LDA could be used for deployment purposes. Deploying Scikit-Learn models is very easy using the Python package called Pickle, which allows exporting and importing of multiple data formats such as integers, floats, dictionaries/list, and even user defined function/built-in functions. This is useful because LDA/QDA from Scikit-Learn can export the function along with the coefficients (or other parameters) of the model to a file and can easily be imported along with any of the models' parameters.

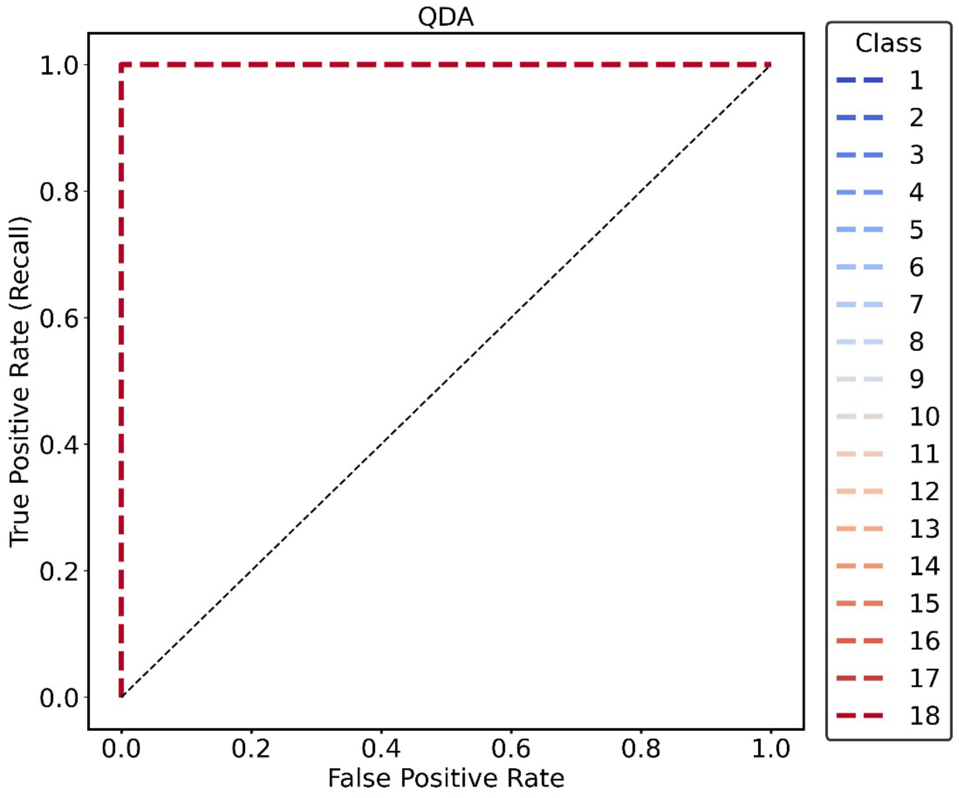


Fig. 58 QDA ROC Curve.

V.III Neural Networks: Classification

If the LDA and QDA models were classifying poorly, then a NNET would be required to improve performance. Instead, we showed that LDA and QDA were doing quite well and a NNET would not be required to fit the data. However, this work is a development of classical and modern methods, so the NNET formulation is developed and shown here.

V.III.I NNET Script Methodology

The methodology is about the same as the NNET regression model, except for a few slight changes. First major change here is that a one-hot encoder is used to convert the output class value to a binary vector of 0's and 1's. For example, if there are 3 classes the first class would be converted to [1,0,0], second class would be [0,1,0], and finally the third class would be [0,0,1]. Since there are 18 classes, each class output would be converted to a binary vector of 18 elements. Finally, the last main difference is that the output layer uses a SoftMax activation to convert output to a probability between 0 and 1. Each output unit in the output layer gets converted using the SoftMax function on Eq. (5.6). Each output unit ranges between 0-1 and every output unit will sum to 1. The output unit with the maximum value is considered to be the class. For example, if output unit 4 (i.e., class 4) has the largest value for the input sample, then it is predicted as class 4.

$$\sigma\left(z_i\right)=\frac{e^{z_i}}{\sum_{j=1}^{K}e^{z_j}} \tag{5.6}$$

The model is then compiled with different parameters. The optimizer used is still Nadam, but Nadam requires a different function to minimize. The NNET regression model used MAPE, but for the classification model, the output is a SoftMax activation, so the categorical cross entropy (CCE) function is used. Equation (5.7) shows the CCE function and can see it's just the summation of each class and each multiple is a value in the true one-hot encoded vector output unit *j* times the log of predicted SoftMax output unit *j*.

$$CCE=-\sum_{j=1}^{K}Y_{\text{True},\,j}\log\left(Y_{\text{Predicted},\,j}\right) \tag{5.7}$$

V.III.II NNET Results Trained for 250 Epochs

Because LDA & QDA can easily replicate the data, the NNET will also be able to easily replicate the data. The first model shown uses 9,000 samples for testing, 4,500 samples for validation, and 31,500 samples for training. It uses a single layer with 50 hidden units and was trained for 250 epochs. Can see that the accuracy increases to 1.00 quickly within 100 epochs and the CCE goes to zero around 200 epochs. Even the error seems to be near zero, the best epoch is

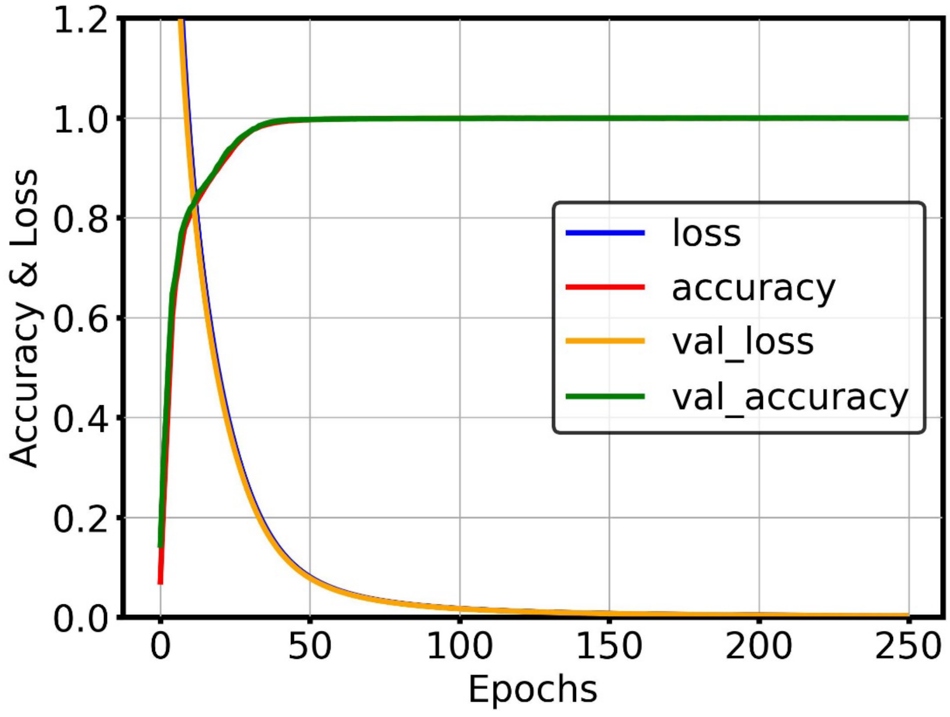


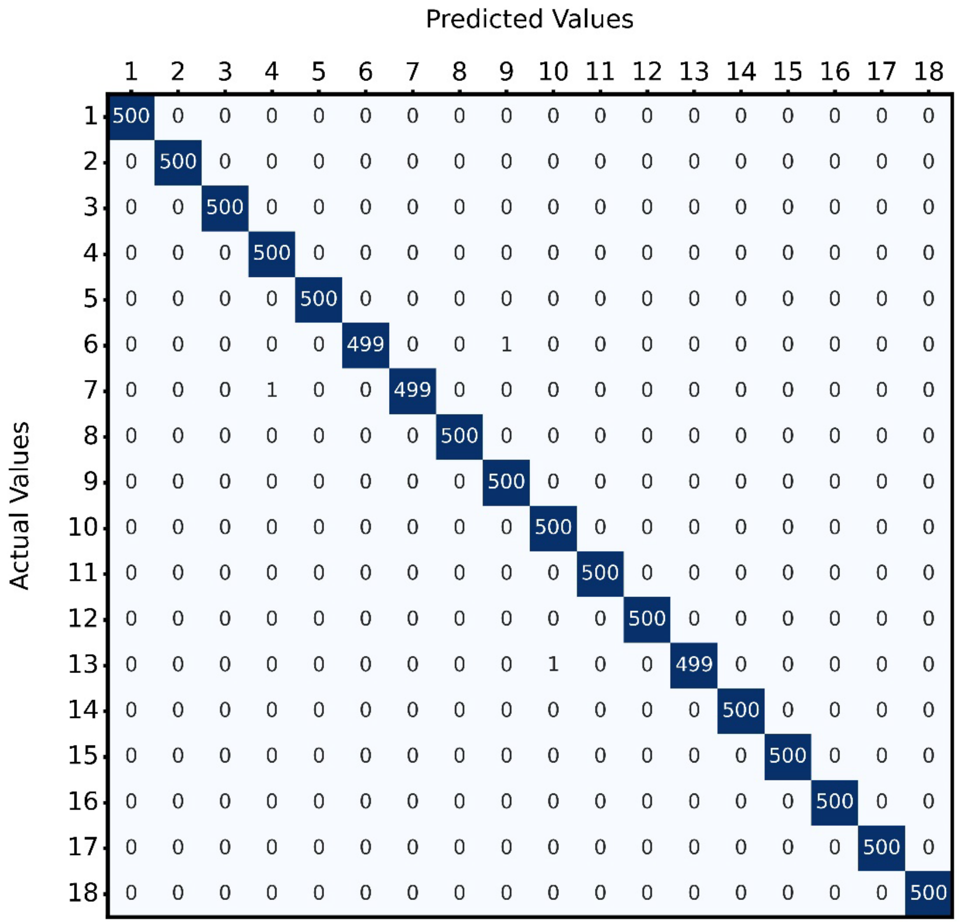
Fig. 59 Training/Validation Loss & Accuracy.

calculated at epoch 250. Can see the testing CCE is 0.0023 on Table 32 and the accuracy is 99.97%. Can see that there are only 3 samples misclassified out of 9,000 testing samples.

Table 32 NNET CCE & Accuracy

Metric	Training	Validation	Testing
CCE	0.0025	0.0025	0.0023
Accuracy	0.9996	0.9996	0.9997
MisRate	0.0381	0.0444	0.0333
Misclassified	12	2	3

We can see the confusion matrix on Fig. 61 and we can see that there are only 3 samples not predicted correctly. It appears that the misclassified samples are under or over predicting by 3 classes. Table 33 shows the precision, recall, and F1-score of the testing data and shows most classes have value of 1.00 and the rest of the other metrics are almost perfect.



Predicted Values

Actual Values

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
1	500	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	500	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	500	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	500	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	500	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	499	0	0	1	0	0	0	0	0	0	0	0	0
7	0	0	0	1	0	0	499	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	500	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	500	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	500	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	500	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0	500	0	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0	1	0	0	499	0	0	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0	500	0	0	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	500	0	0	0
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	500	0	0
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	500	0
18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	500

Fig. 60 NNET Confusion Matrix.

Table 33 NNET Test Data Classification Report

Class	Precision	Recall	F1-Score	Support
1	1	1	1	500
2	1	1	1	500
3	1	1	1	500
4	0.998	1	0.999	500
5	1	1	1	500
6	1	0.998	0.999	500
7	1	0.998	0.999	500
8	1	1	1	500
9	0.998	1	0.999	500
10	0.998	1	0.999	500
11	1	1	1	500
12	1	1	1	500
13	1	0.998	0.999	500
14	1	1	1	500
15	1	1	1	500
16	1	1	1	500
17	1	1	1	500
18	1	1	1	500

V.III.III NNET Results for 500, 1000, & 5000 Epochs

If the NNET is trained for 500, 1000, and 5000 epochs can see how the metrics improve on Table 34. Can eventually see the number of misclassified samples decrease to zero for the training and validation data if the model is trained for 5000 epochs. However, the number of misclassified increases for the testing data but it is still very small. It may be worth having some training error to have minimal testing error.

Table 34 NNET Metric for 500, 1000, & 5000 Epochs

Epoch	Metric	Training	Validation	Testing
500	CCE	0.0008	0.0009	0.0007
500	Accuracy	0.9999	0.9998	0.9999
500	MisRate	0.0127	0.0222	0.0111
500	Misclassified	4	1	1
1000	CCE	0.0004	0.0003	0.0004
1000	Accuracy	0.9999	1	0.9998
1000	MisRate	0.0063	0	0.0222
1000	Misclassified	2	0	2
5000	CCE	0.0001	0	0.0005
5000	Accuracy	1	1	0.9997
5000	MisRate	0	0	0.0333
5000	Misclassified	0	0	3

V.III.IV NNET Classification: Multi-Layer, Multi-Unit Matrix

The classification model is trained using 1-5 layers and 5-50 hidden units per layer and Table 35 shows the testing accuracy per each model. Each model was trained for 1000 epochs. It is however not useful for this case because the classification model does so well with very few units. The author chose to use 50 units for a single layer to ensure adequate accuracy.

Table 35 Classification Accuracy: Multi-Layer, Multi Unit Matrix

Layer	1	2	3	4	5
5	0.999556	0.999556	0.999333	0.999444	0.999778
10	0.999667	0.999778	0.999778	**0.999778**	0.999556
15	**0.999778**	0.999778	0.999778	0.999778	0.999667
20	0.999778	0.999778	0.999667	0.999778	0.999778
25	0.999778	0.999667	0.999556	0.999667	0.999667
30	0.999667	**0.999889**	**0.999889**	0.999778	0.999667
35	0.999778	0.999778	0.999778	0.999778	0.999778
40	0.999778	0.999778	0.999889	0.999667	**0.999889**
45	0.999222	0.999556	0.999889	0.999444	0.999556
50	0.999667	0.999889	0.999778	0.999444	0.999778

VI. Model Robustness & Sensitivity

The models developed in the previous two chapters both used data that was simulated from Fortran codes which provided all the necessary data. Samples which caused errors during simulation were removed so the dataset was not corrupted with bad samples. When data is collected during real life scenarios such as a health survey or U.S. census, often times when this type of data is collected information is missing. Each sample taken can randomly have multiple missing parameters. If it is possible, then imputation can be done to replace the missing data [100]. There are multiple ways of imputation proposed and reference [128] explains most of the proposed methods. Scikit-Learn learns multiple methods for easily implementing imputation such as multivariate imputation with MICE [129][130], and K-Nearest Neighbors imputation [131]. Recently, more advanced methods using generative adversarial nets (GANs) have been used for imputation but were not used for this work [132]. Future works will consider using GANs.

For the imputation methods used in this work, we use a combination of the linear regression methods and the NNETs. The imputation models here will be used for the classification data only and simulate a combination of parameters missing. For example, BURNTIME could be missing or MAXPC & MAXTHRUST could be missing. However, at least one parameter is required for imputation, a model in general requires at least one parameter. The number of combinations is 4 choose 1 plus 4 choose 2 plus 4 choose 3 equals $(4 + 6 + 4 = 14)$ and so there are 14 models developed per imputation method. There are 4 attempted imputation methods using linear regression with one-way terms, two-way terms, three-way terms, and NNETs. Therefore, there are 56 models in total developed to attempt imputation.

VI.I Imputation Methodology

Implementing missing data is relatively simple. First, the data is read in and is split into training, validation, and testing sets. We assume all the sets of data are complete. To test missing data, another variable is set to the testing data but excludes one of the missing data combinations. The training and validation data excluding the missing data is used to fit an imputation model to predict the missing data. A classification model is already pretrained using the model that was trained for 5000 epochs and is used to predict classifications on the true test data and the imputed test data. The number of misclassifications, MisRate, and accuracy is recorded for each missing set of data. The test sets had 9,000 samples, validation set had 4,500 samples, and training set had 31,500 samples. The model classification predictions on the true test data are the following: 3 misclassifications, 0.03% MisRate, and 99.97% accuracy.

VI.II One-Way Model: Imputation & Classification Results

The model classification predictions on the imputed test data are shown on Table 36. Can see the metrics for each imputed parameter, the first row means that BURNTIME was the simulated missing parameter that needed to be imputed. The metrics then show how the classifier works when having to impute for the missing parameter. So, when we are in fact missing BURNTIME from the dataset, then the number of misclassified samples increases from 3 samples to 2,186 samples, therefore reducing the accuracy of the classifier to 75.71% from 99.97%. When MAXTHRUST is missing, the number of misclassified samples are 2185 and the accuracy decreases to 75.72%. When MAXPC is missing, the number of misclassified samples are 1237 and the accuracy is 86.26% which is fairly good.

When MAXPE is missing, the number of misclassified samples increases greatly to 3597 samples and the accuracy reduces to 60.03%. It important to note that when we are missing MAXPE, the imputation of MAXPE causes the classification accuracy to reduce significantly. This suggests that MAXPE is a very important parameter to include in the model and because a one-way linear regression is used, the imputation itself is not very accurate and will cause the classification to be worse. When the number of missing parameters increases to two, the number of misclassified samples increase drastically, and the classification accuracy drops below 51%. Finally, when missing three parameters out of four, the classification accuracy completely drops below 22%. Due to the requirements of imputation of the missing parameters, it could be introducing large amounts of noise to the classifier which is causing the classifier to break down. The only way this method will work is if the imputation error can be reduced.

Table 36 Metrics for One-Way Imputation Data

Missing Data	Misclassified	MisRate	Accuracy
BURNTIME	2186	24.29	75.71
MAXTHRUST	2185	24.28	75.72
MAXPC	1237	13.74	86.26
MAXPE	3597	39.97	60.03
BURNTIME, MAXTHRUST	6520	72.44	27.56
BURNTIME, MAXPC	5603	62.26	37.74
BURNTIME, MAXPE	4741	52.68	47.32
MAXTHRUST, MAXPC	4891	54.34	45.66
MAXTHRUST, MAXPE	6782	75.36	24.64
MAXPC, MAXPE	4408	48.98	51.02
BURNTIME, MAXTHRUST, MAXPC	7802	86.69	13.31
BURNTIME, MAXTHRUST, MAXPE	7745	86.06	13.94
BURNTIME, MAXPC, MAXPE	7025	78.06	21.94
MAXTHRUST, MAXPC, MAXPE	7777	86.41	13.59

If the predicted imputation test data is plotted versus the true test data on Fig. 62, can see that there is significant error in the imputation model. Can see that there are obvious distributions of data that are not being modeled correctly and this is due to discontinuities between the 18 different classes. The classes each have a different output distribution, so it would make sense that the linear regression model is not capable of replicating the discontinuities of the output distributions.

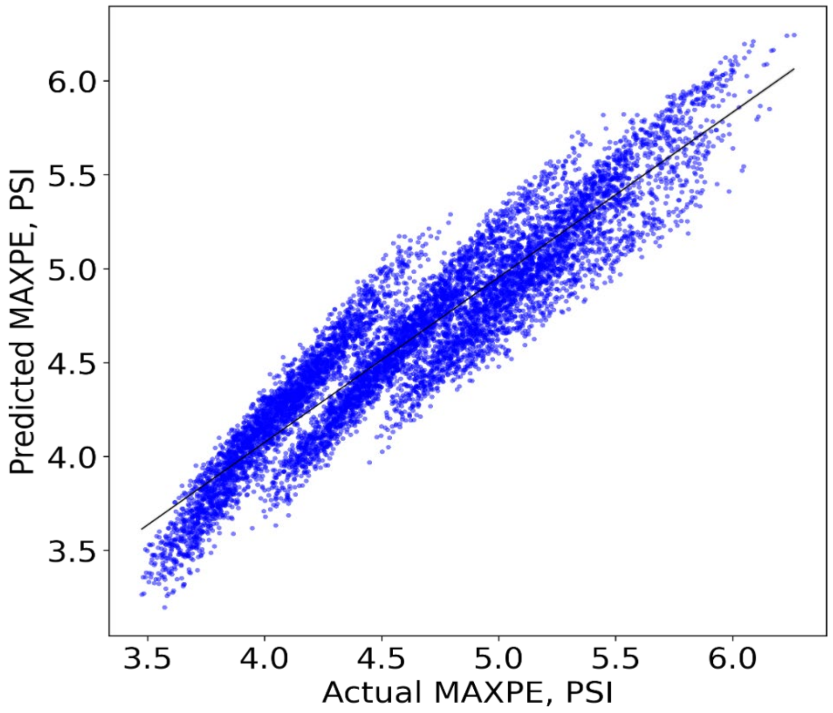


Fig. 61 Imputed MAXPE vs. True MAXPE for a Single Missing Parameter.

If we assess three missing parameters: BURNTIME, MAXTHRUST, & MAXPC, on predicted versus actual plots can see how bad the models are replicating the true data. Figure 63 shows the imputed versus actual for BURNTIME & MAXTHRUST and can see that linear regression model is not capable of replicating the output. MAXPC can be seen to be predicted as two very different distributions compared to the true data. This is because only one single parameter is used to fit the output. It is obvious that there are multiple distributions that cannot be modeled by a linear regression model.

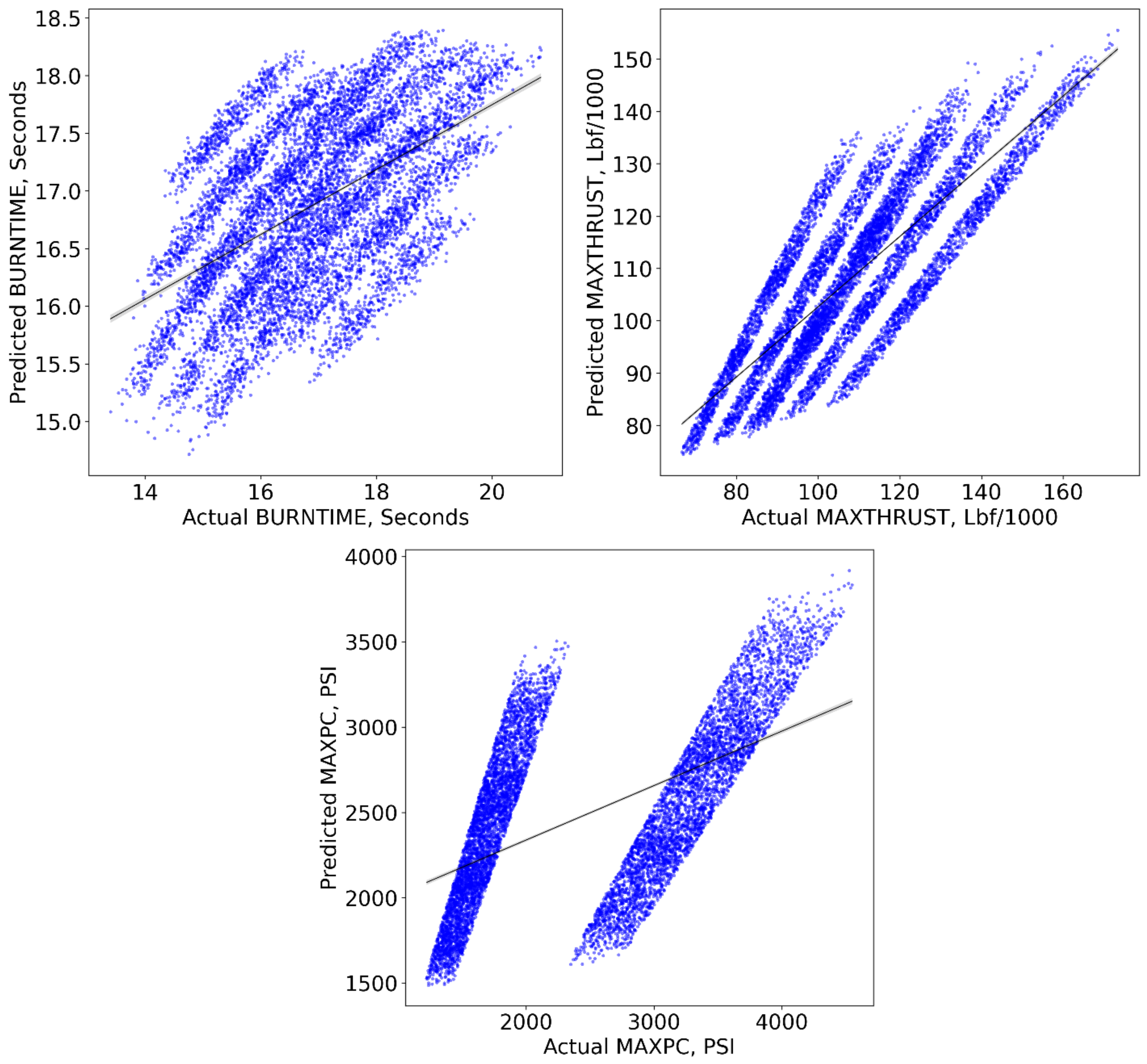


Fig. 62 Imputed vs. Actual for BURNTIME (Top Left), MAXTHRUST (Top Right), & MAXPC (Bottom).

VI.III Two-Way Model: Imputation & Classification Results

It is then obvious that the imputation model order should be increased to include two-way interactions and squared terms. Table 37 shows the metrics for using the two-way imputation model and can see that for single missing parameters that the accuracy does increase by approximately 5% for BURNTIME & MAXTHRUST, 4% for MAXPC, and 10% for MAXPE. However, when increasing the number of missing parameters to two, the accuracy gained is only about 0-5%. When increasing the number of missing parameters to three, there is zero gain in performance. It is obvious to see that when missing three of the four parameters is extremely detrimental to the imputation and classification accuracy.

Table 37 Metrics for Two-Way Imputation Data

Missing Data	Misclassified	MisRate	Accuracy
BURNTIME	1731	19.23	80.77
MAXTHRUST	1742	19.36	80.64
MAXPC	898	9.98	90.02
MAXPE	2758	30.64	69.36
BURNTIME, MAXTHRUST	6517	72.41	27.59
BURNTIME, MAXPC	5506	61.18	38.82
BURNTIME, MAXPE	4414	49.04	50.96
MAXTHRUST, MAXPC	4441	49.34	50.66
MAXTHRUST, MAXPE	6381	70.90	29.10
MAXPC, MAXPE	4324	48.04	51.96
BURNTIME, MAXTHRUST, MAXPC	7888	87.64	12.36
BURNTIME, MAXTHRUST, MAXPE	7764	86.27	13.73
BURNTIME, MAXPC, MAXPE	6935	77.06	22.94
MAXTHRUST, MAXPC, MAXPE	7781	86.46	13.54

If only missing a single parameter such as MAXPC on Fig. 64, then can see that imputation model does fairly well bringing the accuracy up to 90%. The imputed versus actual shows that the imputation model fits well and there is error, but the model is at least fitting the data to an extent.

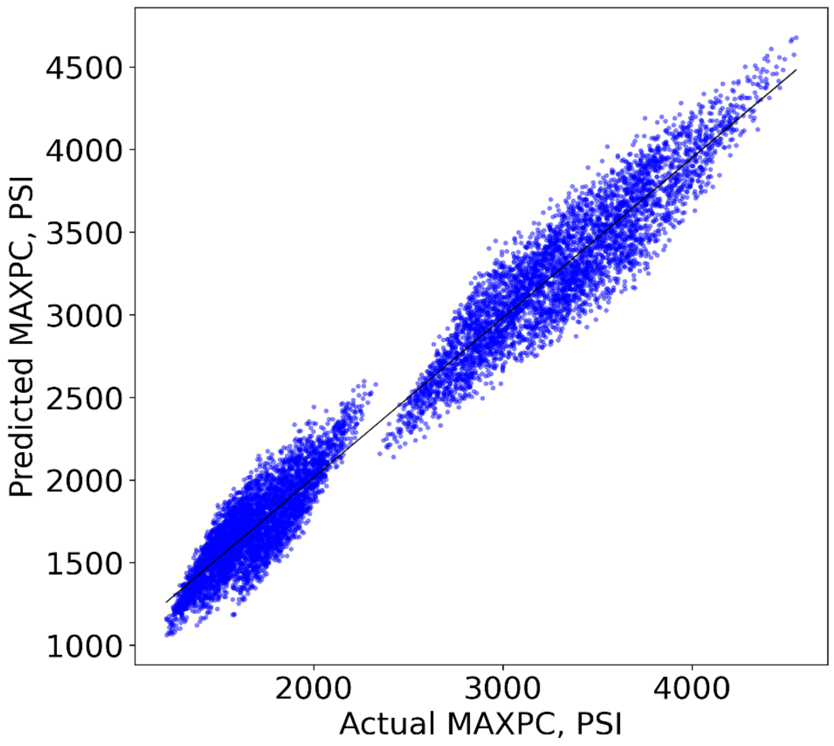


Fig. 63 Imputed vs. Actual for MAXPC.

When the number of missing parameters is increased to three, can see that the classification model is extremely inaccurate. The only explanation is that the building an imputation with only one input variable cannot fully replicate the other three parameters of interest. For example, when looking at Fig. 65 can see that the imputation model is not replicating the data correctly and is very similar to the results on Fig. 63. The only difference from increasing the order is that the imputed results look more curvilinear on MAXTHRUST & MAXPC.

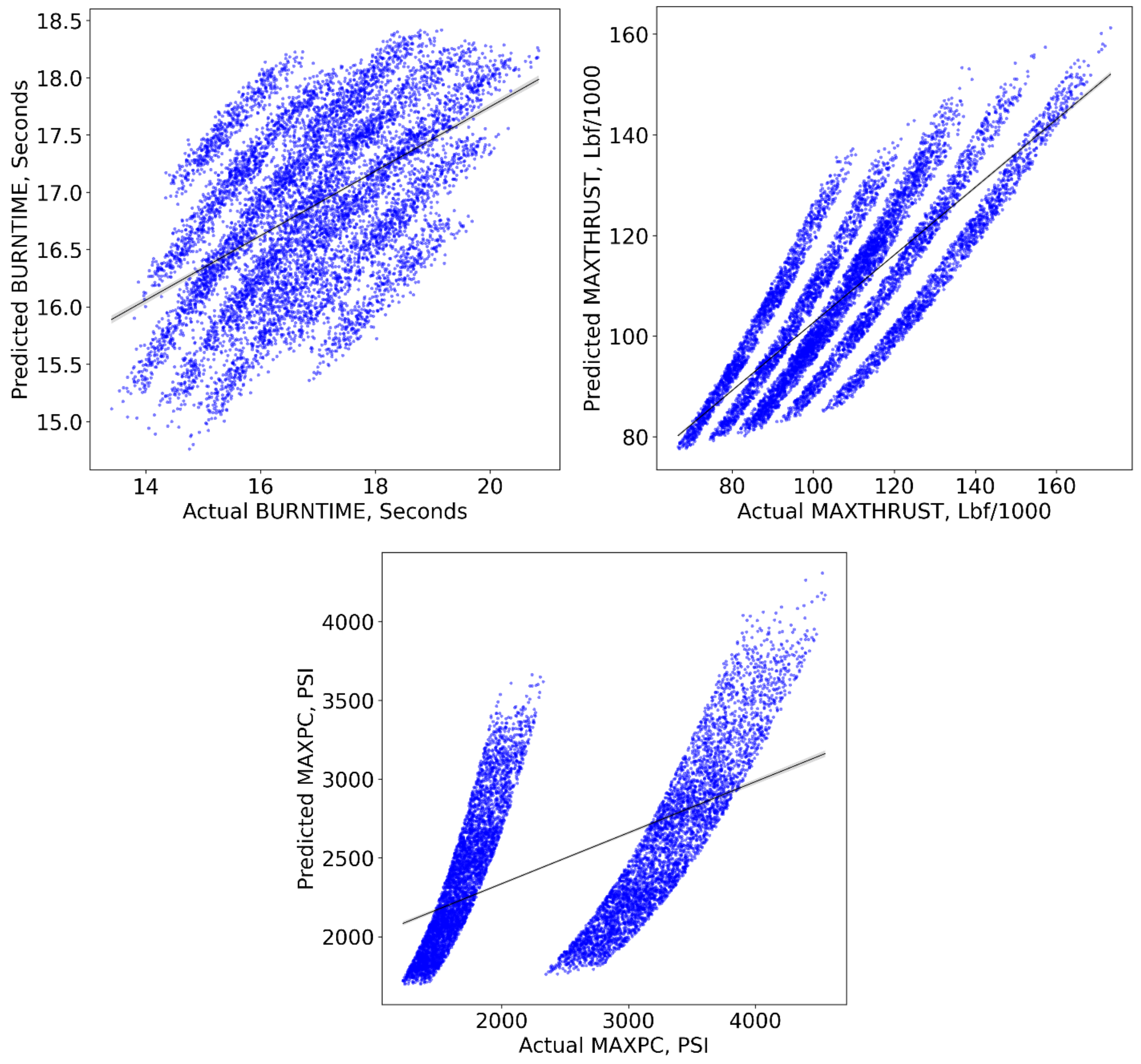


Fig. 64 One-Way Linear Regression Model, Imputed vs. Actual for BURNTIME (Top Left), MAXTHRUST (Top Right), & MAXPC (Bottom).

VI.IV Three-Way Model: Imputation & Classification Results

Finally for linear regression imputation, the order of polynomials is increased to three so there are three-way interactions and includes squared and cubic terms such as x_1^3 and x_2^2. Table 38 shows the metrics of the third order imputation classification model and can see that there is no improvement in accuracy for BURNTIME, MAXTHRUST, and MAXPC. MAXPE does actually increase by 3% in accuracy. There is some accuracy improvement when increasing the number of missing parameters to two. For example, BURNTIME & MAXPE accuracy increases to 58% from 50% in the two-way model; MAXTHRUST & MAXPE accuracy increases to 37% from 29% in the two-way model. There is barely any increase in accuracy when increasing the number of missing parameters to three, which is not surprising. Now it is proven that the linear regression methods can decently replicate the true data if only one parameter is missing, however, when increasing the number of parameters to two the accuracy drops significantly. Finally, when there is only one input to impute three missing parameters the classification accuracy is terrible and should not be used in production.

Table 38 Metrics for Three-Way Imputation Data

Missing Data	Misclassified	MisRate	Accuracy
BURNTIME	1706	18.96	81.04
MAXTHRUST	1711	19.01	80.99
MAXPC	834	9.27	90.73
MAXPE	2514	27.93	72.07
BURNTIME, MAXTHRUST	6503	72.26	27.74
BURNTIME, MAXPC	5455	60.61	39.39
BURNTIME, MAXPE	3747	41.63	58.37
MAXTHRUST, MAXPC	4262	47.36	52.64
MAXTHRUST, MAXPE	5661	62.90	37.10
MAXPC, MAXPE	4114	45.71	54.29
BURNTIME, MAXTHRUST, MAXPC	7882	87.58	12.42
BURNTIME, MAXTHRUST, MAXPE	7588	84.31	15.69
BURNTIME, MAXPC, MAXPE	6906	76.73	23.27
MAXTHRUST, MAXPC, MAXPE	7786	86.51	13.49

The imputed versus actual test data is shown for BURNTIME and MAXPE on Fig. 66 and can see a fairly decent model fit using only two of the four parameters. With BURNTIME there is obviously two distributions of data that must be modeled but is quite erroneous. With MAXPE there is a much better fit overall and can see the model fitting through the true test data. Obviously with both parameters there is error, but the classification model was still able to achieve 58% accuracy, which is better than expected for most real life scenarios.

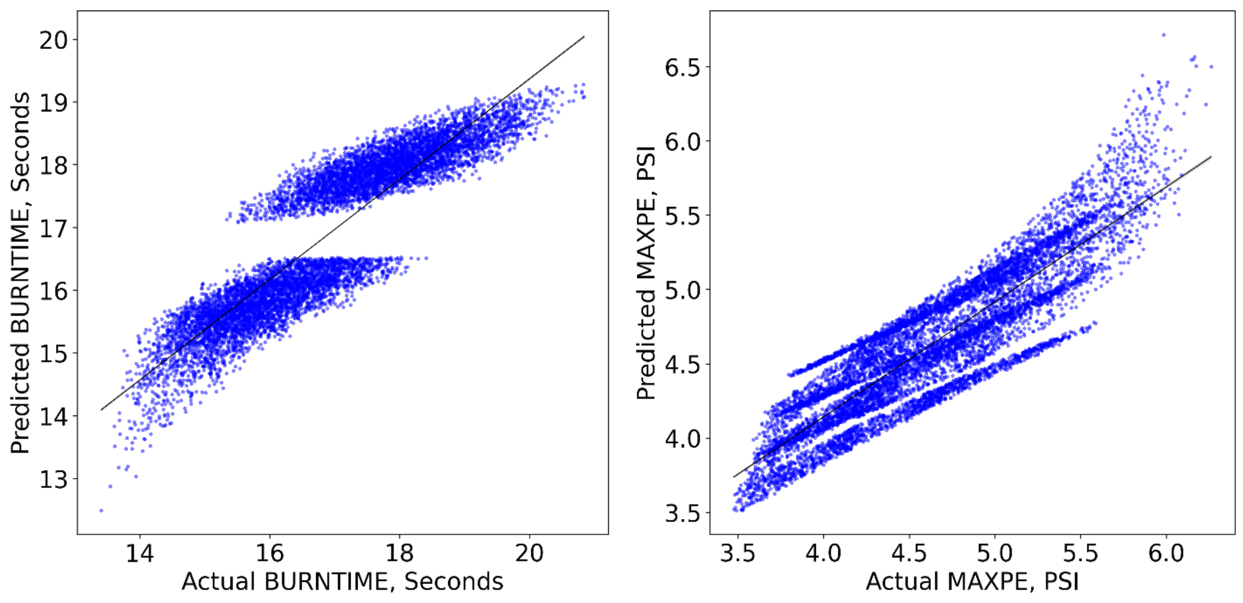


Fig. 65 Two-Way Linear Regression Model, Imputed vs. Actual for BURNTIME (Left) & MAXPE (Right).

Figure 67 shows the imputed versus actual for BURTIME, MAXTHRUST, and MAXPC and can see these predictions are basically the same from Fig. 65. Again, the linear regression methods are NOT capable of replicating the data when only one parameter is available. Therefore, we then attempt to impute the data with higher order nonlinear model using NNETs.

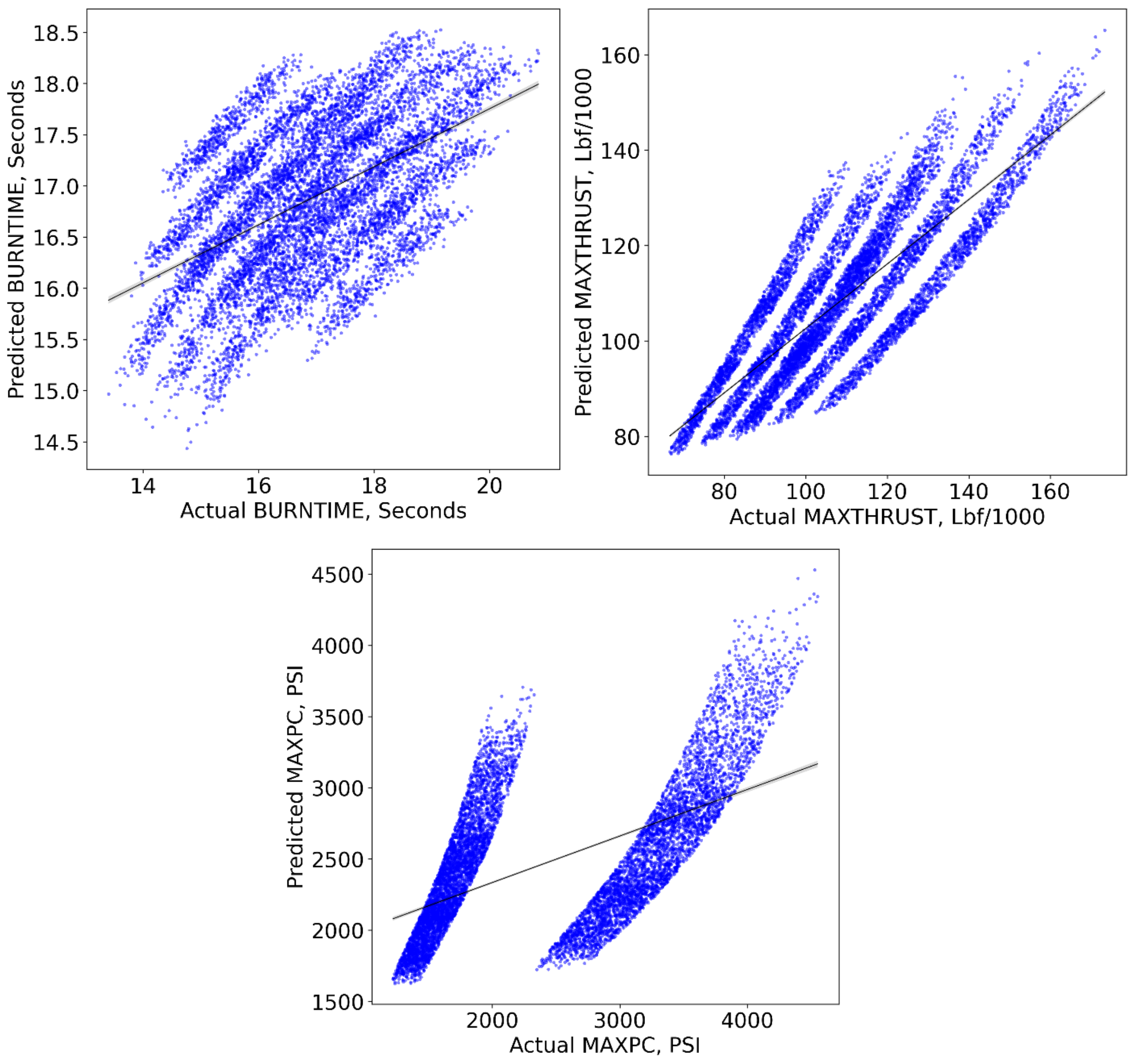


Fig. 66 Three-Way Linear Regression Model, Imputed vs. Actual for BURNTIME (Top Left), MAXTHRUST (Top Right), & MAXPC (Bottom).

VI.V NNET Trained for 1,000 Epochs: Imputation & Classification Results

Instead of using a linear regression for imputation a NNET is developed for imputation. The NNET trained here is a single layer with 60 hidden units with "ELU" and are trained for 1000 epochs using Nadam and MAPE as the loss function. The model with the best validation loss is saved to ensure that the best model is used for imputation. Can see on Table 39 that BURNTIME, MAXTHRUST, & MAXPC do not improve with using the NNET, however, MAXPE does increase by 10%. When increasing to two parameters, some parameters do not increase in accuracy; however, some combinations such as BURNTIME & MAXPC, BURNTIME & MAXPE, MAXTHRUST & MAXPC, MAXTHRUST & MAXPE, and MAXPC & MAXPE increase by about 10% in accuracy. It is then shown that when increasing to three missing parameters, BURNTIME, MAXTHRUST, & MAXPC reduces in accuracy; BURNTIME, MAXTHRUST, & MAXPC increases by about 10%; BURNTIME, MAXPC, & MAXPE does not improve; MAXTHRUST, MAXPC, & MAXPE only improves by about 4%.

Table 39 Metrics for NNET Imputation Data

Missing Data	Misclassified	MisRate	Accuracy
BURNTIME	1712	19.02	80.98
MAXTHRUST	1695	18.83	81.17
MAXPC	1042	11.58	88.42
MAXPE	1466	16.29	83.71
BURNTIME, MAXTHRUST	6471	71.90	28.10
BURNTIME, MAXPC	4760	52.89	47.11
BURNTIME, MAXPE	2743	30.48	69.52
MAXTHRUST, MAXPC	3144	34.93	65.07
MAXTHRUST, MAXPE	3787	42.08	57.92
MAXPC, MAXPE	3090	34.33	65.67
BURNTIME, MAXTHRUST, MAXPC	8367	92.97	7.03
BURNTIME, MAXTHRUST, MAXPE	6846	76.07	23.93
BURNTIME, MAXPC, MAXPE	7001	77.79	22.21
MAXTHRUST, MAXPC, MAXPE	7455	82.83	17.17

Figure 68 shows the imputed versus actual for BURNTIME, MAXTHRUST, & MAXPC and can see the results do not improve with using a NNET. It is now obvious that not even a NNET may be able to impute the full data. The next set of results will then use a larger network similar to the model developed in IV.III.VI Results: 10,000 Epochs.

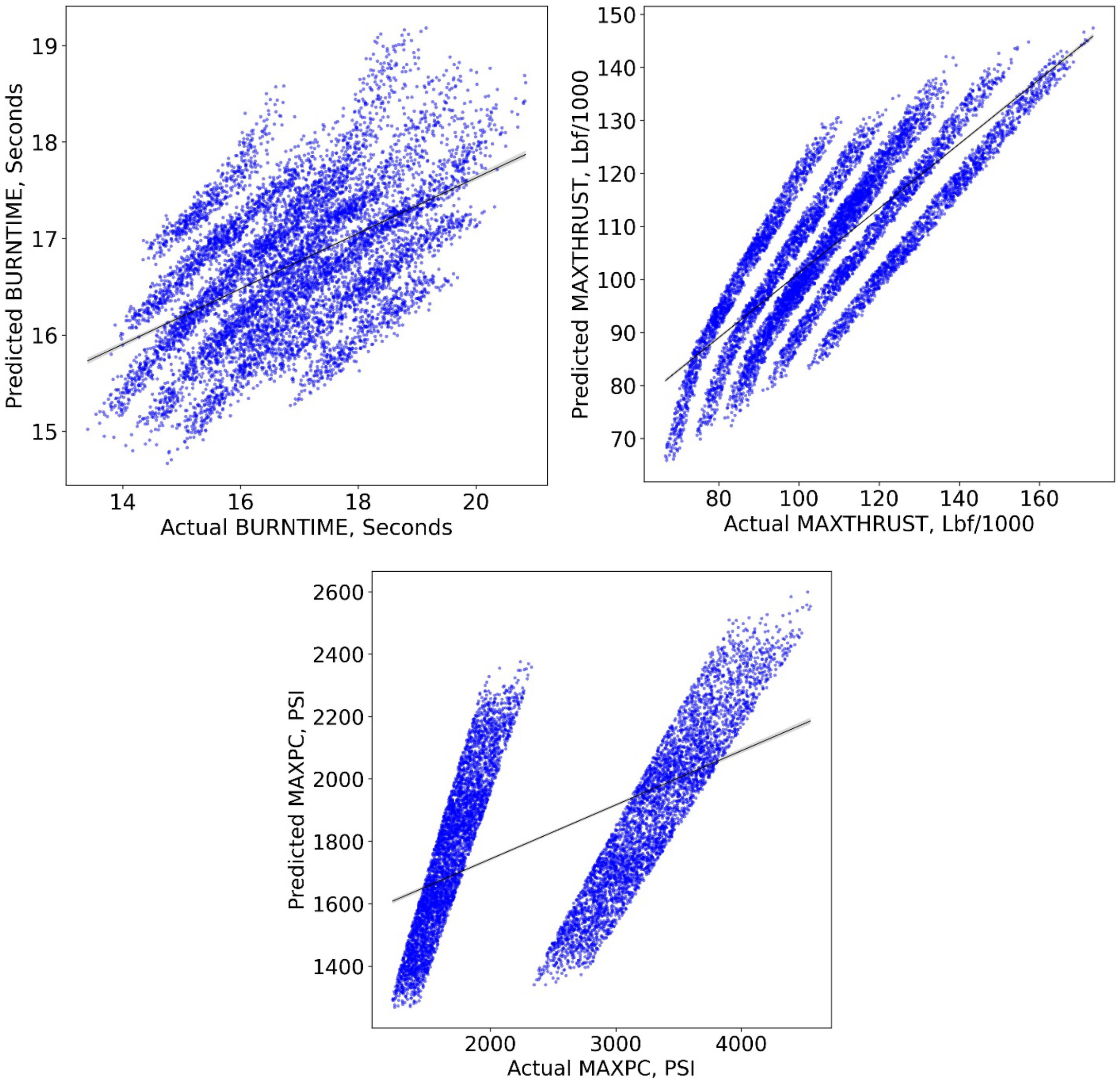


Fig. 67 NNET Trained for 1000 Epochs, Imputed vs. Actual for BURNTIME (Top Left), MAXTHRUST (Top Right), & MAXPC (Bottom).

VI.VI NNET Trained for 5,000 Epochs: Imputation & Classification Results

The model developed here uses three hidden layers with 90 hidden units per layer with ELU activation, Nadam optimization with MAPE as the loss function. Again, the best model with the lowest validation loss is saved during training to ensure the best model is taken. Each imputation network is trained for 5000 epochs. For a single missing parameter, BURNTIME and MAXTHRUST have barely increased in accuracy, but MAXPC and MAXPE have both increased greatly over 96%. When increasing the missing parameters to two, BURNTIME & MAXTHRUST does not increase in accuracy, but the other sets increase quite significantly. MAXPC & MAXPE even increases to almost 90%, which increased from 51% in the one-way linear regression model. When increasing the number of missing parameters to three, the accuracy does not increase at all.

Table 40 Metrics for NNET Trained for 5000 Epochs Imputation Data

Missing Data	Misclassified	MisRate	Accuracy
BURNTIME	1672	18.58	81.42
MAXTHRUST	1511	16.79	83.21
MAXPC	4	0.04	99.96
MAXPE	290	3.22	96.78
BURNTIME, MAXTHRUST	6365	70.72	29.28
BURNTIME, MAXPC	2983	33.14	66.86
BURNTIME, MAXPE	2455	27.28	72.72
MAXTHRUST, MAXPC	2198	24.42	75.58
MAXTHRUST, MAXPE	3488	38.76	61.24
MAXPC, MAXPE	1166	12.96	87.04
BURNTIME, MAXTHRUST, MAXPC	8188	90.98	9.02
BURNTIME, MAXTHRUST, MAXPE	6820	75.78	24.22
BURNTIME, MAXPC, MAXPE	7055	78.39	21.61
MAXTHRUST, MAXPC, MAXPE	7604	84.49	15.51

Figure 69 shows the imputed versus actual for missing BURNTIME, MAXTHRUST, & MAXPC. Can see how the NNET is attempting to model the data and can see how predicted data appears to be making a step. It seems the predicted data increases and then plateaus and it seems that the model is attempting to make the prediction better.

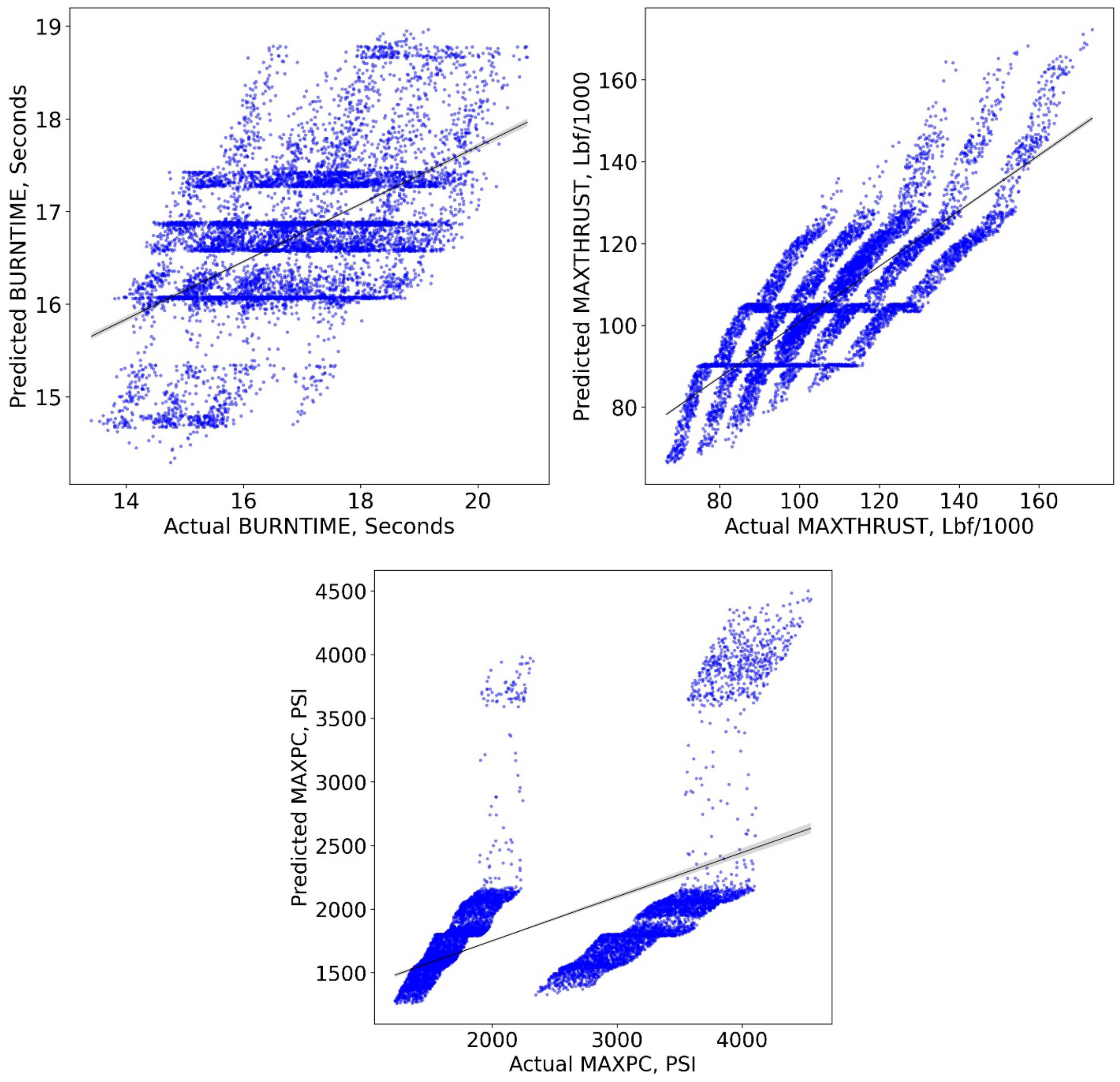


Fig. 68 NNET Trained for 5000 Epochs, Imputed vs. Actual for BURNTIME (Top Left), MAXTHRUST (Top Right), & MAXPC (Bottom).

VI.VII NNET Trained for 20,000 Epochs: Imputation & Classification Results

The imputation models were then trained for 20,000 epochs using the same network configuration from the previous section. Table 41shows the metrics for the NNET trained for 20,000 epochs and can see that the accuracy does not increase. From these results can realize that maybe certain models will improve performance. It was shown that increasing the number of hidden units and hidden layers improves performance. However, this might not ever improve accuracy if only one single parameter is known.

Table 41 Metrics for NNET Trained for 20000 Epochs Imputation Data

Missing Data	Misclassified	MisRate	Accuracy
BURNTIME	1679	18.66	81.34
MAXTHRUST	1502	16.69	83.31
MAXPC	4	0.04	99.96
MAXPE	250	2.78	97.22
BURNTIME, MAXTHRUST	6368	70.76	29.24
BURNTIME, MAXPC	2986	33.18	66.82
BURNTIME, MAXPE	2491	27.68	72.32
MAXTHRUST, MAXPC	2180	24.22	75.78
MAXTHRUST, MAXPE	3534	39.27	60.73
MAXPC, MAXPE	1072	11.91	88.09
BURNTIME, MAXTHRUST, MAXPC	8161	90.68	9.32
BURNTIME, MAXTHRUST, MAXPE	6778	75.31	24.69
BURNTIME, MAXPC, MAXPE	7113	79.03	20.97
MAXTHRUST, MAXPC, MAXPE	7576	84.18	15.82

VI.VIII Future Work for Robustness & Sensitivity

One of the first approaches that can be done is to focus on improving the imputation models by increasing the number of hidden units and hidden layers. It is hypothesized that adding more hidden units may allow the NNET to develop relationships between the known data and simulated missing parameters. Another method, not shown in this work, would be to utilize adversarial learning methods to improve the robustness of the classification models [133]. Currently, there are multiple Python packages for implementing adversarial learning such as CleverHans [134] and Adversarial-Robustness-Toolbox [135]. Both tools allow for training NNETs using adversarial samples to improve the overall robustness to noise.

VII. Model Explainability

As the models were generated in the past several chapters, there was no detail on how the models generate a prediction. For typical linear regression models, such as those in section IV.II Linear Regression Methods and section V.II Linear and Quadratic Discriminant Analysis, the coefficients can be used to assess which parameters have more influence on the model. The downside to just using coefficients is that it only explains globally how parameters affect the output. When the linear regression model is fit, a coefficient will be either positive, zero, or negative and will only show which direction the output goes regardless of what the input is. The coefficients do not say how important a local sample input is to the model prediction. With linear regression and linear discriminant analysis there is least some type of global model explainability; however, with NNETs the weights and biases cannot be directly used to assess which parameters are most influential due to the fact that there are nonlinear activations of the linear combinations of weights and biases. A new method has been developed for evaluating regression methods including linear regression, decision trees & ensemble methods, NNETs, and even agnostic models and it uses an old method of calculating Shapley Values.

VII.I Shapley Values

The concept of Shapley values was first introduced by Lloyd Shapley [136] to measure the contribution of players to a game. This concept was very foundational to game theory as it allows for direct computations of a player's contribution. The overall concept is simple to understand, if there are *n* players, then a player *i*'s contribution can be evaluated by the difference between a game's output including player i and a games output excluding i. To compute player i's overall contribution, multiple games or coalitions *M* are evaluated using a combination of the *N* players which will either include or exclude player *i*. The classical Shapley value estimation [137] is shown on Eq. (7.1), where ϕ_i is the estimated Shapley value for player (input feature) i, $v(M \cup \{i\})$ is the prediction of the model using coalition *M* including feature i, $v(M)$ is the prediction of the model using coalition *M* excluding feature i, and $\gamma_n(M)$ is the weight of proportions to enter coalition *M* [138]. Little m is the number of participants excluding *i* and *n* is the total number of players. $[\mathrm{v}(M \cup \{i\}) - \mathrm{v}(M)]$ is the main portion of calculating the Shapley value and is known as the marginal contribution. The Shapley value can be thought of as the marginal contribution ratioed by the number of coalitions, which is shown on Eq. (7.2), and can see the marginal contribution is just the difference between the sets of coalitions which include and exclude feature *i* [139].

$$\phi_i = \sum_{\text{All } M} \gamma_n(M)\left[v\left(M \cup \{i\}\right) - v(M)\right]$$
$$\gamma_n(M) = \frac{m!(n-m-1)!}{n!} \tag{7.1}$$

$$\phi_i = \frac{1}{n} \sum_{\text{All } M} \frac{\text{Marginal Contribution of } i}{\text{Number of Coalitions excluding } i} \tag{7.2}$$

VII.II Hypothetical Shapley Value Example

The Shapley value concept is best understood by looking at a hypothetical example case. For example, the time of flight (TOF) is modeled using DBODY, PC, and BURNTIME and we wish to obtain the contributions (Shapley Values) that DBODY has on the model. To do this, TOF must be modeled using each coalition available, and since there 3 input features there are $M = 2^n = 2^3 = 8$ coalitions required.

Table 42 Hypothetical Coalitions for TOF

Coalitions	Players	$v(M)$ TOF Prediction, Seconds
1	None ($\emptyset$) (No Input or Zero)	0
2	DBODY	15
3	PC	25
4	BURNTIME	30
5	DBODY, PC	45
6	DBODY, BURNTIME	35
7	PC, BURNTIME	55
8	DBODY, PC, BURNTIME	95

Table 42 shows the hypothetical results of TOF being modeled by each coalition. For example , can see on coalition 1 that if TOF has no input then there is no predicted TOF; cannot make a model prediction with no input. If TOF is modeled using coalition 2, then there is a TOF prediction of 15 seconds using just DBODY. To calculate the Shapley value for DBODY, there are then $M/2 = 8/2 = 4$ sets in the summation of Eq. (7.1). Equation (7.3) shows the summation of the sets to calculate the Shapley value for DBODY.

$$\phi(DBODY) = \sum_{M \cup i}^{4} \frac{m!(n-m-1)!}{n!}\left[v\left(M \cup \{i\}\right) - v(M)\right] \tag{7.3}$$

Equation (7.3) is expanded to show the four sets required to calculate the Shapley Value on Eq. (7.4). The first set is the model using DBODY subtracting the model with no inputs ($\emptyset$). The second set is the coalitions with the first part using DBODY & PC and second part just using PC (excludes DBODY). The third set is the coalition with the first part using DBODY & BURNTIME and second part using just BURNTIME (excludes DBODY). The fourth final set is the coalition of DBODY, PC, & BURNTIME and the second part using PC & BURNTIME (excludes DBODY).

$$\begin{aligned}\phi(DBODY) = {} & \gamma(\varnothing)\left[v(DBODY)-v(\varnothing)\right]+ \\ & \gamma(PC)\left[v(DBODY,PC)-v(PC)\right]+ \\ & \gamma(BURNTIME)\left[v(DBODY,BURNTIME)-v(BURNTIME)\right]+ \\ & \gamma(PC,BURNTIME)\left[v(DBODY,PC,BURNTIME)-v(PC,BURNTIME)\right]\end{aligned} \tag{7.4}$$

The values on Table 42 can then be plugged into Eq. (7.4) to give Eq. (7.5). For this example, *n* is three for DBODY, PC, & BURNTIME. *m* is zero for the first set, *m* is one for both the second and third set, which refer to PC and BURNTIME, and the fourth set *m* is two for PC & BURNTIME.

$$\begin{aligned}\phi(DBODY) = {} & \frac{0!(3-0-1)!}{3!}\left[15-0\right]+\frac{1!(3-1-1)!}{3!}\left[45-25\right]+ \\ & \frac{1!(3-1-1)!}{3!}\left[35-30\right]+\frac{2!(3-2-1)!}{3!}\left[95-55\right]\end{aligned} \tag{7.5}$$

Simplifying Eq. (7.5) into Eq. (7.6) finally shows the Shapley value for DBODY. Notice the Shapley value is in units of the output feature, so it is in units of seconds for TOF. This shows that DBODY has an average contribution of 22.50 seconds to TOF.

$$\begin{aligned}\phi(DBODY) &= \frac{1*2!}{3*2!}\left[15\right]+\frac{1*1!}{3!}\left[20\right]+\frac{1*1!}{3!}\left[5\right]+\frac{2!*0!}{3*2!}\left[40\right] \\ \phi(DBODY) &= \frac{1}{3}\left[15\right]+\frac{1}{6}\left[20\right]+\frac{1}{6}\left[5\right]+\frac{1}{3}\left[40\right] \\ \phi(DBODY) &= 5+3.3333+0.8333+13.3333=22.50\text{ Seconds}\end{aligned} \tag{7.6}$$

The calculations of the Shapley values for PC and BURNTIME are shown in Appendix M: Hypothetical Shapley Value Example. In the upcoming sections, the Shapley value concept is extended to linear regression and a few methods for approximating the Shapley Values are introduced.

VII.III Shapley Values Related to Linear Regression

Shapley values can be utilized for multiple statistical and machine learning models. The Shapley values were applied to linear regression modeling functions to compare against the model standardized coefficients [140][141]. Can see that the Shapley value for the i^{th} input is simply the standardized coefficient times the difference between the input and the input average, which is shown on Eq. (7.7).

$$\phi_i\left(\hat{f}(x)\right) = \beta_i x_i - E\left[\beta_i x_i\right] = \beta_i\left(x_i - E\left[x_i\right]\right) \tag{7.7}$$

If the Shapley values are summed for each standardized coefficients, then the right hand side of Eq. (7.7) can also be summed and reduced to the difference between the model prediction and the average of the output feature, which is shown on Eq. (7.8).

$$\begin{aligned}
\sum_{i=1}^{n}\phi_i\left(\hat{f}\right) &= \sum_{i=1}^{n}\left(\beta_i x_i - E\left[\beta_i x_i\right]\right) \\
\sum_{i=1}^{n}\phi_i\left(\hat{f}\right) &= \left(\beta_0 + \sum_{i=1}^{n}\beta_i x_i\right) - \left(\beta_0 + \sum_{i=1}^{n}E\left[\beta_i x_i\right]\right) \\
\sum_{i=1}^{n}\phi_i\left(\hat{f}\right) &= \hat{f}(x) - E\left[\hat{f}(x)\right]
\end{aligned} \tag{7.8}$$

VII.IV Shapley Value Approximation Methods

The classical Shapley calculation shown on Eq. (7.1) and its implementation in the hypothetical example on section VII.II Hypothetical Shapley Value Example are not actually practical for computational reasons. Since there are $\mathrm{M} = 2^n$ coalitions required, for the regression database with 15 features, then there are $\mathrm{M} = 2^n = 2^{15} = 32{,}768$ coalitions required. For the dataset including fin autopilot parameters, there are *n*=26 inputs, so there $\mathrm{M} = 2^n = 2^{26} = 67{,}108{,}864$ coalitions required. It is obvious to see that number of coalitions explodes as more input features are added. This also does not include evaluating multiple samples, which would then be multiplied by the number of coalitions required. So, the classical Shapley analysis is very impractical for determining the contributions computationally.

A powerful python package has been developed which includes many approximation methods for linear regression models using Linear SHAP [137][140][141], decision trees & ensemble models using Tree SHAP [142][143], NNETs using Deep Lift SHAP [137][144][145], and model agnostic functions using Kernel SHAP [137]. This package is called SHAP [137] (Shapley Additive Explanation), is available for free from Anaconda. The algorithms are explained in details in the references listed, but if the model is a linear regression function then SHAP identifies that it is a Scikit-Learn model and then a variation of Eq. (7.7) can be used to calculate the SHAP values, shown on Eq. (7.9) where *j* refers to the sample number and *i* is still the feature input. Can then see how a local sample SHAP value $\left(\phi_{ij}\right)$ is determined and can see how the global SHAP value (ϕ_i) is derived.

$$\phi_{ij} = \beta_i \left(x_{ij} - E\left[x_i \right] \right)$$

$$\phi_i = \frac{1}{n} \sum_{i=1, j=1}^{n} \left| \phi_{ij} \right| \tag{7.9}$$

VII.V Regression Explanations

To explain these the SHAP package is used for linear regression (excluding Ridge & Lasso) and regression NNETs. SHAP allows using Scikit-Learn and TensorFlow to easily calculate the SHAP values necessary. Word of caution, the NNET version can be very slow if a lot of samples are used. For this data, if more than 5000 samples are used it gets very slow and a warning will mostly appear. Although not used here, the Kernel SHAP version can also be very slow as well. The Linear SHAP is extremely Fast and there should be no issues when executing this version.

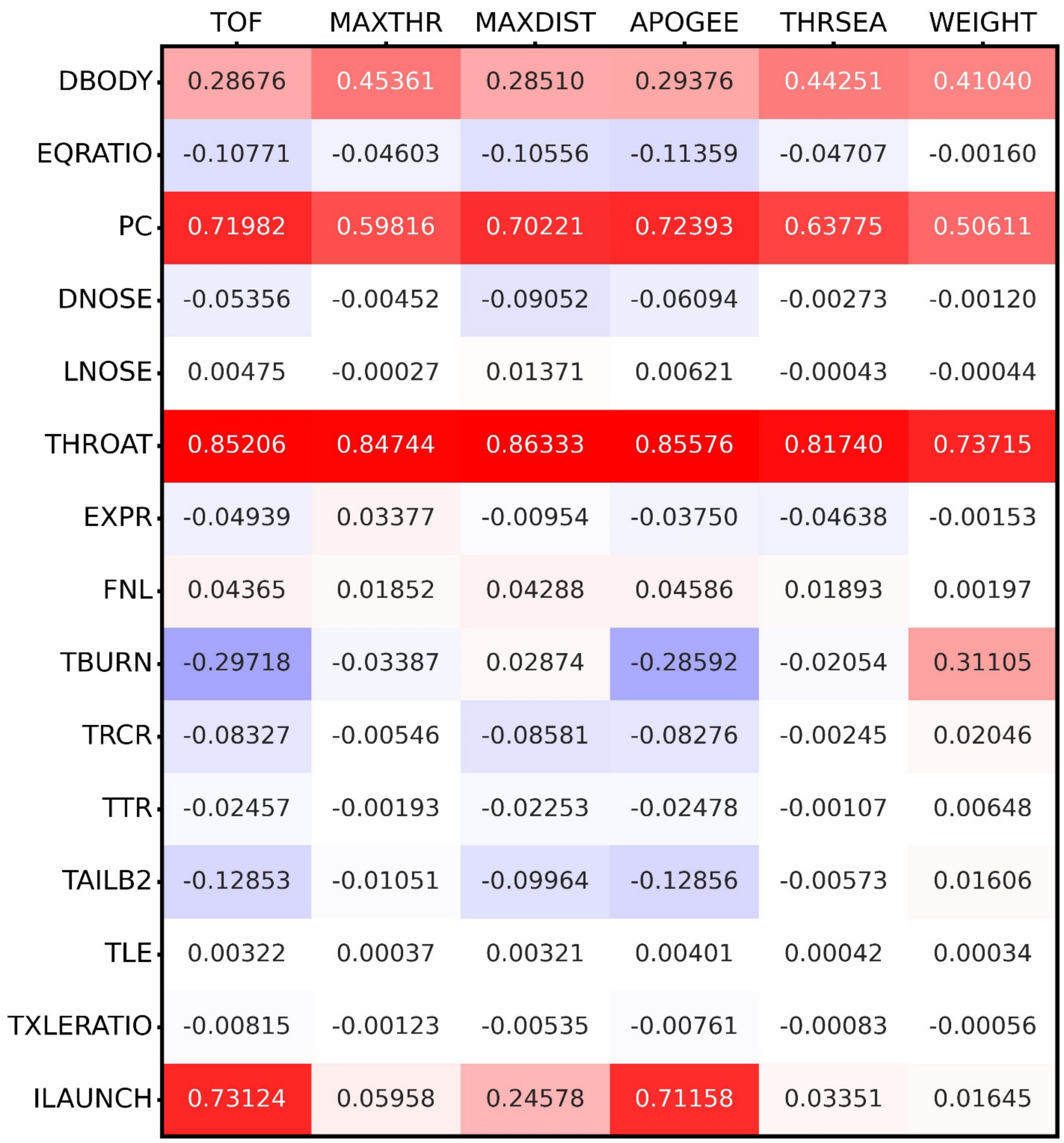
VII.V.I Linear Regression & Shapley Value Global Average

First, we look at the linear regression model standardized coefficients and compare them to the global average SHAP values. Can see that the standardized coefficients are easily calculated by multiplying the coefficient to the ratio of the input feature standard deviation and the target feature standard deviation, which is shown on Eq. (7.10) First, the standardized coefficients of the linear regression model using one-way terms are plotted on a heatmap on Fig. 70. The standardized coefficients are shown for each input feature along the output feature columns. When evaluating the TOF, can see that THROAT has the highest positive contribution to TOF, followed by ILAUNCH, PC, and DBODY, but TBURN has the highest negative contribution followed by

TAILB2 and EQRATIO. Parameters which are positive increase the value of TOF and parameters which are negative decrease TOF. So, it can be said DBODY is positively contributing to TOF and TBURN is negatively contributing to TOF.

$$\beta_i^* = \beta_i \frac{\sigma(X_1)}{\sigma(y)} \tag{7.10}$$

When looking at MAXTHR, can see that THROAT has the largest positive contribution, followed by PC and DBODY. Will then notice that the other parameters do not really contribute to MAXTHR, which makes sense because the tail fin parameters would not contribute to thrust in any way. For MAXDIST, THROAT has the largest positive contribution, followed by PC, DBODY, and ILAUNCH. EQRATIO has the largest negative contribution, followed by TAILB2, DNOSE, and TRCR. For APOGEE, THROAT has the largest positive contribution followed by PC, ILAUNCH, and DBODY; TBURN has the largest negative contribution followed by TAILB2, EQRATIO, TRCR, and DNOSE. For THRSEA, THROAT has the largest positive contribution, followed by PC and DBODY, and notice that this follows the same trend as MAXTHR. For WEIGHT, THROAT has the largest positive contribution, followed by PC, DBODY, and TBURN. THROAT can be hard parameter to understand as to why it has such a large contribution to WEIGHT. When looking at Eq. (2.6), can see that the mass flow rate is function of PC, THROAT, and the propellant characteristic velocity. The total propellant mass is then calculated from Eq. (2.7), which is a function of mass flow rate and BURNTIME. Therefore, propellant mass is a function of BURNTIME, PC, and THROAT, which will account for 60-70% of the total mass.

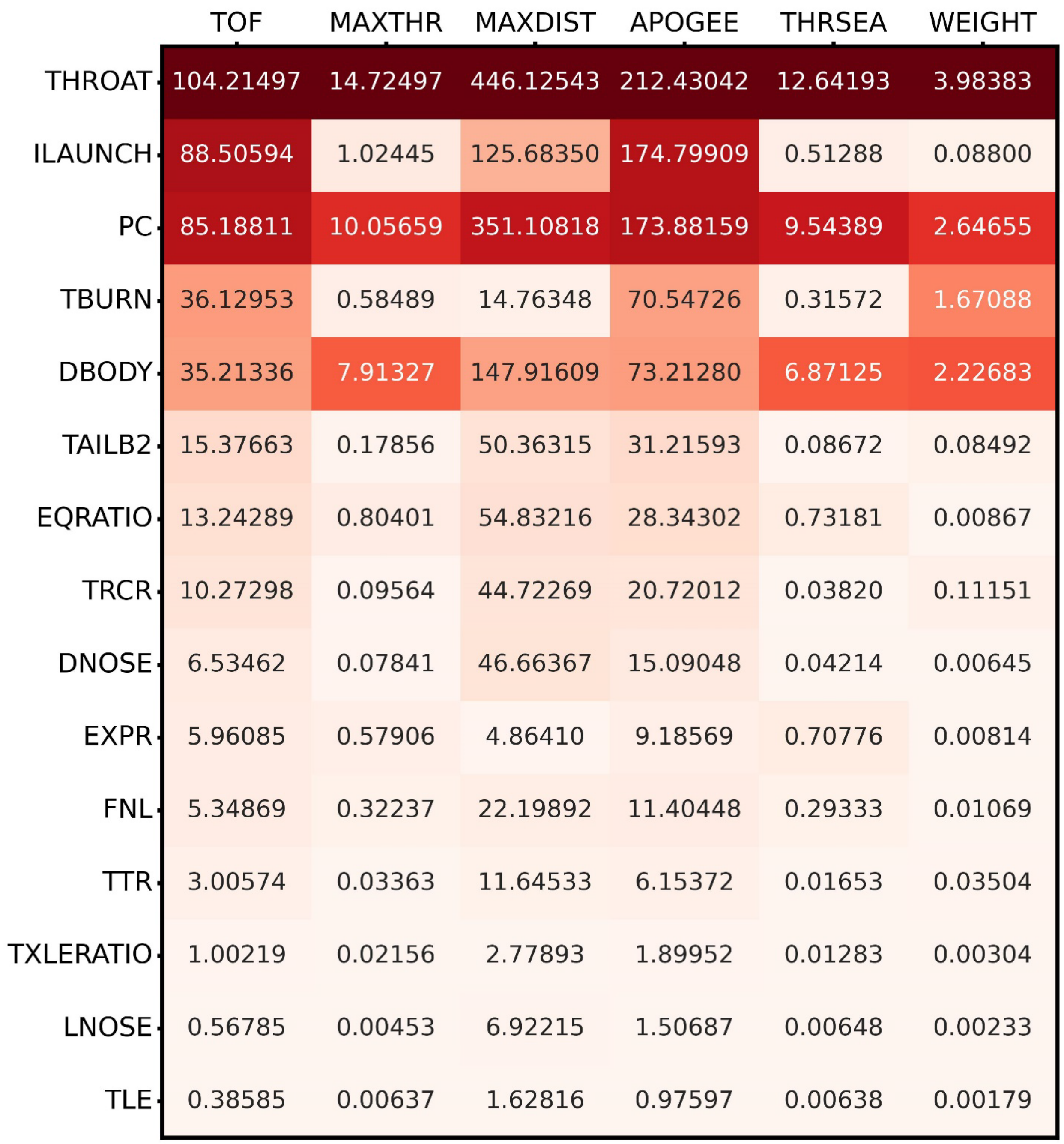


	TOF	MAXTHR	MAXDIST	APOGEE	THRSEA	WEIGHT
DBODY	0.28676	0.45361	0.28510	0.29376	0.44251	0.41040
EQRATIO	-0.10771	-0.04603	-0.10556	-0.11359	-0.04707	-0.00160
PC	0.71982	0.59816	0.70221	0.72393	0.63775	0.50611
DNOSE	-0.05356	-0.00452	-0.09052	-0.06094	-0.00273	-0.00120
LNOSE	0.00475	-0.00027	0.01371	0.00621	-0.00043	-0.00044
THROAT	0.85206	0.84744	0.86333	0.85576	0.81740	0.73715
EXPR	-0.04939	0.03377	-0.00954	-0.03750	-0.04638	-0.00153
FNL	0.04365	0.01852	0.04288	0.04586	0.01893	0.00197
TBURN	-0.29718	-0.03387	0.02874	-0.28592	-0.02054	0.31105
TRCR	-0.08327	-0.00546	-0.08581	-0.08276	-0.00245	0.02046
TTR	-0.02457	-0.00193	-0.02253	-0.02478	-0.00107	0.00648
TAILB2	-0.12853	-0.01051	-0.09964	-0.12856	-0.00573	0.01606
TLE	0.00322	0.00037	0.00321	0.00401	0.00042	0.00034
TXLERATIO	-0.00815	-0.00123	-0.00535	-0.00761	-0.00083	-0.00056
ILAUNCH	0.73124	0.05958	0.24578	0.71158	0.03351	0.01645

Fig. 69 One-Way Parameter Linear Model: Standardized Coefficients.

Figure 71 shows the one-way linear regression global SHAP values. The global SHAP values are just the mean absolute SHAP values and shows the average contribution for an input feature. For TOF, can see that THROAT has the large average contribution, so on average

THROAT contribute 104.21497 seconds to TOF. ILAUNCH, PC, TBURN, DBODY, TAILB2, and EQRATIO are on average the largest contributors for TOF. Notice that SHAP values are



	TOF	MAXTHR	MAXDIST	APOGEE	THRSEA	WEIGHT
THROAT	104.21497	14.72497	446.12543	212.43042	12.64193	3.98383
ILAUNCH	88.50594	1.02445	125.68350	174.79909	0.51288	0.08800
PC	85.18811	10.05659	351.10818	173.88159	9.54389	2.64655
TBURN	36.12953	0.58489	14.76348	70.54726	0.31572	1.67088
DBODY	35.21336	7.91327	147.91609	73.21280	6.87125	2.22683
TAILB2	15.37663	0.17856	50.36315	31.21593	0.08672	0.08492
EQRATIO	13.24289	0.80401	54.83216	28.34302	0.73181	0.00867
TRCR	10.27298	0.09564	44.72269	20.72012	0.03820	0.11151
DNOSE	6.53462	0.07841	46.66367	15.09048	0.04214	0.00645
EXPR	5.96085	0.57906	4.86410	9.18569	0.70776	0.00814
FNL	5.34869	0.32237	22.19892	11.40448	0.29333	0.01069
TTR	3.00574	0.03363	11.64533	6.15372	0.01653	0.03504
TXLERATIO	1.00219	0.02156	2.77893	1.89952	0.01283	0.00304
LNOSE	0.56785	0.00453	6.92215	1.50687	0.00648	0.00233
TLE	0.38585	0.00637	1.62816	0.97597	0.00638	0.00179

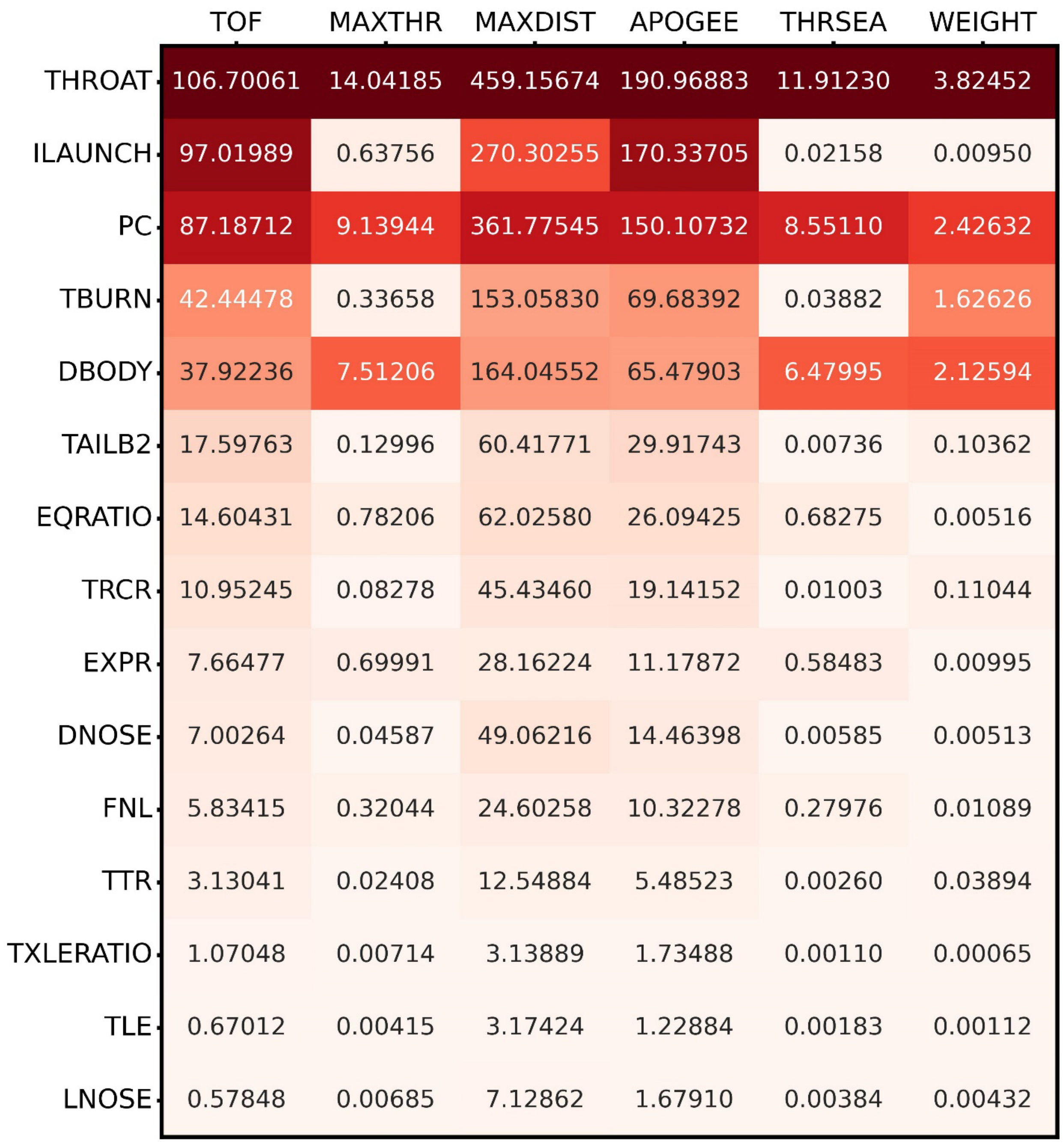
Fig. 70 One-Way Linear Regression Mean(|SHAP|).

similar in magnitude ordering, so ILAUNCH was the second most contributing on Fig. 70 and on Fig. 71. PC & TBURN are the third and fourth, respectively, most contributing parameters on both

Fig. 70 and Fig. 71. So, can see that the SHAP values for a linear model are just scaled values of the standardized coefficients. For MAXTHR, can see that THROAT is the most contributing (also on Fig. 70), followed by PC (as it is on Fig. 70), and DBODY (as it is on Fig. 70). For MAXDIST, THROAT, ILAUNCH, PC, DBODY, EQRATIO, and TAILB2 are the most contributing parameters. For APOGEE, THROAT, ILAUNCH, PC, TBURN, and DBODY are the most contributing parameters. For THRSEA, THROAT ,PC, and DBODY are the most contributing. For WEIGHT, THROAT, PC, TBURN, and DBODY are the most contributing parameters. For WEIGHT, THROAT, PC, TBURN, and DBODY are the most contributing parameters.

VII.V.II Neural Network Regression & Shapley Value Explanations

Next, we can apply the SHAP package using the "DeepExplainer" function to obtain the SHAP values for NNET using TensorFlow. Figure 72 shows the mean absolute global contributions (SHAP values) for the NNET that was trained for 50,000 epochs in Sect. IV.III.VII. Notice that global SHAP values are almost one to one the same as the SHAP values for the linear regression model on Fig. 71. Will notice that sense the NNET was able to get improved metrics, the SHAP values are slightly different and much greater for ILAUNCH. For TOF & MAXDIST, will notice that ILAUNCH has a larger contribution, and for the rest of the outputs they are a bit smaller. For MAXDIST, can see that TBURN has a much larger contribution than it did for the linear regression model on Fig. 71 and in fact, many of the parameters for MAXDIST have increase with few parameters decreasing. There is some change on TOF & APOGEE and very small changes on MAXTHR, THRSEA, and WEIGHT.



	TOF	MAXTHR	MAXDIST	APOGEE	THRSEA	WEIGHT
THROAT	106.70061	14.04185	459.15674	190.96883	11.91230	3.82452
ILAUNCH	97.01989	0.63756	270.30255	170.33705	0.02158	0.00950
PC	87.18712	9.13944	361.77545	150.10732	8.55110	2.42632
TBURN	42.44478	0.33658	153.05830	69.68392	0.03882	1.62626
DBODY	37.92236	7.51206	164.04552	65.47903	6.47995	2.12594
TAILB2	17.59763	0.12996	60.41771	29.91743	0.00736	0.10362
EQRATIO	14.60431	0.78206	62.02580	26.09425	0.68275	0.00516
TRCR	10.95245	0.08278	45.43460	19.14152	0.01003	0.11044
EXPR	7.66477	0.69991	28.16224	11.17872	0.58483	0.00995
DNOSE	7.00264	0.04587	49.06216	14.46398	0.00585	0.00513
FNL	5.83415	0.32044	24.60258	10.32278	0.27976	0.01089
TTR	3.13041	0.02408	12.54884	5.48523	0.00260	0.03894
TXLERATIO	1.07048	0.00714	3.13889	1.73488	0.00110	0.00065
TLE	0.67012	0.00415	3.17424	1.22884	0.00183	0.00112
LNOSE	0.57848	0.00685	7.12862	1.67910	0.00384	0.00432

Fig. 71 NNET Regression Mean(|SHAP|).

Next, the model global SHAP values can be plotted using the SHAP "Summary Plot" function, which plots the SHAP values from Fig. 72 for each output (the values on along the row) as a stacked bar chart. Figure 73 shows the global mean(|SHAP|) values plotted, where each bar is the summed mean(|SHAP|) values from each row on Fig. 72. For the NNET regression model, can see that THROAT is globally the largest contributor to the model. Will notice that MAXTHR, THRSEA, and WEIGHT are small in magnitude compared to MAXDIST ,APOGEE, and TOF, but is not an issue. Part of the reason the magnitude is small for MAXTHR, THRSEA, and

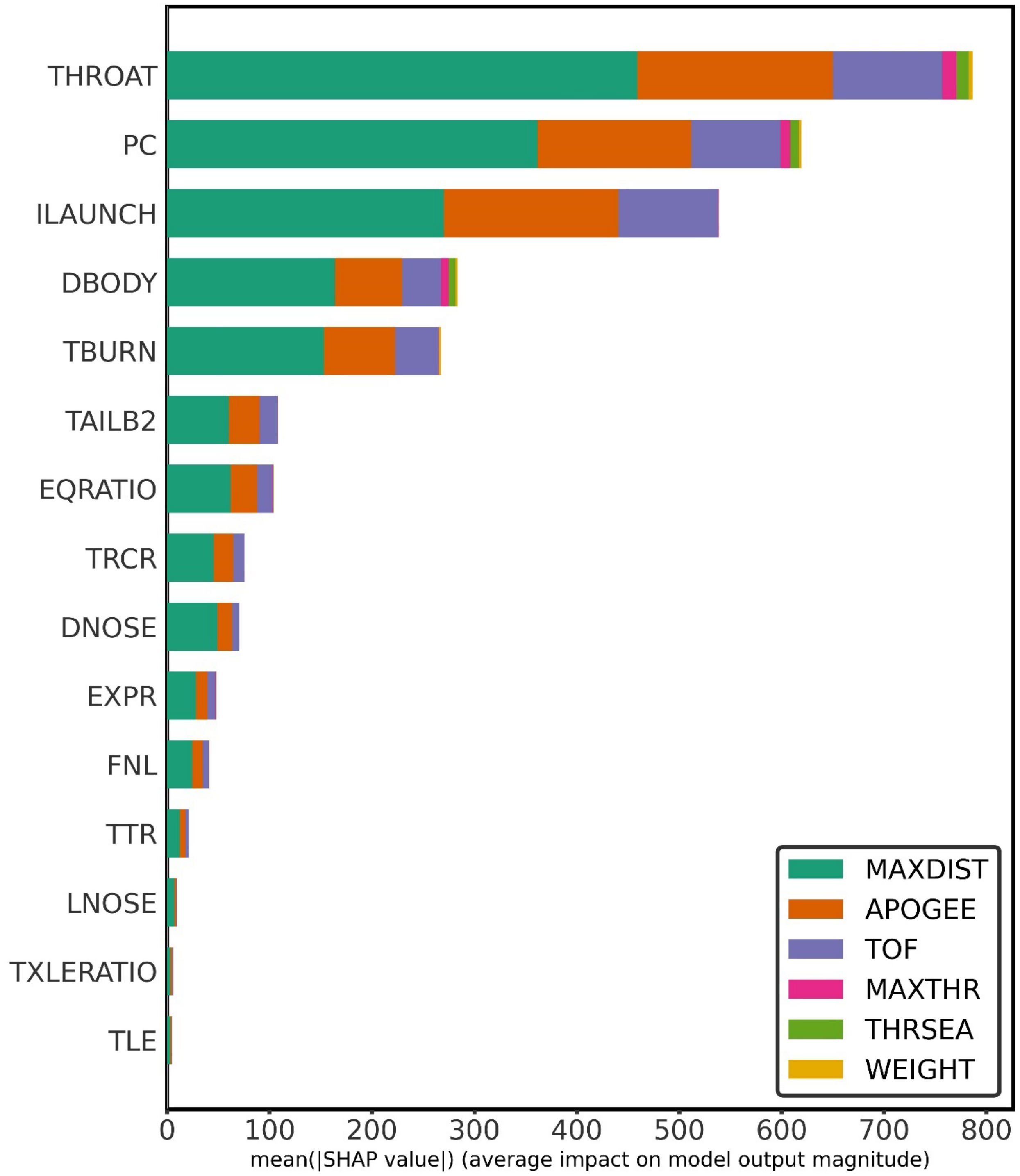


Fig. 72 NNET Global Mean (|SHAP|) Values Stacked Bar Chart.

WEIGHT is just due to the fact that the ranges are also smaller compared to MAXDIST ,APOGEE, and TOF. Will then notice that PC is the second most contributing, followed by ILAUNCH, DBODY, TBURN, TAILB2, EQRATIO, TRCR, DNOSE, EXPR, FNL, TTR, LNOSE, TXLERATIO, and TLE. The only issues with utilizing figures 72-73, is that both figures display the mean absolute SHAP values and so there is loss of effect for single samples, which can cause one to misconstrue the model input feature contributions. One must take caution in interpreting the SHAP values and for figures 72-73, one must remember that the values refer to the mean absolute SHAP values. For example, for TOF, THROAT has an AVERAGE SHAP (contribution) value of 106.7 seconds to the NNET model. So, not every sample will have 106.7 seconds of contribution. Therefore, it is important to display sample input SHAP available from using the "DeepExplainer.shap_values(X)" function call.

Figure 74 shows the individual sample SHAP values for the output parameter TOF. The samples are colored blue to red, which displays its magnitude from low (blue) to high (red), respectively. THROAT for example, shows the lower inputs are blue and the higher inputs are red. Can then see SHAP value on the x-axis and will see if it is negative or positive. This plot is extremely useful because we can see the sample SHAP value and determine how its input value is scaled. Looking back at THROAT, we can see that lower values of THROAT have a negative contribution to the output and higher values of THROAT have a positive contribution to the output. ILAUNCH, PC, DBODY, and FNL all show that their lower respective values have negative contributions, and their higher respective values have positive contributions. Looking at TBURN, the opposite effect occurs and can see that LOWER values of TBURN have a POSITIVE contribution and HIGHER values of TBURN have a NEGATIVE contribution. TAILB2, EQRATIO, TRCR, EXPR, DNOSE, and TTR show that their lower respective values have a positive contribution, whereas their higher respective values have a negative contribution. Notice that TXLERATIO, TLR, and LNOSE have an extremely low contribution.

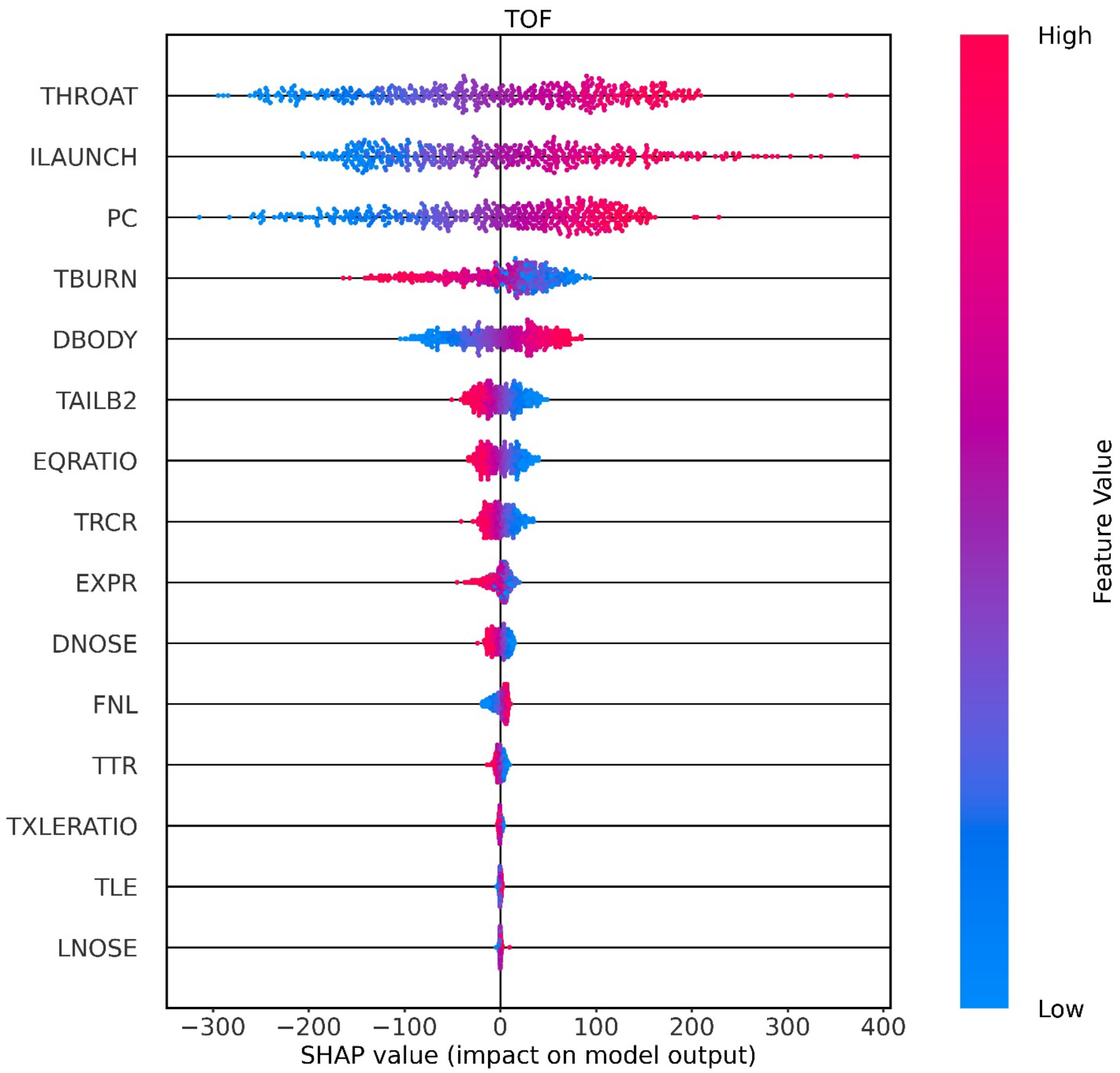


Fig. 73 TOF Sample SHAP Values using Summary Plot.

Figure 75 shows the individual sample SHAP values for the output parameter MAXTHR. Will see that for THROAT, PC, DBODY, EXPR, and ILAUNCH show that their lower respective values have a negative contribution, and their higher respective values have a positive contribution. For EQRATIO and TBURN, their higher respective values have a negative contribution and their lower respective values have a positive contribution. The parameters below and including FNL, all have extremely low contributions to the model.

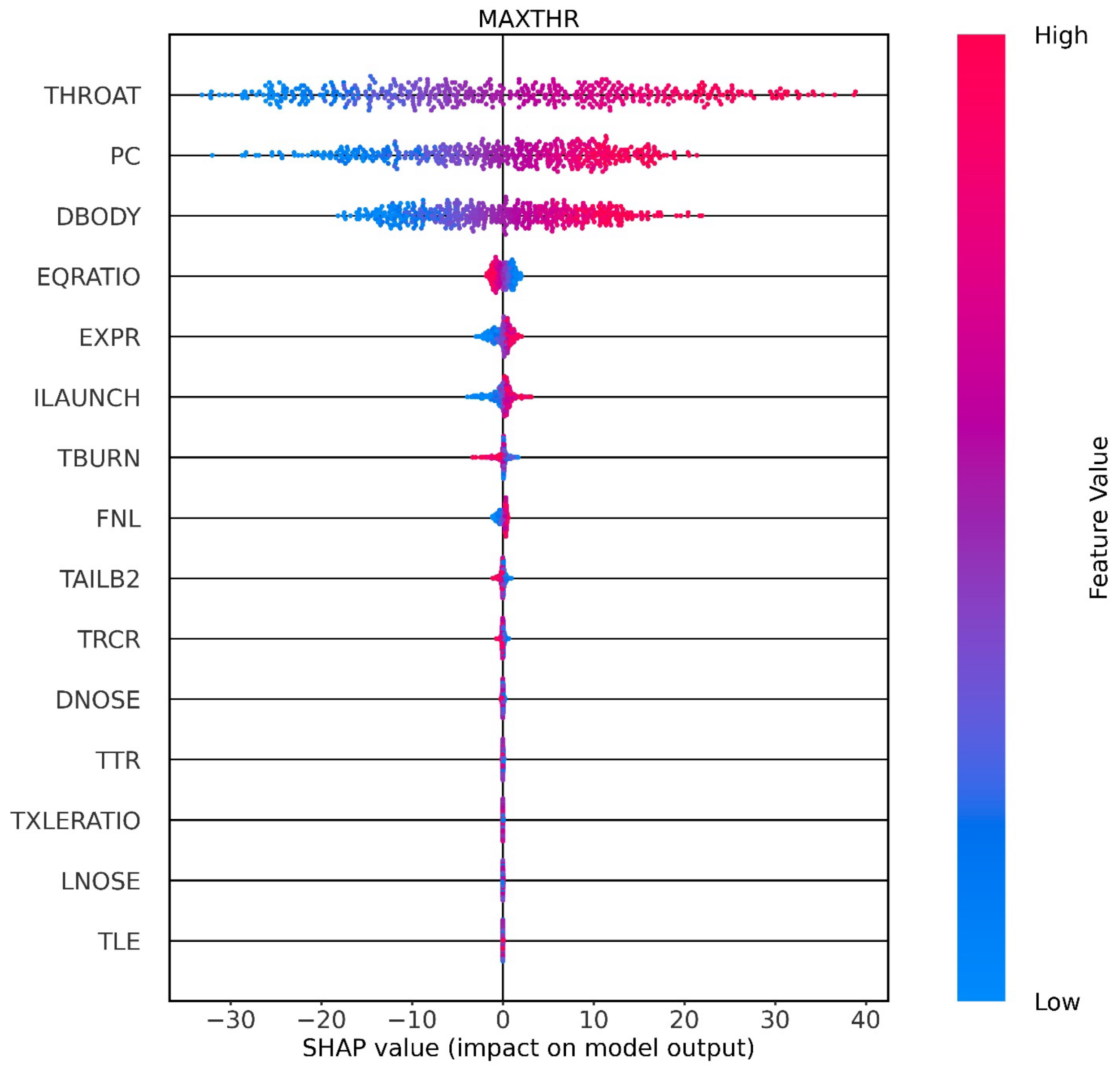


Fig. 74 MAXTHR Sample SHAP Values using Summary Plot.

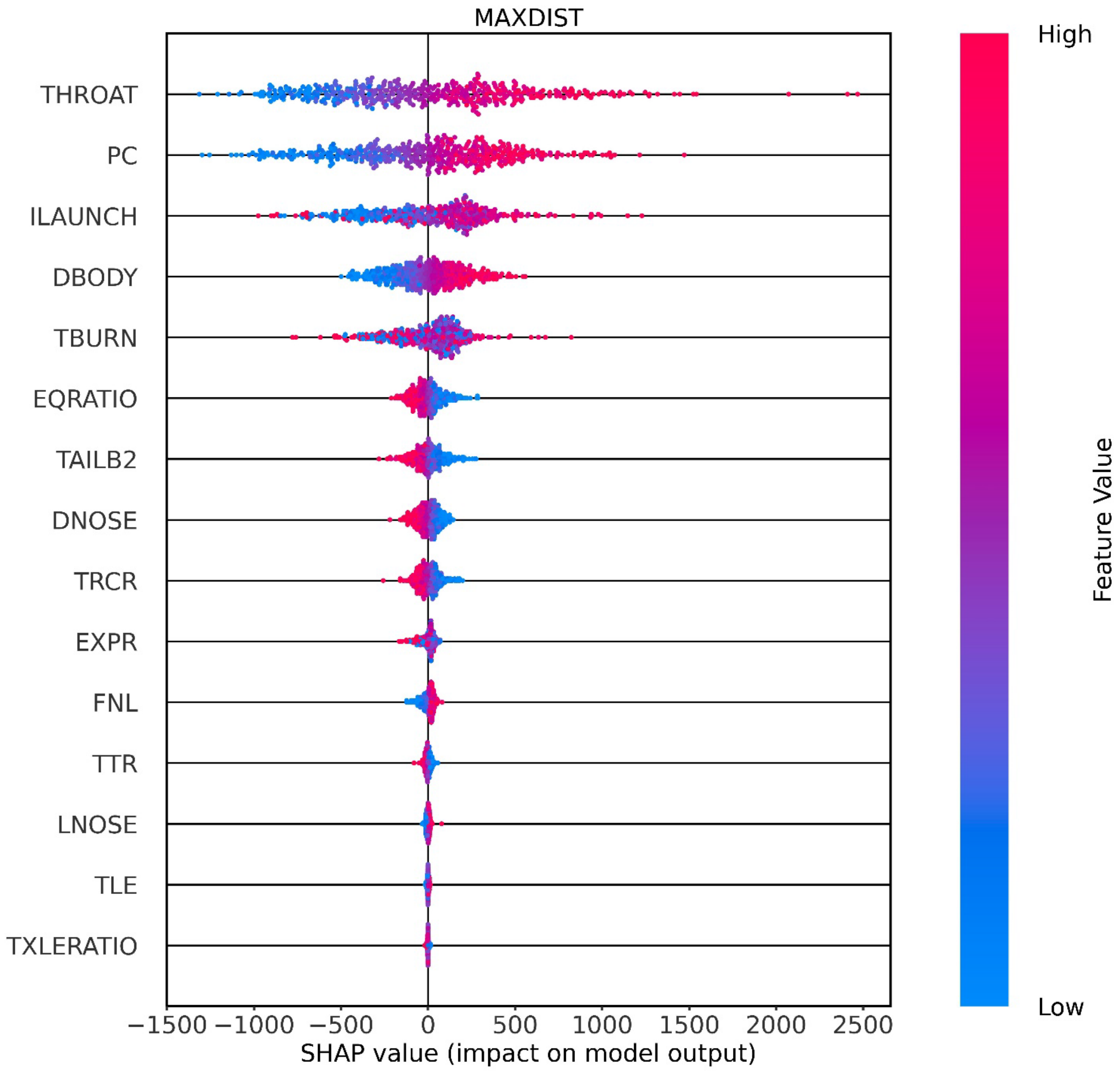


Fig. 75 MAXDIST Sample SHAP Values using Summary Plot.

Figure 76 shows the individual sample SHAP values for the output parameter MAXDIST. Can see that for THROAT, PC, and DBODY that their higher respective values have a positive contribution, and their lower respective values have a negative contribution. EQRATIO, TAILB2, DNOSE, and TRCR show that their lower respective values have a positive contribution, and their higher respective values have a negative contribution. Will notice that ILAUNCH and TBURN were not initially mentioned and that is because we can see that there is a mixture of contributions.

So, for example, on TBURN can see that low and higher values can have both positive and negative contribution to the output.

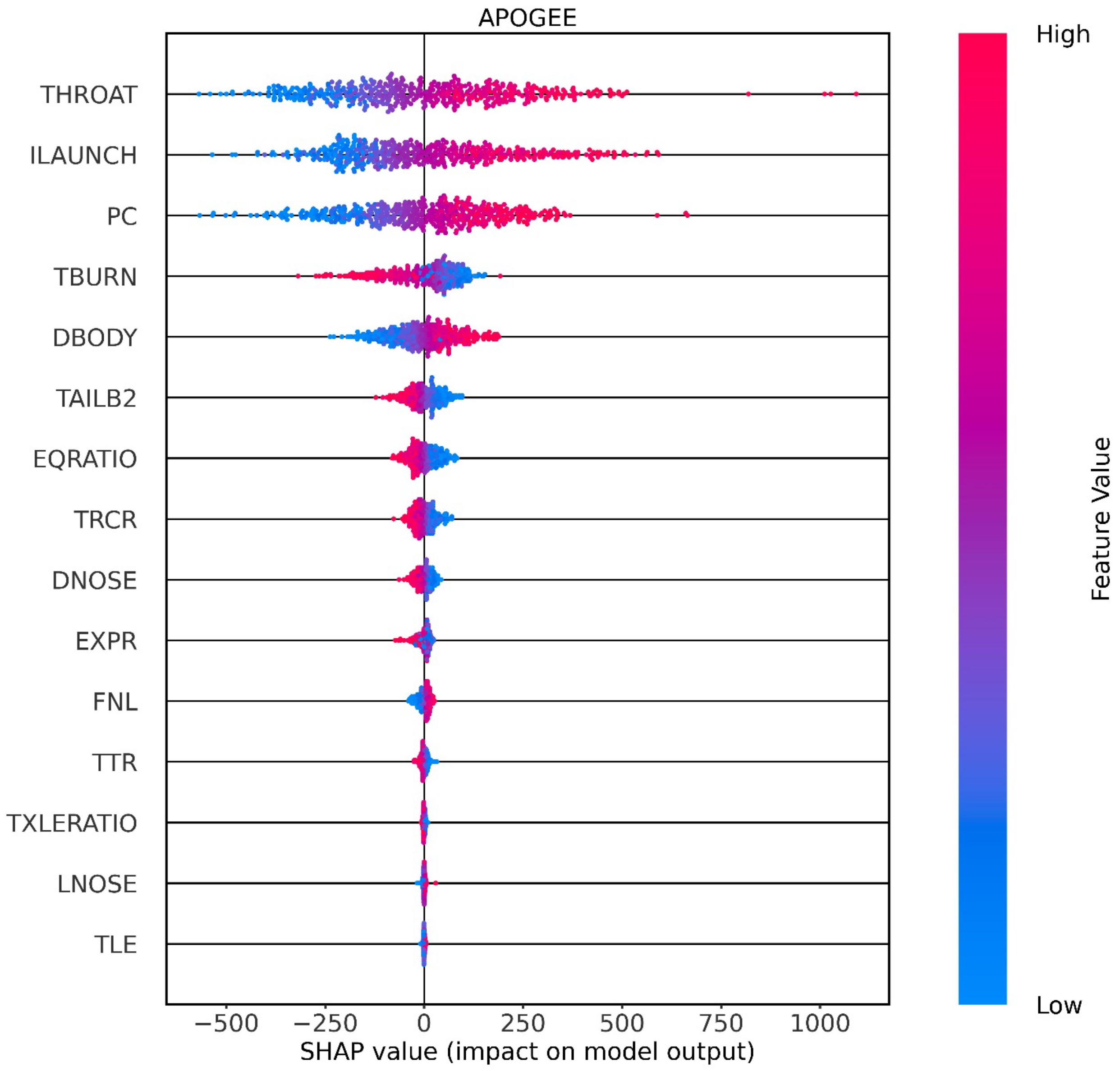


Fig. 76 APOGEE Sample SHAP Values using Summary Plot.

Figure 77 shows the individual sample SHAP values for the output parameter APOGEE. THROAT, ILAUNCH, PC, and DBODY show that their higher respective values have a positive contribution, and their lower respective values have a negative contribution. TBURN, TAILB2, EQRATIO, TRCR, and DNOSE show that their lower respective values have a positive contribution, and their higher respective values have a negative contribution. The parameters below and including EXPR have very little contribution to the output.

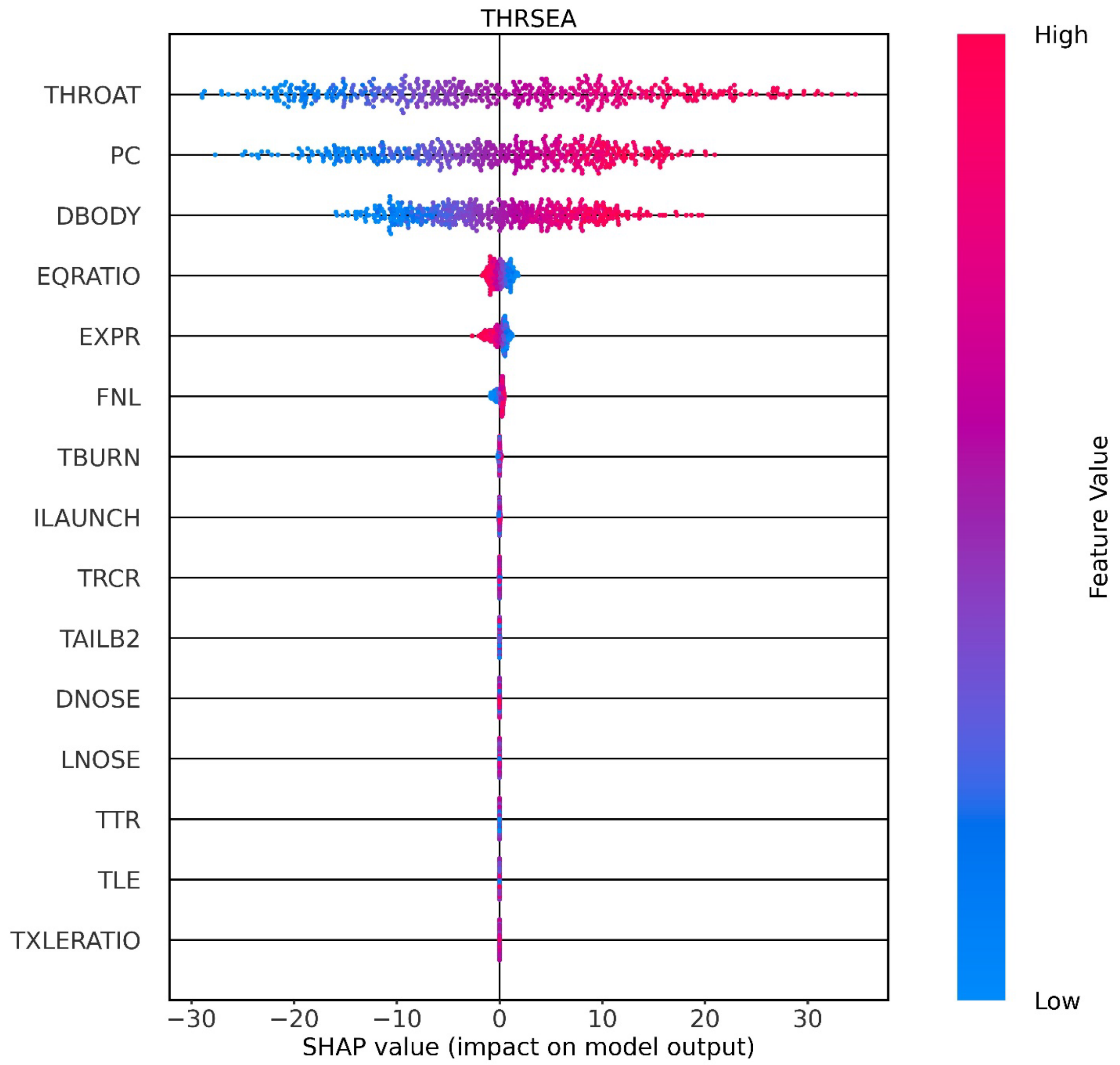


Fig. 77 THRSEA Sample SHAP Values using Summary Plot.

Figure 78 shows the individual sample SHAP values for the output parameter THRSEA. THROAT, PC, and DBODY show that their higher respective values have a positive contribution, and their lower respective values have a negative contribution. EQRATIO and EXPR show that their lower respective values have a positive contribution, and their higher respective values have a negative contribution. The parameters below and including FNL have very little contribution to model output.

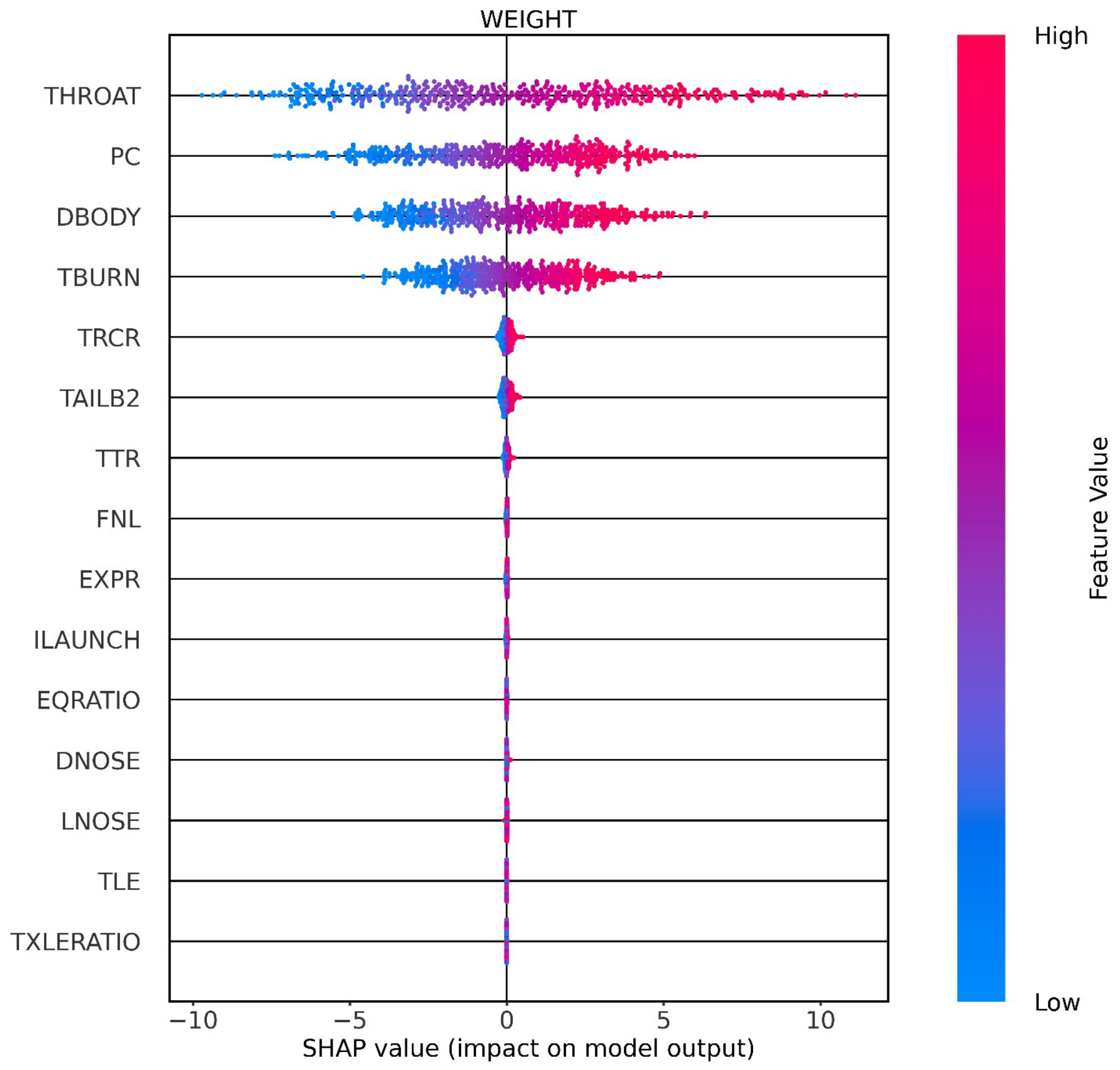


Fig. 78 WEIGHT Sample SHAP Values using Summary Plot.

Figure 79 shows the individual sample SHAP values for the output parameter WEIGHT. THROAT, PC, DBODY, and TBURN show that their higher respective values have a positive contribution, and their lower respective values have a negative contribution. The parameters below and including TRCR have very little contribution to the model output.

With the summary plots, it is important to note that the plots do not tell what the magnitude of the output is, so just because an input feature has positive contribution will not necessarily mean that the output will have a large value. So, we can now examine single samples to see how the local SHAP values adjust the output. The next six plots will show the same sample for each output parameter. The sample is plotted using the SHAP package function called "force_plot". Figure 80 shows the single sample force plotted for TOF. The values which have a positive contribution, i.e., increase the output, are colored red on the left. The values which have a negative contribution, i.e., decrease the output, are colored blue on the right. The input feature which are most important are displayed on the force and their respective input values are displayed as well. The base value shows what the average output value is and can see it is about 280 seconds. Can see that the output for the sample is 382.14 seconds. For this sample, can see that TAILB2, TBURN, PC, and THROAT have positive contributions and most important; ILAUNCH and DBODY have negative contributions and are also most important.

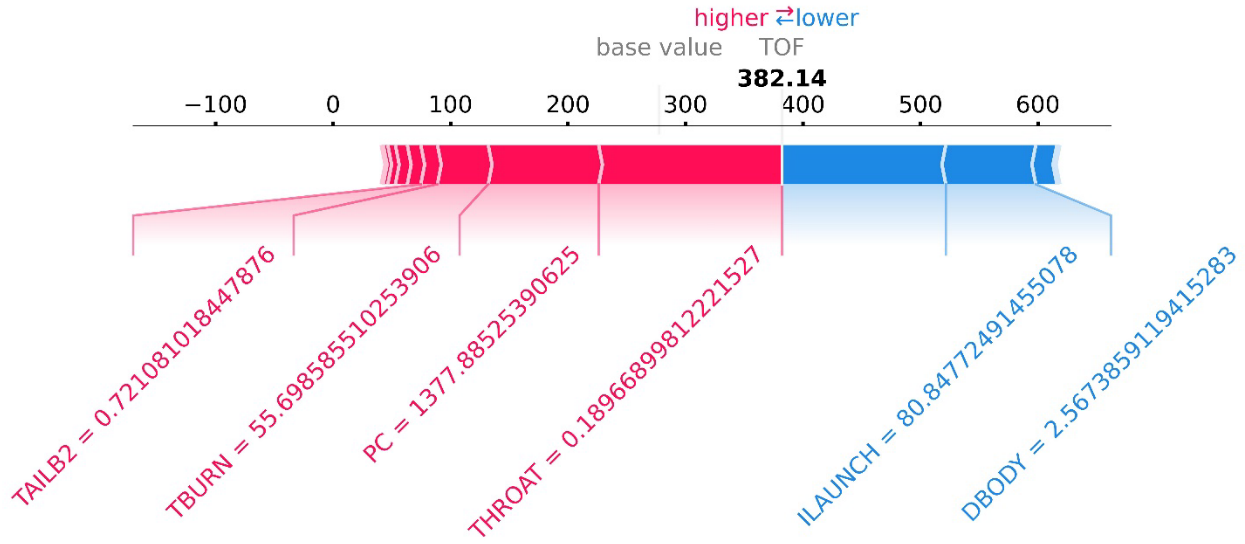


Fig. 79 TOF Single Sample Force Plot.

Next, looking at Fig. 81 shows the single sample force plot for MAXTHR. Can see that PC and THROAT are the most important positive contributors and DBODY is the most important negative contributor. The average (base value) for MAXTHR is approximately 46 Lbf/1000 and the sample output is 59.80 Lbf/1000.

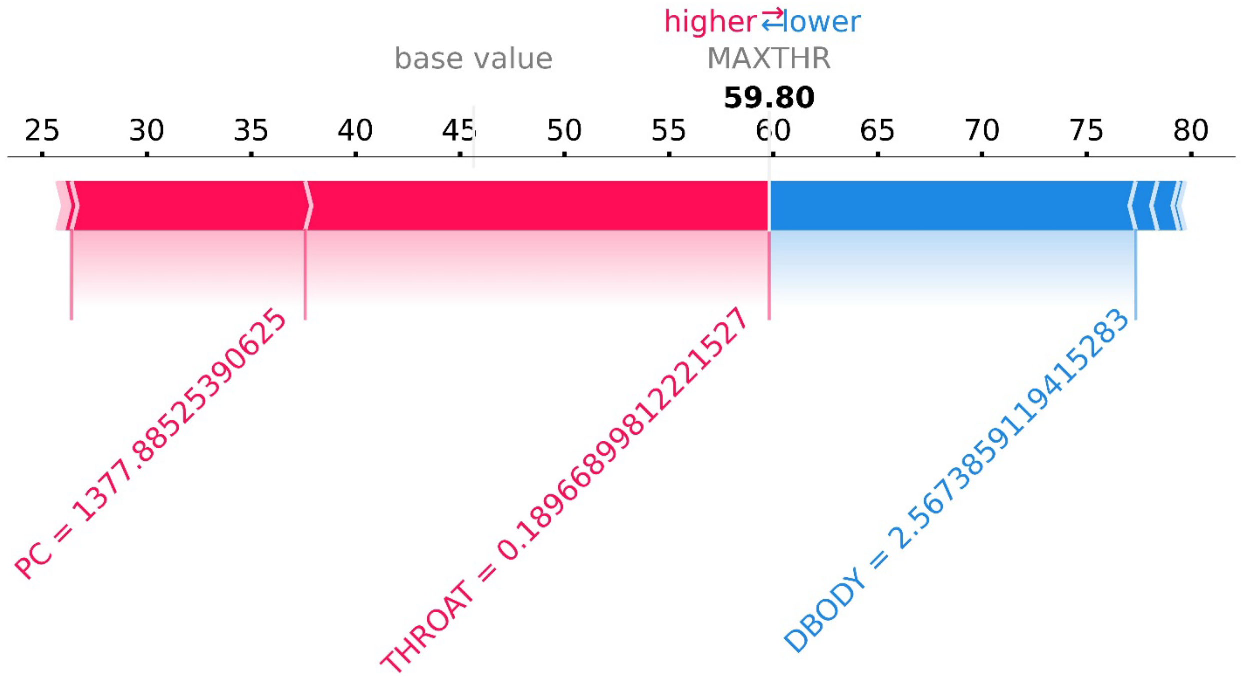


Fig. 80 MAXTHR Single Sample Force Plot.

Next, looking at Fig. 82 shows the single sample force plot for MAXDIST. Can see that DBODY is the most important negative contributor, and TRCR, TAILB2, TBURN, DNOSE, PC, & THROAT are the most important positive contributors. The average value for MAXDIST is approximately 841 ft/1000 and the sample output is 1772.17 ft/1000.

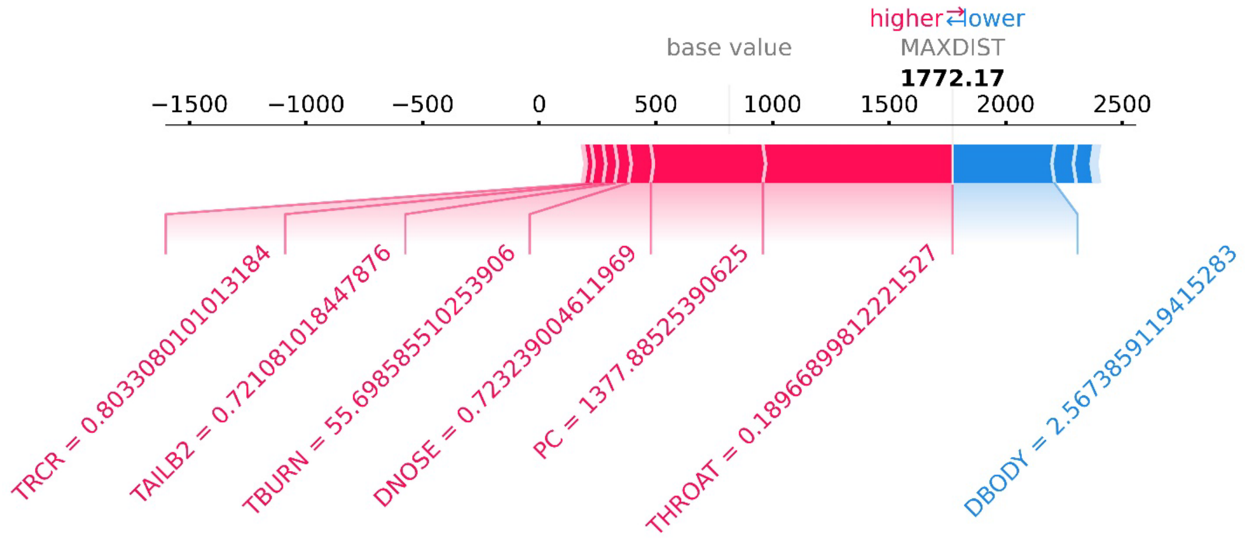


Fig. 81 MAXDIST Single Sample Force Plot.

Next, looking at Fig. 83 shows the single sample force plot for APOGEE. Can see that DBODY & ILAUNCH are the most important negative contributor, and TAILB2, TBURN, PC, & THROAT are the most important positive contributors. The average value for APOGEE is approximately 298.69 ft/1000 and the sample output is 458.71 ft/1000.

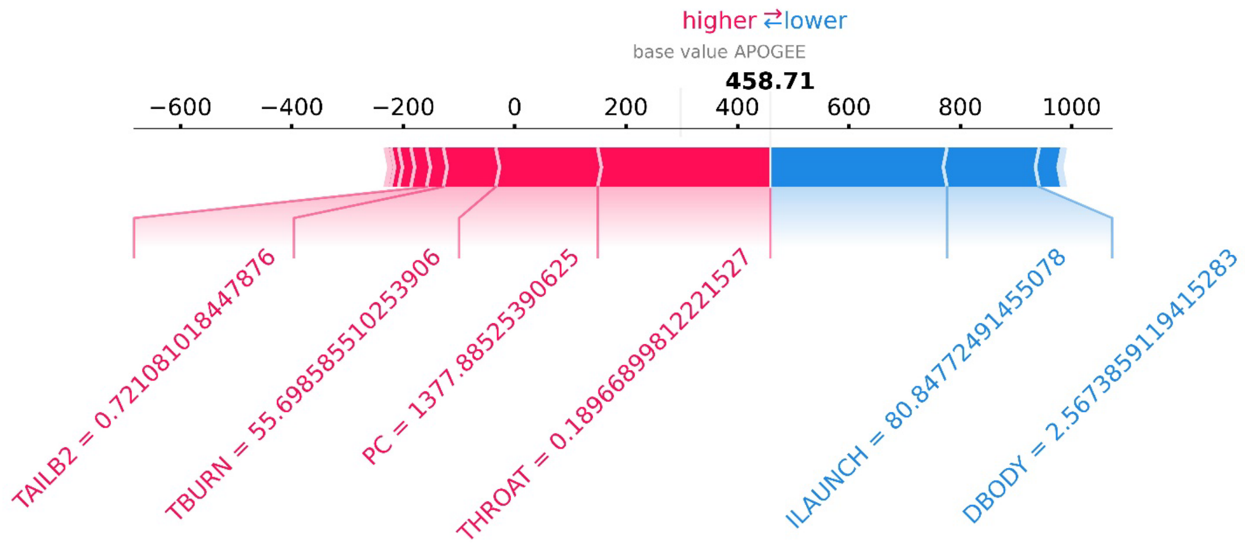


Fig. 82 APOGEE Single Sample Force Plot.

Next, looking at Fig. 84 shows the single sample force plot for THRSEA. Can see that DBODY is the most important negative contributor, and PC & THROAT are the most important positive contributors. The average value for THRSEA is approximately 40.31 Lbf/1000 and the sample output is 55.48 Lbf/1000.

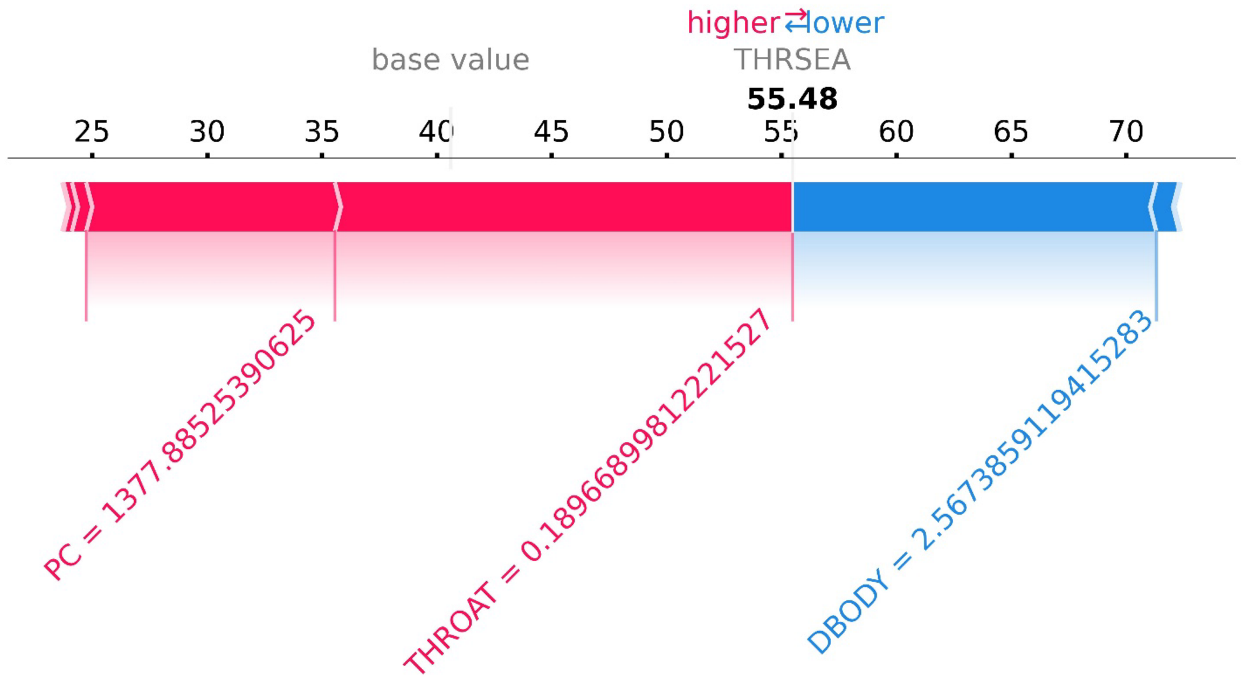


Fig. 83 THRSEA Single Sample Force Plot.

Next, looking at Fig. 85 shows the single sample force plot for WEIGHT. Can see that DBODY is the most important negative contributor, and PC & THROAT are the most important positive contributors. The average value for WEIGHT is approximately 16.56 Lbm/1000 and the sample output is 19.30 Lbm/1000.

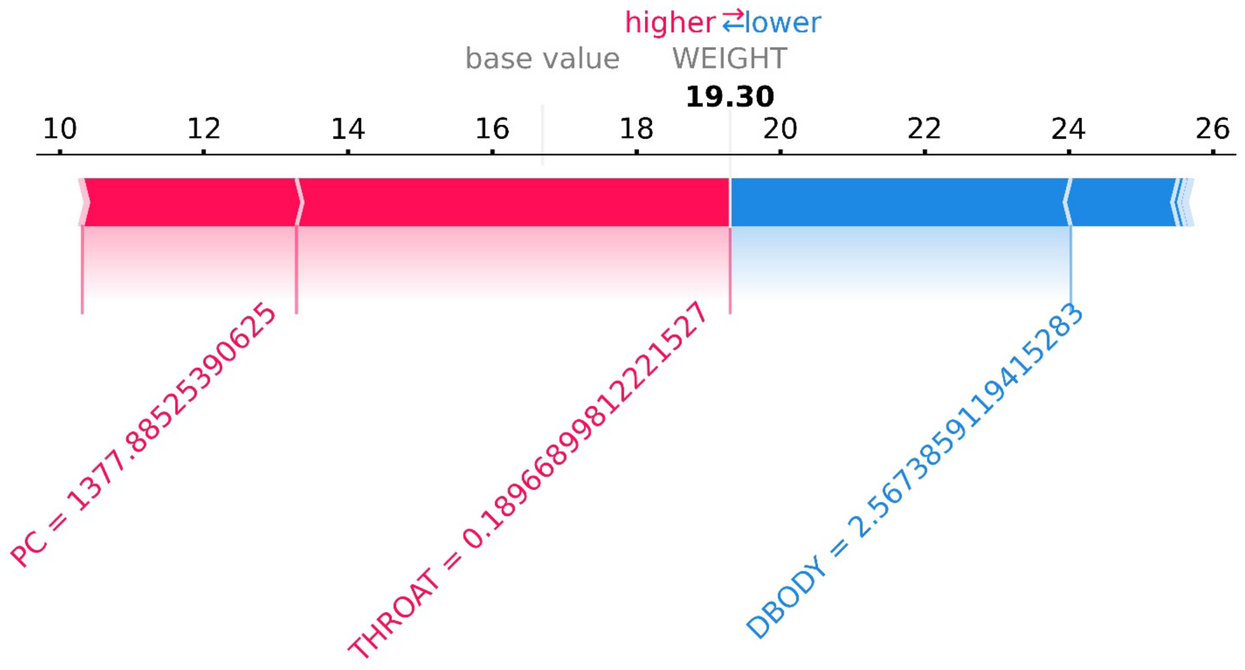


Fig. 84 WEIGHT Single Sample Force Plot.

VII.VI Classification Explanations

To explain the classification model the SHAP package can be used for LDA (we will exclude QDA) and the classification NNET. For this data, remember that a class value is being predicted. So, take caution when interpreting the SHAP values for classification.

VII.VI.I LDA & Shapley Value Global Average

First, LDA is evaluated and is shown to be very useful as an interpretable model. So, LDA can be used to directly interpret how the model is making predictions by evaluating the dimensionally reduced components. Fortunately, when using LDA in Scikit-Learn, the number of dimensionally reduced terms ("n_components") can be set. Here, this parameter is set two so that the parameters can be easily visualized on a two-dimensional plot. Figure 86 shows the

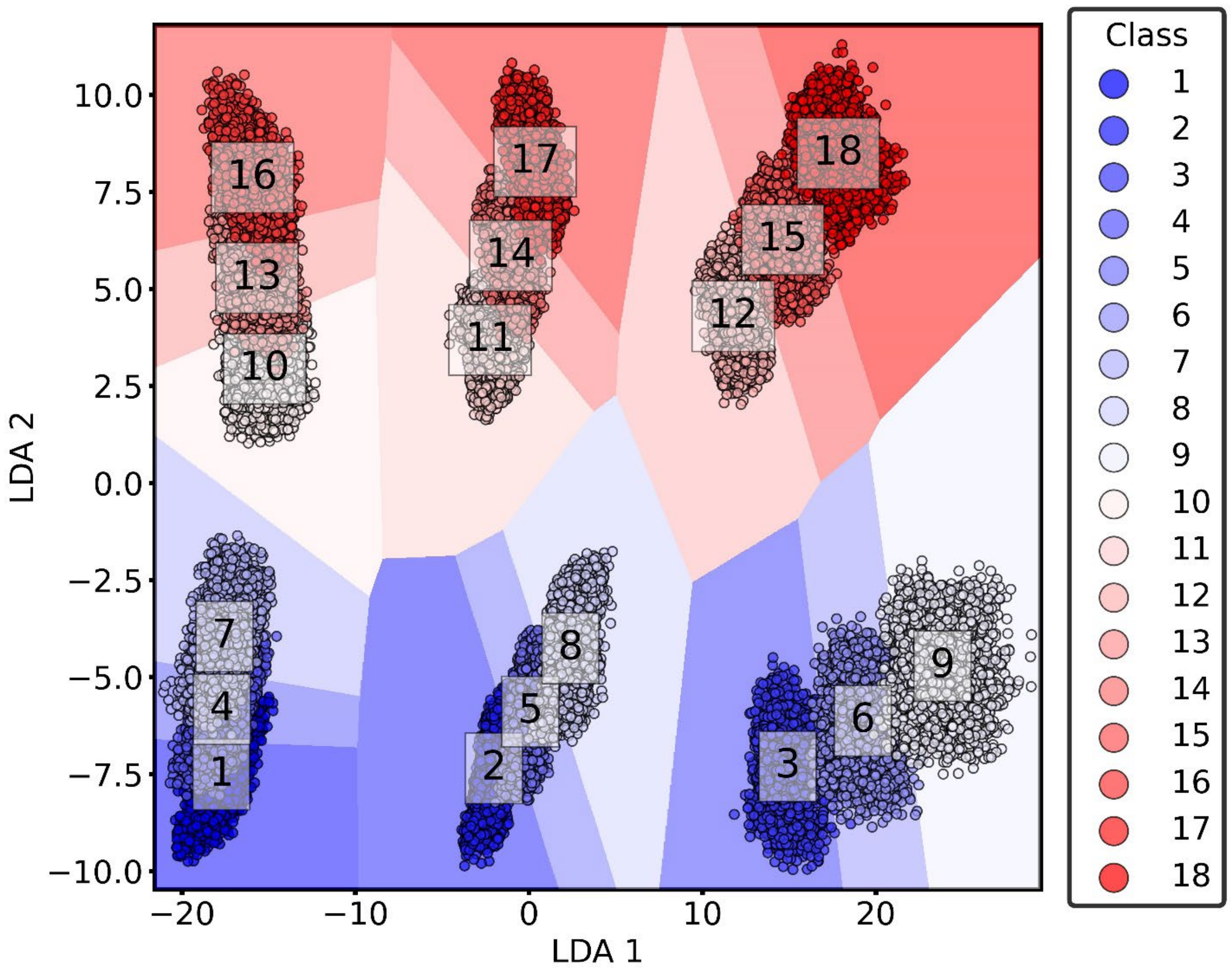
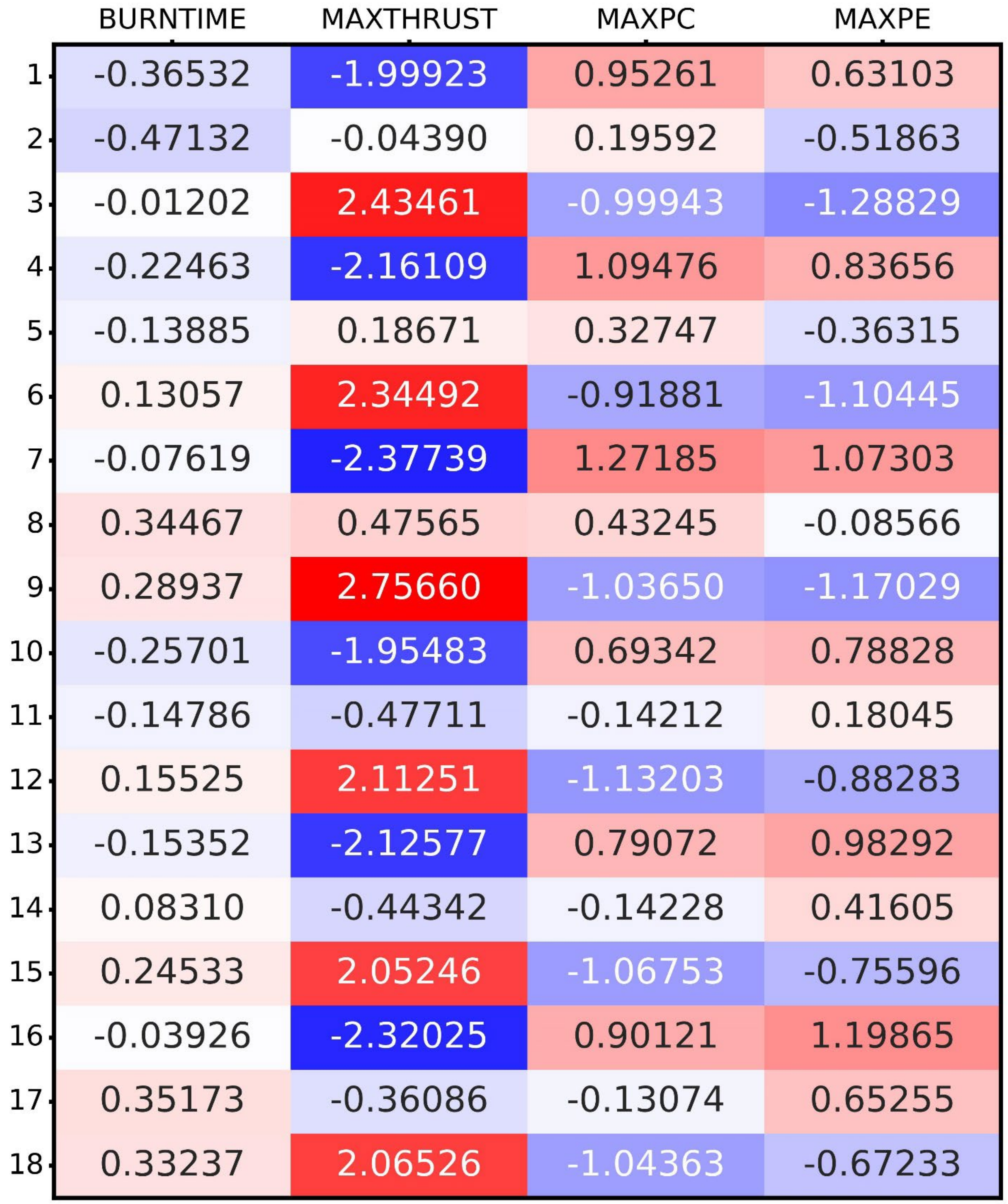


Fig. 85 LDA 1 vs. LDA 2 Dimensionally Reduced Components.

dimensionally reduced parameters LDA 1 versus LDA 2 for the entire set of 18 classes, which each class number of plotted on its corresponding class group. Can see that the classifier is able to separate the classes. On LDA 2, can see that the classes a split vertically and can see that the classifier was able to split the classes based on THROAT, which the first 9 classes have THROAT equal to 0.14264 and classes 10-18 have THROAT equal to 0.18264. On LDA 1 can see that the classifier is able split the classes based on DBODY. Classes 1, 4, 7, 10, 13, & 16 have the same DBODY equal to 0.844. Classes 2, 5, 8, 11, 14, & 17 have the same DBODY equal to 0.884. Classes 3, 6, 9, 12, 15, & 18 have the same DBODY equal to 0.924. Can see that classes 1, 4, & 7 have the same DBODY, but are not separated as much and is an effect of the reduction method.

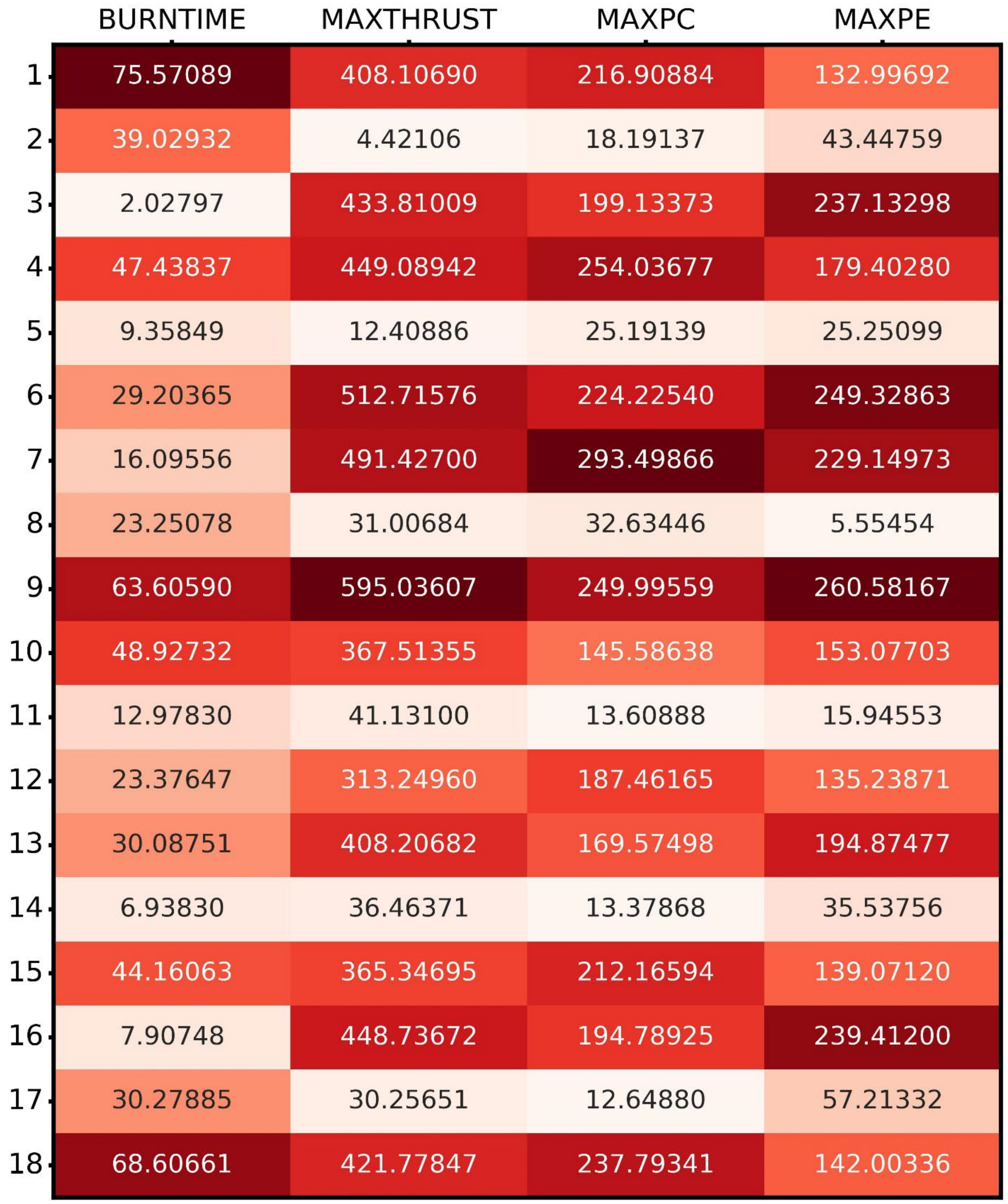
These classes have different FINENESS and only separated by small subregions. So, can see that the LDA reduction method is having a harder time of separating the classes due to FINENESS. This reduction method would have been more appropriate if it would have been reduced to three components and shown on a three dimensional plot.

Next, the standardized coefficients of the LDA model are plotted on a heatmap to show which parameters are most influential. These standardized coefficients are transposed on Fig. 87, so that they can fit on the document. Standardized coefficients for LDA may not make sense, but utilizing Eq. (5.5) we can write the LDA equation in a form similar to linear regression and make use of the "standardized coefficients". The coefficients must be carefully translated compared to the regression. For the output Class 1 (top row), BURNTIME & MAXTHRUST have a negative weight or contribution, and MAXPC & MAXPE have a positive contribution. Can see that MAXTHRUST has the largest contribution. Overall BURNTIME seems to have a very small contribution. Interestingly enough, MAXTHRUST appears to have cyclical importance across the output features (along the MAXTHRUST column). Notice that its starts largely negative on class 1, gets very small in magnitude on class 2, then gets largely positive on class 3. Can see that this repeats for every 3 classes. On MAXPC column, notice that class 1 starts out largely positive, class 2 starts positively smaller, and class 3 is largely negative. The cycle repeats for the first 9 classes and when class reaches 11 (different value of THROAT but corresponds to same DBODY as class 2) the small positive value changes to small negative. This new cycle repeats for class 10-18. Finally, on MAXPE, notice that class 1 starts positive, class 2 starts negative, and class 3 starts out larger negatively, and this cycle repeats for the first 9 classes as well. Similarly, to MAXPC, when reaching class 11, the would be negative value changes to positive contribution and the new cycle repeats for classes 10-18. Overall, it does seem that MAXTHRUST is the most contributing and BURNTIME seems to be the least contributing.



	BURNTIME	MAXTHRUST	MAXPC	MAXPE
1	-0.36532	-1.99923	0.95261	0.63103
2	-0.47132	-0.04390	0.19592	-0.51863
3	-0.01202	2.43461	-0.99943	-1.28829
4	-0.22463	-2.16109	1.09476	0.83656
5	-0.13885	0.18671	0.32747	-0.36315
6	0.13057	2.34492	-0.91881	-1.10445
7	-0.07619	-2.37739	1.27185	1.07303
8	0.34467	0.47565	0.43245	-0.08566
9	0.28937	2.75660	-1.03650	-1.17029
10	-0.25701	-1.95483	0.69342	0.78828
11	-0.14786	-0.47711	-0.14212	0.18045
12	0.15525	2.11251	-1.13203	-0.88283
13	-0.15352	-2.12577	0.79072	0.98292
14	0.08310	-0.44342	-0.14228	0.41605
15	0.24533	2.05246	-1.06753	-0.75596
16	-0.03926	-2.32025	0.90121	1.19865
17	0.35173	-0.36086	-0.13074	0.65255
18	0.33237	2.06526	-1.04363	-0.67233

Fig. 86 LDA Standardized Coefficients.



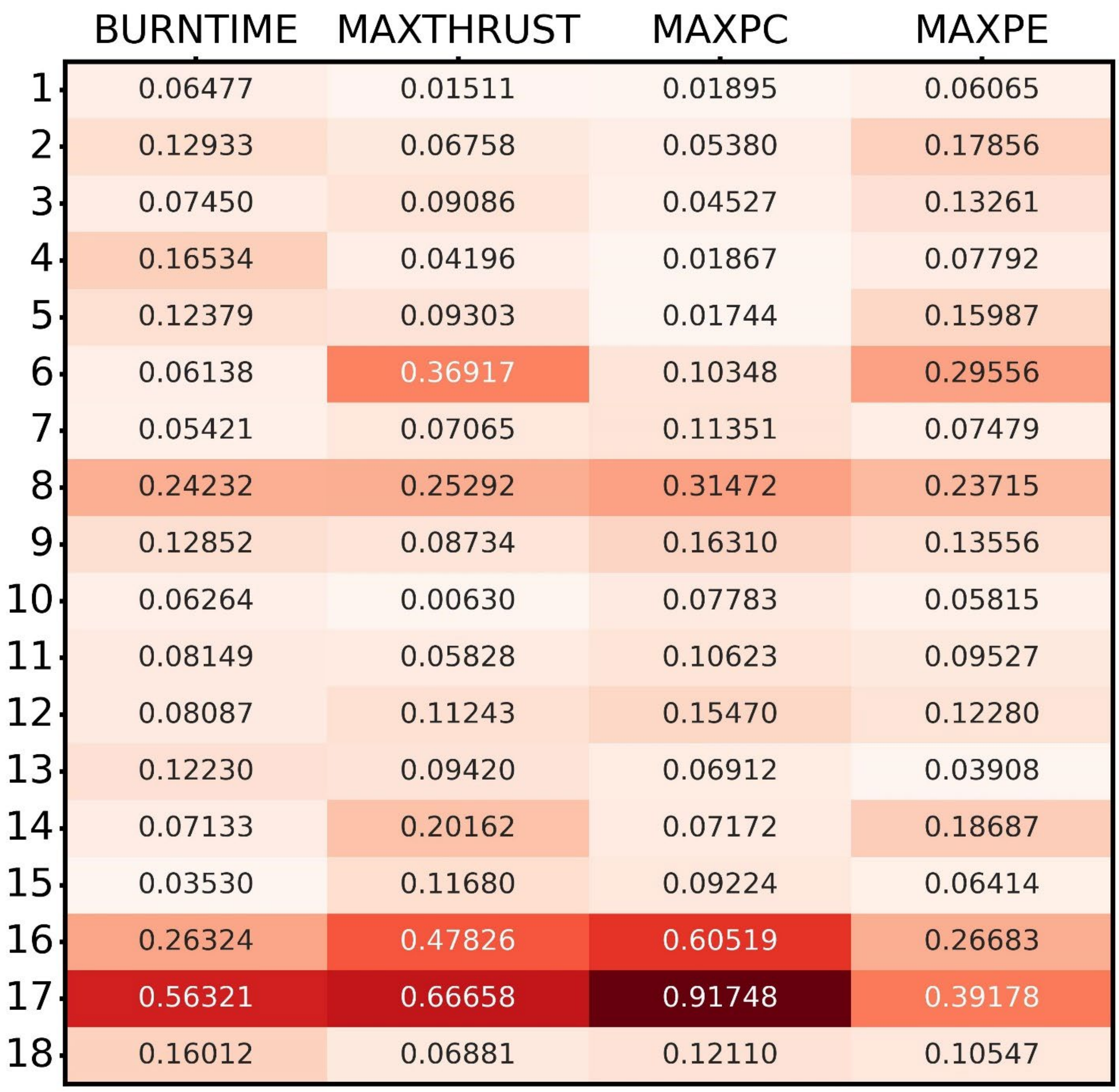
	BURNTIME	MAXTHRUST	MAXPC	MAXPE
1	75.57089	408.10690	216.90884	132.99692
2	39.02932	4.42106	18.19137	43.44759
3	2.02797	433.81009	199.13373	237.13298
4	47.43837	449.08942	254.03677	179.40280
5	9.35849	12.40886	25.19139	25.25099
6	29.20365	512.71576	224.22540	249.32863
7	16.09556	491.42700	293.49866	229.14973
8	23.25078	31.00684	32.63446	5.55454
9	63.60590	595.03607	249.99559	260.58167
10	48.92732	367.51355	145.58638	153.07703
11	12.97830	41.13100	13.60888	15.94553
12	23.37647	313.24960	187.46165	135.23871
13	30.08751	408.20682	169.57498	194.87477
14	6.93830	36.46371	13.37868	35.53756
15	44.16063	365.34695	212.16594	139.07120
16	7.90748	448.73672	194.78925	239.41200
17	30.27885	30.25651	12.64880	57.21332
18	68.60661	421.77847	237.79341	142.00336

Fig. 87 LDA Mean(|SHAP|) Values.

Next, the mean absolute SHAP values are plotted on Fig. 88 and notice that there are no negative values like on Fig. 87. We will notice very similar trends as those on Fig. 87. BURNTIME does not seem follow a trend. For MAXTHRUST, class 1 shows a large contribution, class 2 shows a small contribution, and class 3 shows a large contribution. This cycle repeats for every 3 classes for all the classes. For MAXPC, class 1 shows a large contribution, class 2 shows a small contribution, and class 3 shows a large contribution and this cycle repeats for every 3 classes for all classes. For MAXPE, class 1 shows a large contribution, class 2 shows a small contribution, and class 3 shows a large contribution and this cycle repeats for every 3 classes for all classes. When looking at the rows, can see that MAXTHRUST seems to be the most contributing. MAXPC and MAXPE seems to be second most contributing, and they seem to alternate for the classes. BURNTIME is the least contributing.

VII.VI.II Neural Network Classification & Shapley Value Explanations

Next, we can apply the SHAP package again using the "DeepExplainer" function to obtain the SHAP values for NNET using TensorFlow. Figure 89 shows the mean absolute global contributions (SHAP values) for the NNET that was trained for 5,000 epochs in Sect. V.III.III. Because the model uses the categorical cross entropy as the loss function, the SHAP values are scaled much smaller. So, it is much harder to observe any characteristics and relate how the SHAP values are predicting the output. Instead, it may be useful to utilize other SHAP plots to generalize about the model predictions. The complexity in the classification problem, is that the NNET has 18 output units so there are 18 multidimensional arrays of SHAP values that can be used for visualization. Interpreting the NNET classification models are not easy as the NNET regression model.



	BURNTIME	MAXTHRUST	MAXPC	MAXPE
1	0.06477	0.01511	0.01895	0.06065
2	0.12933	0.06758	0.05380	0.17856
3	0.07450	0.09086	0.04527	0.13261
4	0.16534	0.04196	0.01867	0.07792
5	0.12379	0.09303	0.01744	0.15987
6	0.06138	0.36917	0.10348	0.29556
7	0.05421	0.07065	0.11351	0.07479
8	0.24232	0.25292	0.31472	0.23715
9	0.12852	0.08734	0.16310	0.13556
10	0.06264	0.00630	0.07783	0.05815
11	0.08149	0.05828	0.10623	0.09527
12	0.08087	0.11243	0.15470	0.12280
13	0.12230	0.09420	0.06912	0.03908
14	0.07133	0.20162	0.07172	0.18687
15	0.03530	0.11680	0.09224	0.06414
16	0.26324	0.47826	0.60519	0.26683
17	0.56321	0.66658	0.91748	0.39178
18	0.16012	0.06881	0.12110	0.10547

Fig. 88 NNET Classification Mean(|SHAP|).

The mean absolute SHAP are then plotted on a stacked bar chart on Fig. 90. Can see that the overall contributor to the model is MAXPC, followed by MAXTHRUST, MAXPE, and finally BURNTIME. It is obvious to see that looking at the global values on Fig. 89 & Fig. 90 is not very useful and there is not much information to be gained. So, instead it will be more useful to look at specific samples case.

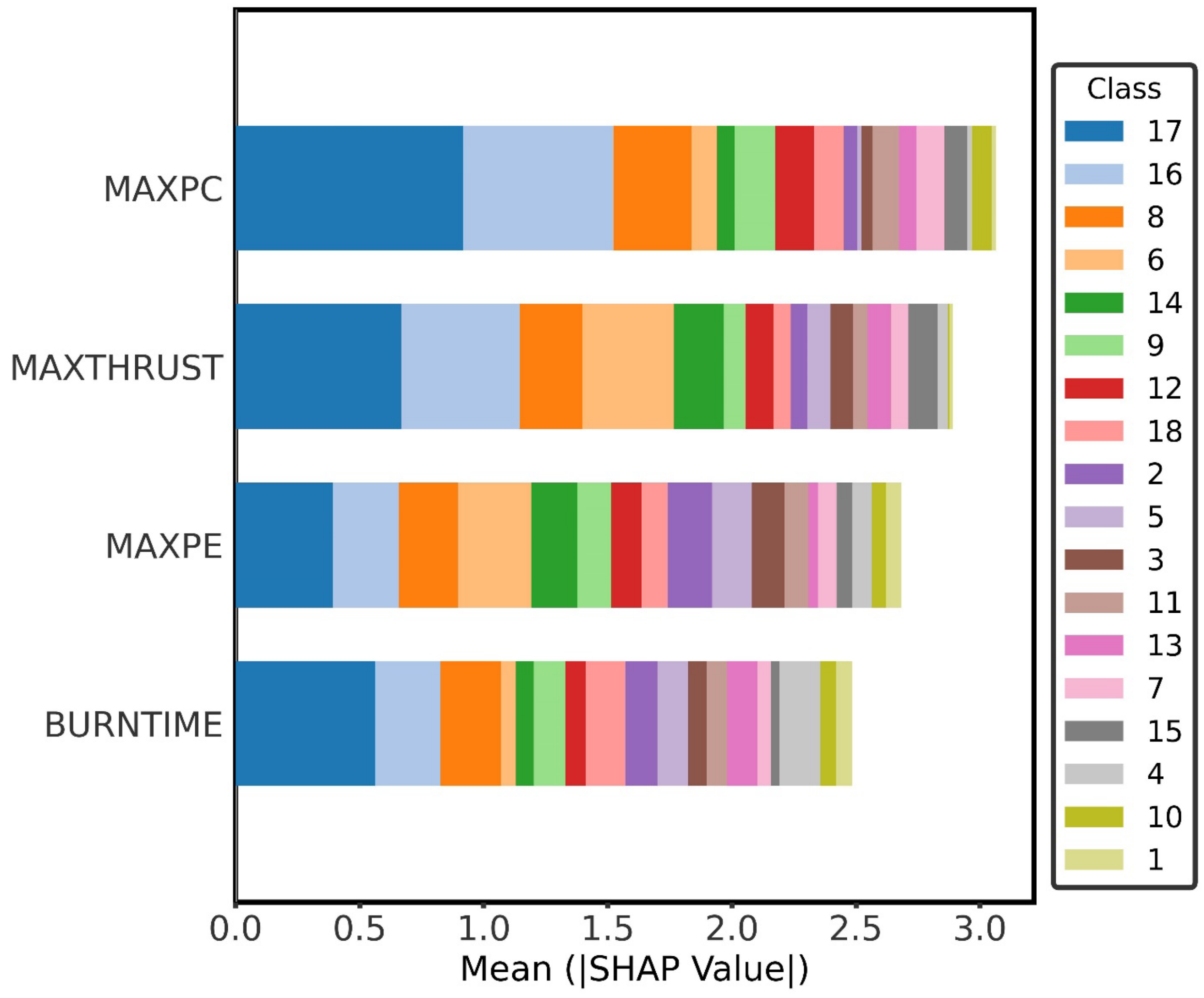


Fig. 89 NNET Classification Global Mean (|SHAP|) Values Stacked Bar Chart.

The force plots can be utilized to see how single samples are being predicted. For example, a single sample is plotted on Fig. 91 and can see how the NNET makes a prediction. Because the output value was plotted for Class 1, can see how the NNET predicts the output value. Notice that the sample predicts an output of 0.00, so the sample does not belong to Class 1. Can see that

BURNTIME & MAXTHRUST were positively contributing and MAXPE & MAXPC were negatively contributing.

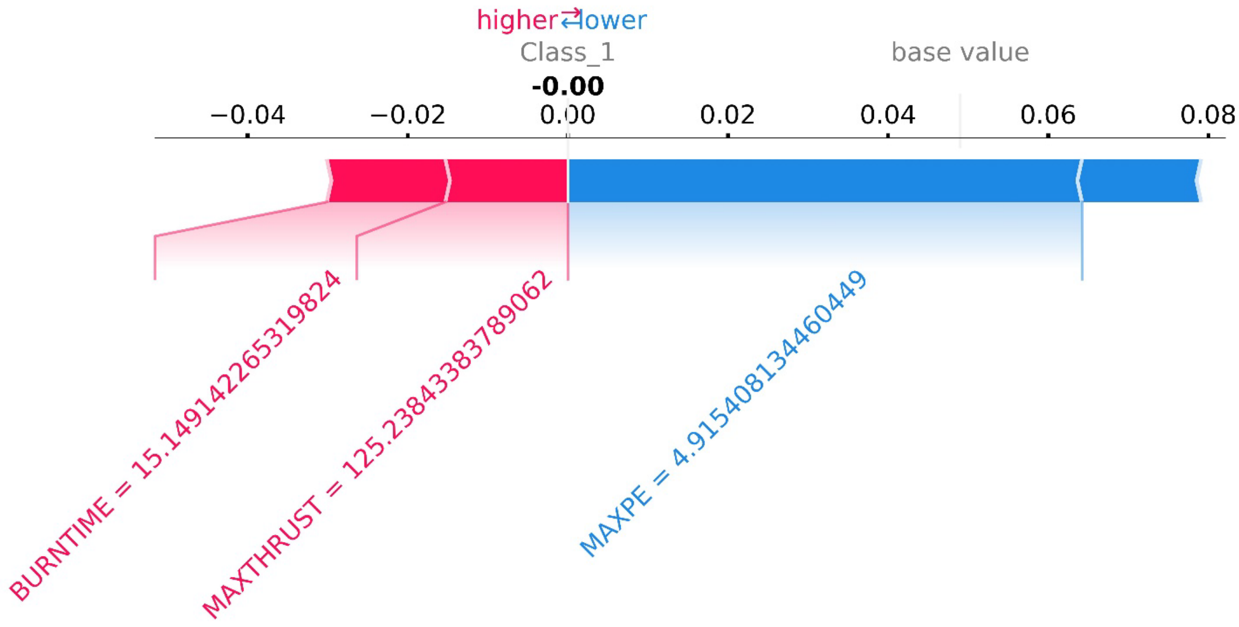


Fig. 90 Single Sample Class 1 Force Plot.

Next, the same sample SHAP values are plotted for Class 5 predictions on Fig. 92. Can see that this sample predicts a value of 1.00, which means that this sample belongs to Class 5. Can see that MAXTHRUST, MAXPE, and BURNTIME were positively contributing and MAXPC was negatively contributing to the output. Because the single sample predicts a value of 1.00 for Class 5, this sample will obviously predict zero for the other SHAP force plots. When calculating the SHAP values, a multidimensional list of [number of classes]=[18] two-dimensional arrays are created. These arrays refer to SHAP values per sample and are of size [number of samples, number of inputs]=[1000, 4]. Therefore, the multidimensional list is of size [18][1000, 4], therefore each list refers to the output class value. For example [0][0, :] refers to the output for class 1 using the sample 1 inputs. So, only one of the lists for a single sample will predict a value of 1.00.

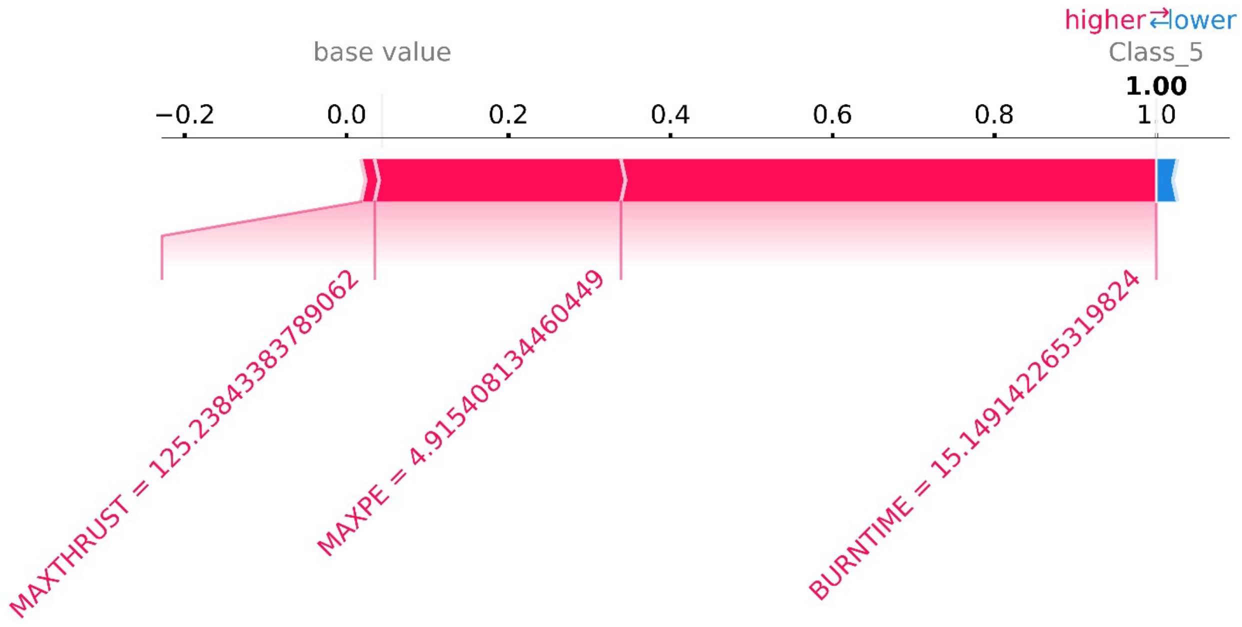


Fig. 91 Single Sample Class 5 Force Plot.

So, if the rest of the class output lists are force plotted for the single sample, they will all just predict a value of zero and will be repetitive, therefore most of the class predictions are not shown. Instead for example purposes, only classes 10 and 15 will be shown. Figure 93 shows the single sample prediction and obviously it is zero. Can see that BURNTIME has a positive contribution to the output and MAXPC & MAXPE have a negative contribution to the output. Can see that MAXTHRUST has no visible contribution. Next on Fig. 94, can see the single sample predict zero. Can see that MAXTHRUST has a positive contribution, and MAXPC, BURNTIME, & MAXPE have a negative contribution.

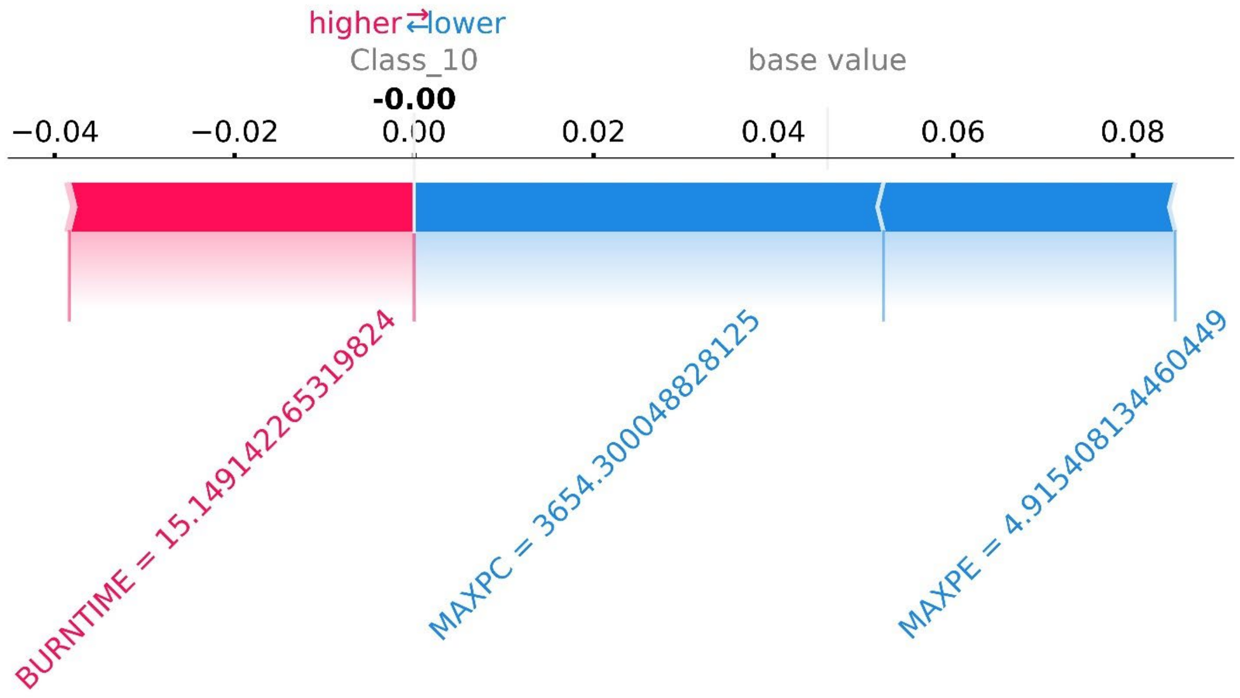


Fig. 92 Single Sample Class 10 Force Plot.

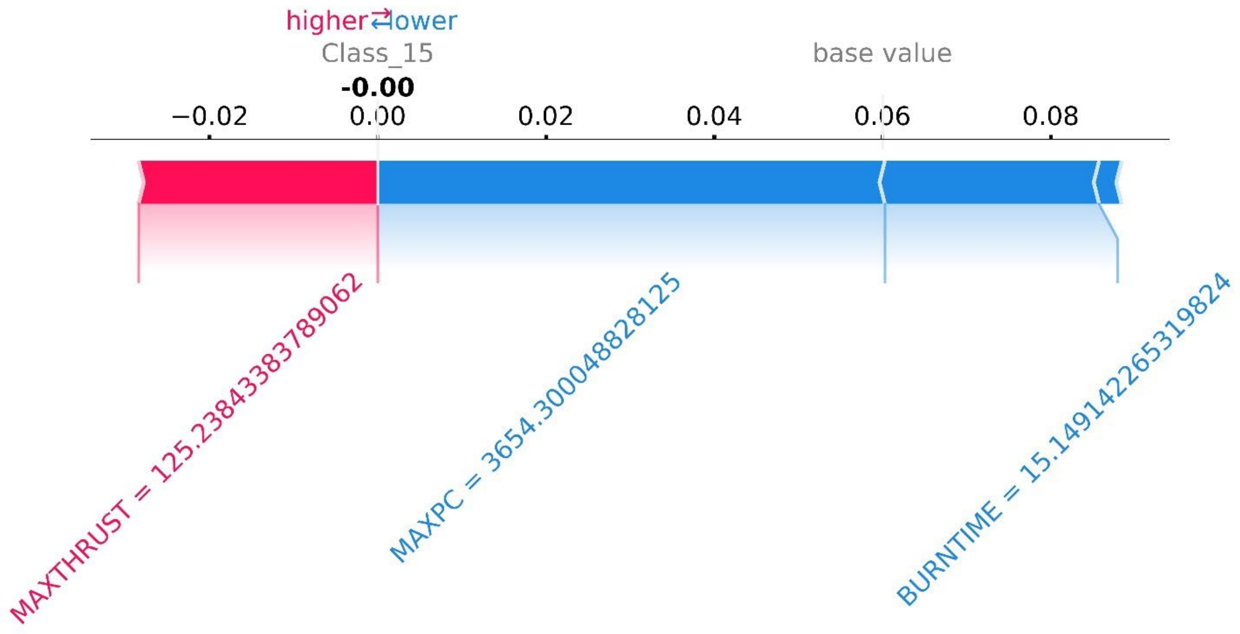


Fig. 93 Single Sample Class 15 Force Plot.

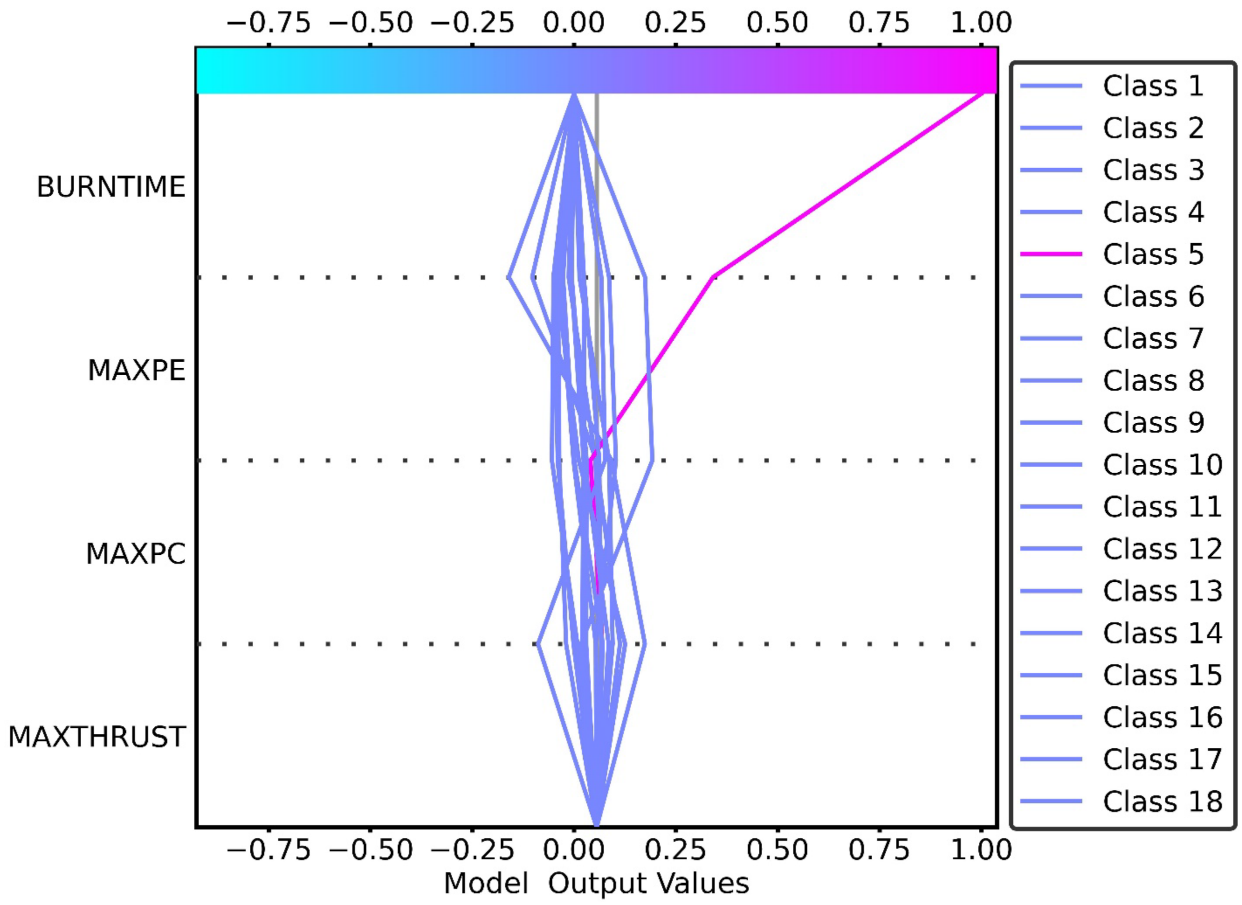


Fig. 94 Single Multi-Output Decision Plot.

Although the force plots are extremely useful for visualizing individual case by case outputs, it can be visually useful to see how each prediction arrives at the final solution. The "multioutput_decision_plot" from SHAP can be used to plot a single sample and see how the NNET makes the predictions. Because this is a classification NNET, the outputs will either arrive at zero or one. Figure 95 shows the NNET take the input sample and predicts which class the sample belongs to and can see from the legend that the sample belongs to class 5, which is the same sample used for figures 91-94. The input features are sorted based on importance and can see that for this specific sample that BURNTIME is the most contributing parameter and MAXTHRUST is the least contributing parameter. So, it is important to understand that the global SHAP values need to be used with caution when generalizing the input features.

VII.VII Other Methods for Explanation

There are other methods for explaining but not all of them are available for TF NNETs and not all of them are generalizable to all models. Using the SHAP package allows for a generalization of all model types built in Python. Refer to Molnar [140] for an in-depth discussion and implementations of model interpretation functions such as global model agnostic methods, which include partial dependence plots and accumulated local effect plots, and local model agnostic methods, which include individual conditional expectation and local surrogate (LIME). Although these plots can be very useful, SHAP seems to be the most useful because of its ability to do global and local explanations. The main issues with SHAP arise mostly from interpretation and one must be careful in how the results are interpreted. However, these issues are also possible with other methods so in general caution must be taken when interpreting the explanation results.

VIII. Conclusions & Future Work

This dissertation described the process for generating simulated experimental data so that statistical analysis and machine learning could be applied. Two datasets were generated and the first was developed for regression purposes and the other dataset was developed for classification. The regression dataset was developed from the Auburn University Liquid Rocket Code and the classification dataset was developed from the Auburn University Solid Rocket Code. For regression, the models were generated so that output features such as TOF and MAXDIST could be predicted from geometric parameters such as DBODY and launch parameters such as ILAUNCH. For classification, the missiles are differentiated by generated 3 different DBODY values, 3 different FINENESS values, and 2 different THROAT values.

For regression, traditional linear regression methods (linear, ridge, and lasso) were developed and shown to be only accurate predicting MAXTHR, THRSEA, and WEIGHT. Therefore, a NNET was implemented in TensorFlow and from the multi-layer multi-unit analysis a network with 3 hidden layers and 90 units per layer was chosen and trained for 50,000 epochs. The overall testing MSE was 144.2935 and predicted versus actual plots show minimal error even for MAXDIST. With the regression database, it was shown that predicting TOF, APOGEE, and MAXDIST were the hardest parameters to model even using a NNET.

For classification, the inputs DBODY, FINENESS, and THROAT are not the inputs for classification, instead the output parameters MAXTHRUST, BURNTIME, MAXPE, and MAXPC are used as inputs and the outputs are the class values 1-18. To classify these missiles, the classical linear and quadratic discriminant analysis functions were used to achieve a 99.3778% and

99.9111% testing accuracy, respectively. In many datasets, LDA and QDA typically would not be able to differentiate between classes, but for this specific database they both work perfectly. A classification NNET was developed using a single layer with 50 hidden units and only needed 5000 epochs to achieve a 99.97% testing accuracy.

Data is not always available and so sample inputs were randomly selected and assumed to be missing. This missing data needs to be replaced and so imputation models were developed to try to replace the missing parameters using linear regression and NNETs. Linear regression was shown not to be able to impute the data and so NNETs were implemented to try and improve results. Results showed that for single missing parameter, the testing accuracy could reduce to 80% and when missing two parameters, the accuracy could drop to 30% testing accuracy but testing accuracy could achieve 87% testing accuracy. When missing three of the four parameters, the testing accuracy drops below 25% testing accuracy. Some of the main reasons the accuracy was reduced so much was that not even NNETs could not accurately reproduce the data when missing three parameters.

Methods for model explainability were produced for both the regression and classification models. If using linear models which can be modeled as linear regression function such as ridge regression and LDA, the standardized coefficients can be directly used for model interpretation. However, the standardized coefficients only allow for global model interpretation. Global model interpretation cannot say how influential individual samples are for prediction. So, the Shapley Additive Explanation (SHAP) methodology was used to develop interpretations for the linear & NNET regression models and the LDA & classification models. The usefulness of SHAP is that it allows for both global and local interpretations and there are extensive visualizations methods to analyze the global and local SHAP values.

VIII.I Future Work

For the regression database, new work is currently being done to develop pitching to a specific angle of attack instead of pitching based on a target. Pitching based on a target causes the missile to pitch over too quickly and drastically and causes the missile to tumble out of control. Despite the performance gains of the NNETs, they require long training times compared to traditional linear regression and are more complex to understand. One method for regression not used in this work was ensemble methods such as random forests or XGBoost and there are works which have shown that given the right database, these ensemble methods can perform similarly to NNETs but are extremely faster compared to NNETs. Ensemble methods are also glass-box methods, so they are directly interpretable. Of course, SHAP can also be used on ensemble methods as well.

For classification, this work only used thrust components, but in real life situations this information may not actually be known but was a very good test case. Future datasets will include other parameters more likely to be known such as TOF and MAXDIST. Of course, other methods such as ensemble methods should be utilized to see if they are capable of performance that NNETs can achieve.

For model imputation, generative adversarial networks (GANs) need to be utilized to try and impute missing data. There are experiments which suggest that GANs are more capable of imputing missing data because GANs essentially build two networks, one which generates samples and another which tries to see if the generate sample is real or fake. Another method which improves performance is adversarial learning, which is similar to GANs, but adversarial learning takes the batch data and adds noise based on the previous error gradient. Since the data is being imputed and is therefore noisier than the true data, then adversarial learning could possibly improve the accuracy. Exploration of more complex NNETs may also be the key for imputation; the networks which were developed in the imputation section may not have been sufficient, so wider and deeper network may be necessary for improving the results.

Further applications of SHAP are still needed to derive more information about model explainability. Other methods which were mentioned in Sect. VII.VII may also be useful for model interpretation. For future datasets, especially classification, missiles with autopilot need to be developed because in most modern missile applications, there is some autopilot which is influencing the missile's trajectory and will cause the dataset to be more nonlinear than it already is. Other regression and classification should also utilize the trajectory time information. Although this adds significant amount of data, more information could be extracted from the trajectory to develop new insights to missile technology. Physic informed neural network (PINNs) could also be utilized to directly learn from the equations of motions to reverse engineer features of the missile. In relation to PINNs, the reverse engineering process should be revisited to predict the missile parameters and launch configurations using output information such as TOF & MAXTHR or use the entire set of trajectory time data.

References

[1] Van Rossum, G., & Drake, F. L. Python 3 Reference Manual. Scotts Valley, CA: CreateSpace, 2009. https://www.python.org/

[2] Scikit-learn: Machine Learning in Python, JMLR 12, pp. 2825-2830, 2011. https://scikit-learn.org/stable/index.html#

[3] Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mane, D., Monga, R., Moore, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viegas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X., "TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems," tensorflow.org, TensorFlow 2015 Whitepaper, 2015. doi: 10.48550/arXiv.1603.04467

[4] Abadi, M., Barham, P., Chen, J., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., Kudlur, M., Levenberg, J., Monga, R., Moore, S., Murray, D., Steiner, B., Tucker, P., Vasudevan, V., Warden, P., Wicke, M., Yu, Y., and Zheng, X., "TensorFlow: A System for Large-Scale Machine Learning," tensorflow.org, 12th USENIX Symposium on Operating Systems Design and Implementation, Savannah, GA, 2016. doi: 10.48550/arXiv.1605.08695

[5] Anderson, M., *Design of a Missile Interceptor Using Genetic Algorithms*, Ph. D. Dissertation, Auburn University, Auburn, AL, 1998.

[6] Burkhalter, J.E., Jenkins, R., and Hartfield, R., "Genetic Algorithms for Missile Analysis," *MSIC Final Report*, 2003.

[7] Hartfield, R., Burkhalter, J.E., Jenkins, R., and Metts, J., "Genetic Algorithm Upgrade," *MSIC Final Report*, 2005.

[8] Hartfield, R.J., Burkhalter, J.E., Jenkins, R.M., Metts, J., Riddell, D., and Dyer, J., "Genetic Algorithm Developments for Liquid Missile Analysis," *MSIC Final Report*, 2006.

[9] Hartfield, R.J., Burkhalter, J.E., Jenkins, R.M., Dyer, J., and McDavid, B., "Genetic Algorithm Developments for Multiple Stage Missile Analysis," *MSIC Final Report*, 2007.

[10] Hartfield, R., Burkhalter, J.E., Hurston, W., Badyrka, J., Johnson, C., and Druckemiller, J., "Genetic Algorithm Guidance & Control Goal Definitions Upgrade," *MSIC Final Report*, 2009.

[11] Hartfield, R., Burkhalter, J.E., Jenkins, R., Carpenter, M., Albarado, K., Badyrka, J., Ritz, S., Hurston, B., and Ahuja, V., "Optimization Techniques for Advanced Missile Analysis," *MSIC Final Report*, 2011.

[12] Hartfield, R., Burkhalter, J.E., Jenkins, R., Carpenter, M., Albarado, K., Ritz, S., Ledlow, T., and Walsh, T., "Missile Design Optimization Methods," *MSIC Final Report*, 2012.

[13] Hartfield, R., Burkhalter, J.E., Carpenter, M., and Kiyak, Z., "Optimization & Search Algorithms for Missile Analysis," *MSIC Final Report*, 2013.

[14] Hartfield, R.J., "Missile System Design Using a Hybrid Evolving Swarm Algorithm," *MSIC Final Report*, 2014.

[15] Hartfield, R.J., Carpenter, M., Cervantes, N., Eckert, J., and Sheils, T., "Emerging Threat Analysis," *MSIC Final Report*, 2020.

[16] Hartfield, R.J., Carpenter, M., Yilmaz, L., Cervantes, N., Belvin, N., and DiMaggio, G., "Quantification of Confidence Level of Missile Models," *MSIC Final Report*, 2021.

[17] Hartfield, R.J., Carpenter, M., Yilmaz, L., Cervantes, N., Pastor, G., and Moore, N., "Engineering and Statistical Analysis of Liquid Missile Systems, Genetic Algorithms, & Augmented Intelligence with Exploratory Modeling and Analysis," *MSIC Final Report*, 2022.

[18] Hartfield, R., Rocket Propulsion Course Notes, Vol. 1-15.

[19] Bayley, D., Hartfield, R., Burkhalter, J.E., and Jenkins, R., "Design Optimization of a Space Launch Vehicle Using a Genetic Algorithm," *Journal of Spacecraft and Rockets*, Vol. 45, No. 4, 2008, pp. 733-740. doi: 10.2514/1.35318

[20] Riddle, D., Hartfield, R., Burkhalter, J. E., and Jenkins, R., "Genetic Algorithm Optimization of Liquid Propellant Missile Systems," 45th AIAA Aerospace Sciences Meeting and Exhibit, Reno, NV, 2007. doi: 10.2514/6.2007-362

[21] Riddle, D., Hartfield, R., Burkhalter, J.E., and Jenkins, R., "Genetic-Algorithm Optimization of Liquid-Propellant Missile Systems," *Journal of Spacecraft and Rockets*, Vol. 46, No. 1, 2009, pp. 151-159. doi: 10.2514/1.30891

[22] Carpenter, M., and Hartfield, R., "*Similitude for Dry Liquid Rocket Engines*," AIAA, 2018 Joint Propulsion Conference, Cincinnati, OH, June 2018. doi: 10.2514/6.2018-4548

[23] Sutton, G. P., and Biblarz, O., Rocket Propulsion Elements, 6th ed., Vol. Wiley-Interscience, New York, 2001.

[24] Huzel, D., and Huang, D. H., *Modern Engineering for Design of Liquid-Propellant Rocket Engines*, 1st ed., Vol. 147, AIAA, Washington, D.C., 1992.

[25] Humble, R. W., Henry, G. N., and Larson, W. J., *Space Propulsion Analysis and Design*, 2nd ed., The McGraw-Hill Companies, Inc., New York, 1995.

[26] Sutton, G. P., History of Liquid Propellant Rocket Engines, AIAA, 2006.

[27] Barrere, M., and Jaumotte, A., *Rocket Propulsion*, Revised ed., Elsevier Publishing Company, Amsterdam, 1960.

[28] Hartfield, R., Jenkins, R., Burkhalter, J. E., and Foster, W., "A Review of Analytical Methods for Solid Rocket Motor Grain Analysis," 39th AIAA/ASME/SAE/ASEE Joint Propulsion Conference, Huntsville, AL, 2003. doi: 10.2514/6.2003-4506

[29] Hartfield, R., Jenkins, R., Burkhalter, J.E., and Foster, W., "Analytical Methods for Predicting Grain Regression in Tactical Solid-Rocket Motors " *Journal of Spacecraft and Rockets*, Vol. 41, No. 4, 2004, pp. 689-693. doi: 10.2514/1.3177

[30] Anderson, M., Burkhalter, J. E., and Jenkins, R., "Multi-Disciplinary Intelligent Systems Approach to Solid Rocket Motor Design Part I: Single and Dual Goal Optimization," 37th AIAA/ASME/SAE/ASEE Joint Propulsion Conference and Exhibit, Salt Lake City, UT, 2001. doi: 10.2514/6.2001-3599

[31] Anderson, M., Burkhalter, J. E., and Jenkins, R., "Multi-Disciplinary Intelligent Systems Approach to Solid Rocket Motor Design Part II: Multiple Goal Optimization," 37th AIAA/ASME/SAE/ASEE Joint Propulsion Conference and Exhibit, Salt Lake City, UT, 2001. doi: 10.2514/6.2001-3600

[32] Hartfield, R., Jenkins, R., and Burkhalter, J. E., "Optimizing a Solid Rocket Motor Boosted Ramjet Powered Missile Using a Genetic Algorithm," 41st AIAA/ASME/SAE/ASEE Joint Propulsion Conference & Exhibit, Tucson, AZ, 2005. doi: 10.2514/6.2005-3507

[33] Metts, J., Hartfield, R., Burkhalter, J. E., and Jenkins, R., "Reverse Engineering of Solid Rocket Missiles with a Genetic Algorithm," 45th AIAA Aerospace Sciences Meeting and Exhibit, Reno, NV, 2007. doi: 10.2514/6.2007-363

[34] Albarado, K., Hartfield, R., Hurston, B., and Jenkins, R., "Solid Rocket Motor Performance Matching Using Pattern Search/Particle Swarm Optimization," 47th AIAA/ASME/SAE/ASEE Joint Propulsion Conference & Exhibit, San Diego, CA, 2011. doi: 10.2514/6.2011-5798

[35] Hartfield, R., Burkhalter, J. E., Jenkins, R., Anderson, M., and Witt, J., "Analytical Development of a Slotted Grain Solid Rocket Motor," 38th AIAA/ASME/SAE/ASEE Joint Propulsion Conference & Exhibit, Indianapolis, IN, 2002. doi: 10.2514/6.2002-4298

[36] Hartfield, R., Burkhalter, J.E., Jenkins, R., and Witt, J., "Analytical Development of a Slotted-Grain Solid Rocket Motor," *Journal of Propulsion and Power*, Vol. 20, No. 4, 2004, pp. 690-694. doi: 10.2514/1.11378

[37] Washington, W. D., *Computer Program for Estimating Stability Derivatives of Missile Configurations*, Army Missile Research Development and Engineering Lab Aeroballistics Directorate, Redstone Arsenal, AL, 1976.

[38] Washington, W. D., *Computer Program for Estimating Stability Derivatives of Missile Configurations - User's Manual*, Army Missile Research Development and Engineering Lab Aeroballistics Directorate, Redstone Arsenal, AL, 1976.

[39] Vukelich, S. R., Stoy, S. L., Burns, K. A., Castillo, J. A., and Moore, M. E., *Missile Datcom - Final Report*, AFWAL-TR-86-3091, Vol. 1, 1988.

[40] Vukelich, S. R., Stoy, S. L., and Moore, M. E., *Missile Datcom - User's Manual*, AFWAL-TR-86-3091, Vol. 2, 1988.

[41] Rosemo, C., Doyle, J., and Blake, W.B., "Missile Data Compendium (DATCOM)," *Final Report - AFRL-RQ-WP-TR-2014-0281*, 2014.

[42] McDaniel, M. H., *Dynamic Stability and Control: Methods and Developments*, Ph. D. Dissertation, Auburn University, Auburn, AL, 2016.

[43] Paul, J.L., Vasile, J.D., and DeSpirito, J., "Comparison of Aeroprediction Methods for Guided Munitions," *ARL-TR-9287*, 2021.

[44] Mason, L., Devan, L., Moore, F.G., and McMillan, D., "Aerodynamic Design Manual for Tactical Weapons," *NSWC TR 81-156*, 1981.

[45] Nielsen, J., *Missile Aerodynamics*, 1st ed., McGraw Hill Series in Missile and Space Technology, New York, NY, 1960.

[46] Etkin, B., and Reid, L. D., *Dynamics of Flight: Stability and Control*, 3rd ed., John Wiley & Sons, Inc., USA, 1996.

[47] Etkin, B., *Dynamics of Atmospheric Flight*, 3rd ed., Dover Publications, Inc., Mineola, New York, 2000.

[48] Schmidt, D. K., *Modern Flight Dynamics*, 1st ed., McGraw Hill New York, NY, 2012.

[49] Henderson, D.M., "Euler Angles, Quaternions, and Transformation Matrices," *NASA Technical Report*, 1977.

[50] Cooke, J.M., Zyda, M., Pratt, D.R., and McGhee, R.B., "NPSNET: Flight Simulation Dynamic Modeling Using Quaternions," *Presence Teleoperators & Virtual Environments*, Vol. 1, No. 4, 1994, pp. 404-420. doi: 10.1162/pres.1992.1.4.404

[51] Amoruso, M.J., "Euler Angles and Quaternions in Six Degree of Freedom Simulations of Projectiles," *ADA417259*, 1996, pp. 75. doi:

[52] Fehlberg, E., *Classical Fifth-, Sixth-, Seventh-, and Eighth-Order Runge-Kutta Formulas with Stepsize Control*, *NASA Technical Report R-287*, NASA, Washington, D. C., 1968.

[53] Butcher, J.C., "Coefficients for the Study of Runge-Kutta Integration Processes," *Journal of the Australian Mathematical Society*, Vol. 3, No. 2, 1962, pp. 185-201. doi: 10.1017/S1446788700027932

[54] Zarchan, P., *Tactical and Strategic Missile Guidance*, *Progress in Astronautics and Aeronautics*, 7th ed., Vol. 239, American Institute of Aeronautics and Astronautics, Inc., Reston, VA, 2019.

[55] Yanushevsky, R., *Modern Missile Guidance*, 2nd ed., CRC Press, Boca Raton, FL, 2019.

[56] Blakelock, J. H., *Automatic Control of Aircraft and Missiles*, 2nd ed., Wiley-Interscience Publication, New York, NY, 1991.

[57] Siouris, G. M., *Missile Guidance and Control Systems*, 1st ed., Springer-Verlag, New York, NY, 2004.

[58] Gibeau, D. G., *Missile Design PC TRAP: An Improved PC TRAP for Tactical Missile Design*, Dissertation, Calhoun: The NPS Institutional Archive, Monterey, CA, 1993.

[59] Moran, I., and Altilar, T., "Three Plane Approach for 3D True Proportional Navigation," AIAA Guidance, Navigation, and Control Conference and Exhibit, San Francisco, CA, 2005. doi: 10.2514/6.2005-6457

[60] Nesline, W., and Nesline, M., "How Autopilot Requirements Constrain the Aerodynamic Design of Homing Missiles," 1984 American Control Conference, San Diego, CA, 1984. doi: 10.23919/ACC.1984.4788471

[61] Anderson, M., Burkhalter, J. E., and Jenkins, R., "Intelligent Systems Approach to Designing an Interceptor to Defeat Highly Maneuverable Targets," 39th Aerospace Sciences Meeting and Exhibit, Reno, NV, 2001. doi: 10.2514/6.2001-1123

[62] Seyfert, B. E., *The Motion Control System of the Legendary SCUD-B Missile: Description and Mathematical Analysis*, RoseDog Books, Pittsburgh, PA, 2019.

[63] *Project Rock Nail/Rose Nail*, Vol. 1-6, U.S. Army Missile Intelligence Agency, Redstone Arsenal, AL, 1980.

[64] Scud Missile, https://en.wikipedia.org/wiki/Scud_missile

[65] OMEGA, U.S. Government-Off-The-Shelf Software.

[66] Ripley, B.D., "Computer Generation of Random Variables: A Tutorial," *International Statistic Review*, Vol. 51, No. 3, 1983, pp. 301-319. doi: 10.2307/1402590

[67] Hogg, R. V., Tanis, E. A., and Zimmerman, D. L., *Probability and Statistical Inference*, 9th ed., Pearson Education, Inc., 2015.

[68] Casella, G., and Berger, R. L., *Statistical Inference*, 2nd ed., Duxbury Thomson Learning, 2002.

[69] Marsaglia, G., and Zaman, A., "A New Class of Random Number Generators," *The Annals of Applied Probability*, Vol. 1, No. 3, 1991, pp. 462-480.

[70] McKay, M.D., Beckmen, R.J., and Conover, W.J., "A Comparison of Three Methods for Selecting Values of Input Variables in the Analysis of Output from a Computer Code," *Technometrics*, Vol. 21, No. 2, 1979, pp. 239-245. doi: 10.2307/1268522

[71] Santner, T. J., Williams, B. J., and Notz, W. I., *The Design and Analysis of Computer Experiments*, 2nd ed., Springer Series in Statistics, 2018.

[72] Fang, K. T., Li, R., and Sudijianto, A., Design and Modeling for Computer Experiments, Computer Science and Data Analysis Series, Chapman & Hall/CRC, 2006.

[73] Anderson, V. L., and McLean, R. A., *Design of Experiments: A Realistic Approach*, Marcel Dekker Inc., 1974.

[74] Pitard, F. F., *Theory of Sampling and Sampling Practice*, 3rd ed., CRC Press, 2019.

[75] Stein, M., "Large Sample Properties of Simulations Using Latin Hypercube Sampling," *Technometrics*, Vol. 29, No. 2, 1987, pp. 143-151.

[76] Sobol, I.M., "On the Distribution of Points in a Cube and the Approximate Evaluation of Integrals," *USSR Computational Mathematics and Mathematical Physics*, Vol. 7, No. 4, 1967, pp. 86-112. doi: 10.1016/0041-5553(67)90144-9

[77] Bratley, P., Fox, B.L., and Niederreiter, H., "Programs to Generate Niederreiter's Low-Discrepancy Sequences," *ACM Transactions on Mathematical Software*, Vol. 20, No. 4, 1994, pp. 494-495. doi: 10.1145/198429.198436

[78] Wu, N., *The Maximum Entropy Method*, Springer Series in Information Sciences, 1997.

[79] Knuth, D. E., *The Art of Computer Programming*, 3rd ed., Vol. 2, Addison-Wesley, 1998.

[80] Wichmann, B.A., and Hill, I.D., "Algorithm AS 183: An Efficient and Portable Pseudo-Random Number Generator," *Journal of the Royal Statistical Society, Series C (Applied Statistics)*, Vol. 31, No. 2, 1982, pp. 188-190. doi: 10.2307/2347988

[81] The Pandas Development Team, *Pandas*, Zenodo, Ver. 1.4.3, 2020. doi: 10.5281/zenodo.3509134

[82] McKinney, W., "Data Structures for Statistical Computing in Python," Proceedings of the 9th Python in Science Conference, 2010. doi: 10.25080/Majora-92bf1922-00a

[83] Hunter, J.D., "Matplotlib: A 2D Graphics Environment," *Computing in Science & Engineering*, Vol. 9, No. 3, 2007, pp. 90-95. doi: 10.1109/MCSE.2007.55

[84] Waskom, M.L., "seaborn: Statistical Data Visualization," *The Journal of Open Source Software*, Vol. 6, No. 60, 2021, pp. 3021. doi: 10.21105/joss.03021

[85] Wickham, H., *ggplot2: Elegant Graphics for Data Analysis*, Springer-Verlag, New York, 2016.

[86] Ritz, S., Hartfield, R., Dahlen, J., Burkhalter, J. E., and Woltosz, W. S., "Rapid Calculation of Missile Aerodynamic Coefficients Using Artificial Neural Networks," IEEE Aerospace Conference, Big Sky, MT, 2015. doi: 10.1109/AERO.2015.7119031

[87] Carpenter, M., Hartfield, R., Cervantes, N., and Thacker, J., "High Speed Modeling for Grid Fins Using a DNN Approach," 2020 IEEE Aerospace Conference, Big Sky, MT, 2020. doi: 10.1109/AERO47225.2020.9172259

[88] Carpenter, M., Hartfield, R., Thacker, J., and Cervantes, N., "A Deep Learning Approach to Surrogate Modelling of Missile Aerodynamic Performance," Asia Pacific International Symposium on Aerospace Technology, Gold Coast, Australia, 2019.

[89] Hartfield, R., Carpenter, M., and Cervantes, N., "Statistical Learning for Solid Propellant Performance," AIAA Propulsion and Energy 2021 Forum, 2021. doi: 10.2514/6.2021-3704

[90] Cervantes, N., and Carpenter, M., and Hartfield, R., "A Deep Learning Approach Neural Network Approach to Missile Systems using Liquid Propulsion," IEEE Aerospace Conference, Big Sky, Montana, 2022. doi: 10.1109/AERO53065.2022.9843826

[91] Montgomery, D. C., Peck, E. A., and Vining, G. G., *Introduction to Linear Regression Analysis*, 5th ed., Vol. John Wiley & Sons, Inc., Hoboken, NJ, 2012.

[92] James, G., Witten, D., Hastie, T., and Tibshirani, R., *An Introduction to Statistical Learning with Application in R*, 2nd ed., Springer, New York, 2021.

[93] Hastie, T., Tibshirani, R., and Friedman, J., *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*, 2nd ed., Vol. 1, Springer, 2016.

[94] Géron, A., *Hands-on Machine Learning with Scikit-Learn, Keras & TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*, 2nd ed., Vol. 1, O'Reilly Media, Inc., Sebastopol, CA, 2019.

[95] Bruce, P., Bruce, A., and Gedeck, P., *Practical Statistics for Data Scientists: 50+ Essential Concepts Using R and Python*, 2nd ed., O'Reilly Media, Inc., Sebastopol, CA, 2020.

[96] Myttenaere, A.D., Golden, B., Le Grand, B., and Rossi, F., "Mean Absolute Percentage Error for Regression Models," *ARXIV*, 2016, doi: arXiv.1605.02541

[97] Tofallis, C., "A Better Measure of Relative Prediction Accuracy for Model Selection and Model Estimation," *Journal of the Operational Research Society*, Vol. 66, 2015, pp. 1352-1362. doi: 10.1057/jors.2014.103

[98] Chollet, F., *Deep Learning with Python*, 2nd ed., Manning Publications Co., Shelter Island, NY, 2021.

[99] Aggarwal, C. C., *Neural Networks and Deep Learning*, 1st ed., Springer International Publishing, Yorktown Heights, NY, 2018.

[100] Murphy, K. P., *Probabilistic Machine Learning: An Introduction*, 1st ed., The MIT Press, Cambridge, MA, 2022.

[101] Buduma, N., Buduma, N., and Papa, J., *Fundamentals of Deep Learning*, 2nd ed., O'Reilly Media, Inc., Sebastopol, CA, 2022.

[102] Durr, O., Sick, B., and Murina, E., *Probabilistic Deep Learning: With Python, Keras, and TensorFlow Probability*, 1st ed., Manning Publications, Co., Shelter Island, NY, 2020.

[103] Goodfellow, I., Bengio, Y., and Courville, A., *Deep Learning*, ed., Vol. MIT Press, 2016.

[104] McCulloch, W.S., and Pitts, W., "A Logical Calculus of the Ideas Immanent in Nervous Activity," *Bulletin of Mathematical Biophysics*, Vol. 5, 1943, pp. 115-133. doi: 10.1007/BF02478259

[105] Rosenblatt, F., "The Design of an Intelligent Automaton," *Research Trends, Cornell Aeronautical Laboratory, Inc.*, Vol. 6, No. 2, 1958.

[106] Hebb, D. O., *The Organization of Behavior: A Neuropsychological Theory*, 1st ed., Psychology Press, 1949.

[107] Rumelhart, D.E., Hinton, G.E., and Williams, R.J., "Learning Internal Representations by Error Propagation," *Cognitive Science*, DTIC-ICS Report 8506, 1985.

[108] Ruder, S., "An Overview of Gradient Descent Optimization Algorithms," *ARXIV*, 2016. doi: 10.48550/arXiv.1609.04747

[109] Duchi, J., Hazan, E., and Singer, Y., "Adaptive Subgradient Methods for Online Learning and Stochastic Optimization," *Journal of Machine Learning*, Vol. 12, No. 61, 2011, pp. 2121-2159. doi: 10.5555/1953048.2021068

[110] Kingma, D.P., and Ba, J.L., "ADAM: A Method for Stochastic Optimization," *ARXIV*, 2014. doi: 10.48550/arXiv.1412.6980

[111] Dozat, T., "Incorporating Nesterov Momentum into Adam," 2016.

[112] O'Malley, T., Bursztein, E., Long, J., Chollet, F., Jin, H., and Invernizzi, L., *KerasTuner*, Keras, https://github.com/keras-team/keras-tuner, 2019.

[113] Hahnloser, R., Sarpeshkar, R., Mahowald, M., Douglas, R., and Seung, S., "Digital Selection and Analogue Amplification Coexist in a Cortex-Inspired Silicon Circuit," *Nature*, Vol. 405, 2000, pp. 947-951. doi: 10.1038/35016072

[114] Agarap, A.F., "Deep Learning using Rectified Linear Units (ReLU)," *ARXIV*, 2019. doi: 10.48550/arXiv.1803.08375

[115] Xu, B., Wang, N., Chen, T., and Li, M., "Empirical Evaluation of Rectified Activations in Convolution Network," *ARXIV*, 2015. doi: 10.48550/arXiv.1505.00853

[116] Clevert, D.A., Unterthiner, T., and Hochreiter, S., "Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs)," *ARXIV*, 2016. doi: 10.48550/arXiv.1511.07289

[117] Klambauer, G., Unterthiner, T., Mayr, A., and Hochreiter, S., "Self-Normalizing Neural Networks," *ARXIV*, 2017. doi: 10.48550/arXiv.1706.02515

[118] Cybenko, G., "Approximation by Superpositions of a Sigmoidal Function," *Mathematics of Control, Signals, and Systems*, Vol. 2, 1989, pp. 303-314. doi: 10.1007/BF02551274

[119] Lu, Z., Pu, H., Wang, F., Hu, Z., and Wang, L., "The Expressive Power of Neural Networks: A View from the Width," *ARXIV*, 2017. doi: 10.48550/arXiv.1709.02540

[120] Kidger, P., and Lyons, T., "Universal Approximation with Deep Narrow Networks," *ARXIV*, 2019. doi: 10.48550/arXiv.1905.08539

[121] Fedus, W., Zoph, B., and Shazeer, N., "Switch Transformers: Scaling to Trillion Parameter Models with Simple and Efficient Sparsity," *Journal of Machine Learning Research & ARXIV,*, Vol. 23, 2022, pp. 1-40. doi: 10.48550/arXiv.2101.03961

[122] O'Malley, T., Bursztein, E., Long, J., Chollet, F., Jin, H., and Invernizzi, L., *KerasTuner*, Keras, https://github.com/keras-team/keras-tuner, 2019.

[123] Albarado, K., Hartfield, R., Carpenter, M., Burkhalter, J. E., and Ritz, S., "Rapid Missile Classification for Early Launch Using Neural Networks," 10th annual U. S. Missile Defense Conference and Exhibit, Washington, DC, 2012.

[124] Carpenter, M., Speakman, N., and Hartfield, R., "Rapid Characterization of Munitions Using Neural Networks," AIAA Atmospheric Flight Mechanics Conference, San Diego, California, January 2016. doi: 10.2514/6.2016-0787

[125] Carpenter, M., Hartfield, R., and Zhou, L., "Statistical Learning for Trajectory Prediction," AIAA SCITECH FORUM, 2019. doi: 10.2514/6.2019-0429

[126] Eckert, J., Carpenter, M., Hartfield, R., and Cervantes, N., "Classification of Intermediate Range Missiles After Launch," AIAA SciTech 2020 Forum, Orlando, FL, 2020. doi: 10.2514/6.2020-1852

[127] Ghojough, B., and Crowley, M., "Linear and Quadratic Discriminant Analysis: Tutorial," *ARXIV*, 2019. doi: 10.48550/arXiv.1906.02590

[128] Little, R., and Rubin, D., *Statistical Analysis with Missing Data*, 3rd ed., Wiley & Sons, Inc., Hoboken, NJ, 2020.

[129] Schmitt, P., Mandel, J., and Guedj, M., "A Comparison of Six Methods for Missing Data Imputation," *Journal of Biometrics & Biostatistics*, Vol. 6, No. 1, 2015, pp. 1-6. doi: 10.472/2155-6180.1000224

[130] Azur, M., Stuart, E., Frangakis, C., and Leaf, P.J., "Multiple Imputation by Chained Equations: What Is It and How Does It Work?," *International Journal of Methods in Psychiatric Research*, Vol. 20, No. 1, 2011, pp. 40-49. doi: 10.1002/mpr.329

[131] Troyanskaya, O., Cantor, M., Sherlock, G., Brown, P., Hastie, T., Tibshirani, R., Botstein, D., and Altman, R., "Missing Value Estimation Methods for DNA Microarrays," *Bioinformatics*, Vol. 17, No. 6, 2001, pp. 520-525. doi: 10.1093/bioinformatics/17.6.520

[132] Yoon, J., Jordon, J., and Schaar, M.V.D., "GAIN: Missing Data Imputation using Generative Adversarial Nets," *ARXIV*, 2018. doi: 10.48550/arXiv.1806.02920

[133] Joseph, A. D., Nelson, B., Rubinstein, B. I. P., and Tygar, J. D., *Adversarial Machine Learning*, 1st ed., Cambridge University Press, New York, NY, 2019.

[134] Papernot , N., Faghri, F., Carlini, N., Goodfellow, I., Feinman, R., Kurakin, A., Xie, C., Sharma, Y., Brown, T., Roy, A., Matyasko, A., Behzadan, V., Hambardzumyan, K., Zhang, Z., Juang, Y.-L., Li, Z., Sheatsley, R., Garg, A., Uesato, J., Gierke, W., Dong, Y., Berthelot, D., Hendricks, P., Rauber, J., and Long, R., "Technical Report on the CleverHans v2.1.0 Adversarial Examples Library," *ARXIV*, 2018. doi: 10.48550/arXiv.1610.00768

[135] Nicolae, M.I., Sinn, M., Tran, M.N., Beuesser, B., Rawat, A., Wistube, M., Zantedeschi, V., Baracaldo, N., Chen, B., Ludwig, H., Molloy, I.M., and Edwards, B., "Adversarial Robustness Toolbox v1.0.0," *ARXIV*, 2019. doi: 10.48550/arXiv.1807.01069

[136] Shapley, L.S., "A Value for N-Person Games," *RAND Corporation Series*, Vol. No. P-295, 1952, pp. doi: 10.7249/P0295

[137] Lundberg, S.M., and Lee, S.-I., "A Unified Approach to Interpreting Model Predictions," *ARXIV*, Vol. No. 2017, pp. doi: 10.48550/ARXIV.1705.07874

[138] Lipovetsky, S., and Conklin, M., "Analysis of regression in game theory approach," *Applied Stochastic Models in Business and Industry*, Vol. 17, No. 4, 2001, pp. 319-330. doi: 10.1002/asmb.446

[139] Catav, A., Fu, B., Zoabi, Y., Meilik, A., Shomron, N., Ernst, J., Sankararaman, S., and Bachrach, R., "Marginal Contribution Feature Importance - An Axiomatic Approach for Explaining Data," *Proceedings for Machine Learning Research*, Vol. 139, 2021, pp. 1324-1335. doi: PMC8460841

[140] Molnar, C., *Interpretable Machine Learning: A Guide for Making Black Box Models Explainable*, 2nd ed., 2022.

[141] Strumbelj, E., and Kononenko, I., "Explaining Prediction Models and Individual Predictions with Feature Contributions," *Knowledge and Information Systems*, Vol. 41, 2014, pp. 647-665. doi: 10.1007/s10115-013-0679-x

[142] Lundberg, S.M., Erion, G.G., and Lee, S.-I., "Consistent Individualized Feature Attribution for Tree Ensembles," *ARXIV*, Vol. No. 2019, pp. doi: 10.48550/ARXIV.1802.03888

[143] Mitchell, R., Frank, E., and Holmes, G., "GPUTreeShap: Massively Parallel Exact Calculation of SHAP Scores for Tree Ensembles," *ARXIV*, 2022, doi: 10.48550/arXiv.2010.13972

[144] Shrikumar, A., Greenside, P., and Kundaje, A., "Learning Important Features Through Propagating Activation Differences," *ARXIV*, Vol. 70, No. 2017, pp. 3145-3153. doi: 10.48550/arXiv.1704.02685

[145] Galinkin, E., "Robustness in AI Explanation Methods," *ARXIV*, 2022. doi: 10.48550/arXiv.2203.03729

[146] Cooper, A., "Explaining Machine Learning Models: A Non-Technical Guide to Interpreting SHAP Analyses," 2021. https://www.aidancooper.co.uk/a-non-technical-guide-to-interpreting-shap-analyses/

Appendix A: Bell Nozzle Correction Factor

The empirical curves for the bell nozzle performance correction factor are shown from Huzel and Huang [24] in Eq. (9.1) and are only a function of the nozzle expansion ratio and fractional nozzle length. Both the AUSRC and AULRC can obtain the correction factors and interpolate for varying values of expansion ratio.

$$\begin{aligned}
\frac{A_e}{A_t} &= 40 \qquad \lambda = 0.163L_f^3 - 0.5425L_f^2 + 0.5956L_f + 0.7753 \\
\frac{A_e}{A_t} &= 30 \qquad \lambda = 0.1791L_f^3 - 0.5922L_f^2 + 0.6464L_f + 0.758 \\
\frac{A_e}{A_t} &= 20 \qquad \lambda = 0.2189L_f^3 - 0.7168L_f^2 + 0.7781L_f + 0.7114 \\
\frac{A_e}{A_t} &= 10 \qquad \lambda = 0.0081L_f^3 - 0.2283L_f^2 + 0.4219L_f + 0.7897 \\
\frac{A_e}{A_t} &= 5 \qquad \lambda = -0.0902L_f^3 - 0.0082L_f^2 + 0.2713L_f + 0.8184 \\
\frac{A_e}{A_t} &= 2 \qquad \lambda = -0.1744L_f^3 + 0.1864L_f^2 + 0.1304L_f + 0.8489
\end{aligned} \tag{9.1}$$

Appendix B: Nozzle Entrance Calculation

To calculate the nozzle entrance length (L_E), a series of geometric calculations are required from Fig. 96. First the distance to center of nozzle entrance (y_O) is calculated on Eq. (10.1). The radius of the upstream nozzle is circular and is set equal to the body radius ($r_C = r_B$). Next, we need the cosine of θ_i, shown on Eq. (10.2). The tangent of θ_i is then calculated on Eq. (10.3).

$$y_O = r_t + \sigma r_t \tag{10.1}$$

$$\cos\left(\theta_i\right) = \frac{y_i}{r_C - \sigma r} = \frac{r_B - y_O}{r_C - \sigma r_t} \tag{10.2}$$

$$\tan\left(\theta_i\right) = \frac{X_i}{y_i} = \frac{\sqrt{1-\cos^2\left(\theta_i\right)}}{\cos\left(\theta_i\right)} \tag{10.3}$$

$$L_E = r_C - X_i = r_C - y_i \tan\left(\theta_i\right) \tag{10.4}$$

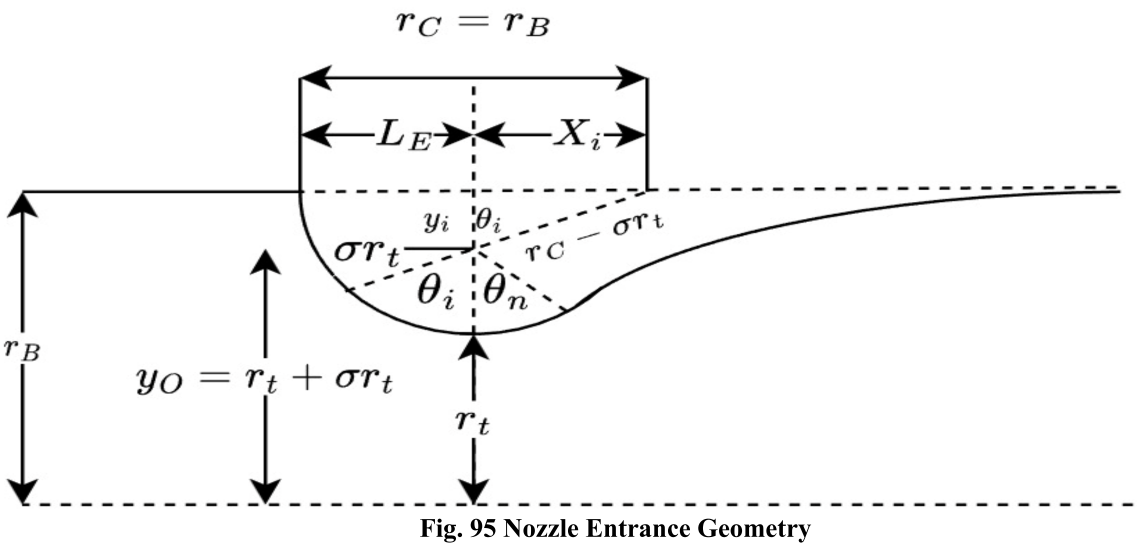
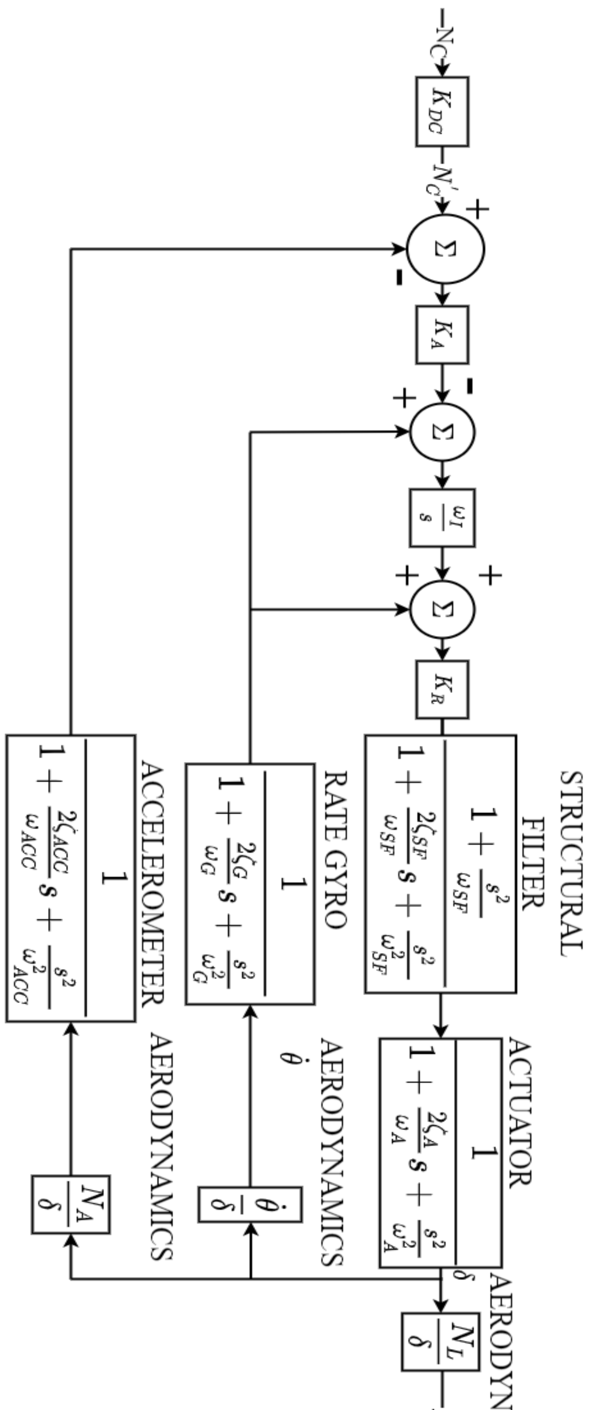


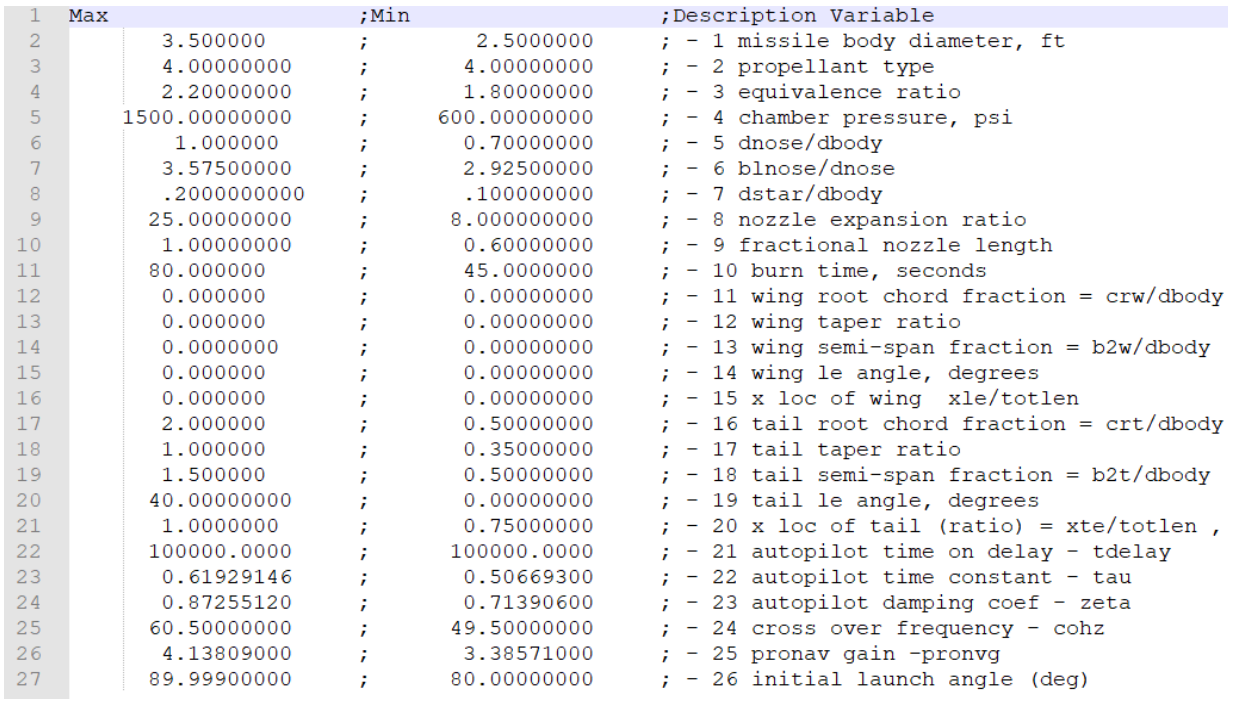
Fig. 95 Nozzle Entrance Geometry

Appendix C: 3-Loop Autopilot



$-N_C$
K_{DC}
N_C'
Σ
$+$
$-$
K_A
$\frac{\omega_I}{s}$
K_R
STRUCTURAL FILTER
$\frac{1 + \frac{s^2}{\omega_{SF}}}{1 + \frac{2\zeta_{SF}}{\omega_{SF}} s + \frac{s^2}{\omega_{SF}^2}}$
ACTUATOR
$\frac{1}{1 + \frac{2\zeta_A}{\omega_A} s + \frac{s^2}{\omega_A^2}}$
δ
AERODYNAMICS
$\frac{N_L}{\delta}$
N_L
RATE GYRO
$\frac{1}{1 + \frac{2\zeta_G}{\omega_G} s + \frac{s^2}{\omega_G^2}}$
AERODYNAMICS
$\dot{\theta}$
$\frac{\dot{\theta}}{\delta}$
ACCELEROMETER
$\frac{1}{1 + \frac{2\zeta_{ACC}}{\omega_{ACC}} s + \frac{s^2}{\omega_{ACC}^2}}$
AERODYNAMICS
$\frac{N_A}{\delta}$

Appendix D: AULRC No Autopilots LHC Input



```
Max                     ;Min                    ;Description Variable
       3.500000         ;        2.5000000      ; - 1 missile body diameter, ft
       4.00000000       ;       4.00000000      ; - 2 propellant type
       2.20000000       ;       1.80000000      ; - 3 equivalence ratio
    1500.00000000       ;     600.00000000      ; - 4 chamber pressure, psi
        1.000000        ;       0.70000000      ; - 5 dnose/dbody
       3.57500000       ;       2.92500000      ; - 6 blnose/dnose
       .2000000000      ;       .100000000      ; - 7 dstar/dbody
      25.00000000       ;      8.000000000      ; - 8 nozzle expansion ratio
       1.00000000       ;       0.60000000      ; - 9 fractional nozzle length
      80.000000         ;      45.0000000       ; - 10 burn time, seconds
       0.000000         ;       0.00000000      ; - 11 wing root chord fraction = crw/dbody
       0.000000         ;       0.00000000      ; - 12 wing taper ratio
       0.0000000        ;       0.00000000      ; - 13 wing semi-span fraction = b2w/dbody
       0.000000         ;       0.00000000      ; - 14 wing le angle, degrees
       0.000000         ;       0.00000000      ; - 15 x loc of wing  xle/totlen
       2.000000         ;       0.50000000      ; - 16 tail root chord fraction = crt/dbody
       1.000000         ;       0.35000000      ; - 17 tail taper ratio
       1.500000         ;       0.50000000      ; - 18 tail semi-span fraction = b2t/dbody
      40.00000000       ;       0.00000000      ; - 19 tail le angle, degrees
       1.0000000        ;       0.75000000      ; - 20 x loc of tail (ratio) = xte/totlen ,
      100000.0000       ;      100000.0000      ; - 21 autopilot time on delay - tdelay
       0.61929146       ;       0.50669300      ; - 22 autopilot time constant - tau
       0.87255120       ;       0.71390600      ; - 23 autopilot damping coef - zeta
      60.50000000       ;      49.50000000      ; - 24 cross over frequency - cohz
       4.13809000       ;       3.38571000      ; - 25 pronav gain -pronvg
      89.99900000       ;      80.00000000      ; - 26 initial launch angle (deg)
```

Fig. 96 AULRC LHC Input File for “gannlDIST.dat”.

Appendix E: AULRC Dataset Including Fin Autopilot

Table 43 AULRC Dataset Including Fin Autopilot

Parameter	Mean	STD DEV	MIN	25%	50%	75%	MAX
DBODY	3.02	0.29	2.50	2.78	3.03	3.27	3.50
KFUEL	2.00	0.12	1.80	1.90	2.00	2.10	2.20
PC	1107.13	245.44	600.00	911.48	1126.43	1318.45	1500.00
DNOSE	0.85	0.09	0.70	0.78	0.85	0.93	1.00
LNOSE	3.25	0.19	2.93	3.09	3.25	3.41	3.57
THROAT	0.16	0.03	0.10	0.14	0.16	0.18	0.20
EXPR	16.32	4.89	8.00	12.07	16.23	20.51	25.00
FNL	0.80	0.12	0.60	0.70	0.80	0.90	1.00
TBURN	60.16	9.67	45.00	51.83	59.18	67.88	80.00
TRCR	1.27	0.42	0.50	0.92	1.28	1.64	2.00
TTR	0.66	0.18	0.35	0.50	0.65	0.81	1.00
TAILB2	0.95	0.28	0.50	0.71	0.93	1.19	1.50
TLE	18.71	11.12	0.00	9.16	18.32	27.84	40.00
TXLERATIO	0.88	0.07	0.75	0.81	0.88	0.94	1.00
TDELAY	227	43	150	190	227	264	300
TAU	0.40	0.12	0.20	0.30	0.40	0.50	0.60
ZETA	0.85	0.09	0.70	0.77	0.85	0.92	1.00
COHZ	34.98	8.66	20.00	27.46	34.96	42.48	50.00
PRONVG	3.55	0.32	3.00	3.28	3.55	3.82	4.10
ILAUNCH	82.98	3.93	75.00	79.86	83.35	86.25	89.99
THRSEA	41.90	18.69	7.59	27.66	38.25	52.67	126.82
MAXTHR	46.92	20.84	8.35	30.93	43.06	59.10	142.69
MAXDIST	731.68	719.80	52.81	184.84	414.93	1115.86	6582.19
APOGEE	231.38	288.81	5.20	28.25	89.38	358.77	3287.15
TOF	241.57	148.95	45.21	116.48	193.24	354.50	1078.81
WEIGHT	16.81	6.47	5.19	11.89	15.53	20.49	51.18

Appendix F: AULRC Dataset Including Fin & Vane Autopilot

Table 44 AULRC Dataset Including Fin & Autopilot

Parameter	Mean	STD DEV	MIN	25%	50%	75%	MAX
DBODY	3.01	0.29	2.50	2.76	3.01	3.25	3.50
KFUEL	4.00	0.00	4.00	4.00	4.00	4.00	4.00
EQRATIO	2.00	0.12	1.80	1.90	2.00	2.10	2.20
PC	1072.66	247.22	600.00	869.93	1077.89	1282.46	1500.00
DNOSE	0.53	0.26	0.10	0.30	0.52	0.76	1.00
LNOSE	3.25	0.19	2.93	3.09	3.25	3.41	3.58
THROAT	0.15	0.03	0.10	0.13	0.15	0.17	0.20
EXPR	16.23	4.89	8.00	11.97	16.09	20.41	25.00
FNL	0.80	0.12	0.60	0.70	0.80	0.90	1.00
TBURN	57.02	11.06	40.00	47.41	55.84	65.78	80.00
TRCR	1.35	0.42	0.50	1.02	1.39	1.71	2.00
TTR	0.60	0.17	0.35	0.46	0.58	0.73	1.00
TAILB2	0.88	0.27	0.50	0.65	0.84	1.08	1.50
TLE	37.26	9.11	25.00	29.80	35.42	43.05	73.26
TXLERATIO	0.81	0.07	0.70	0.76	0.81	0.87	0.93
TDELAY	400.31	86.56	250.00	325.52	400.39	475.18	550.00
TAU	0.50	0.29	0.01	0.26	0.50	0.75	1.00
ZETA	0.90	0.35	0.30	0.60	0.90	1.20	1.50
COHZ	89.99	23.07	50.00	70.01	89.99	109.92	130.00
PRONVG	4.05	0.61	3.00	3.52	4.05	4.58	5.10
ILAUNCH	80.92	3.70	75.00	77.77	80.59	83.78	89.94
NOZDELAY	0.54	0.23	0.15	0.34	0.54	0.74	0.95
XK1	20.00	5.77	10.00	15.00	20.00	24.99	30.00
XK2	70.00	5.78	60.00	65.00	70.01	75.00	80.00
B2VAR	0.09	0.06	0.01	0.04	0.07	0.12	0.25
DELE0	0.25	0.14	0.00	0.13	0.25	0.37	0.50
DELR0	0.60	0.43	0.00	0.22	0.52	0.94	1.50
DTCHEK	0.80	0.40	0.10	0.45	0.80	1.15	1.50
DELTXZ	4499	867	3000	3748	4499	5249	6000
DELTXY	99993	11549	80000	89977	99986	110000	120000
THRSEA	36.80	16.00	7.85	25.02	33.45	45.15	127.42
MAXTHR	41.47	17.87	8.81	28.21	37.95	51.11	141.24
MAXDIST	578.12	571.31	52.80	165.28	357.02	818.19	6084.59
APOGEE	170.43	183.01	1.87	35.92	105.32	252.16	2857.22
TOF	218.63	109.24	41.84	126.22	204.71	295.51	987.64
WEIGHT	14.46	5.61	4.81	10.38	13.18	17.20	50.23

Appendix G: Output vs DBODY Scatter Plots

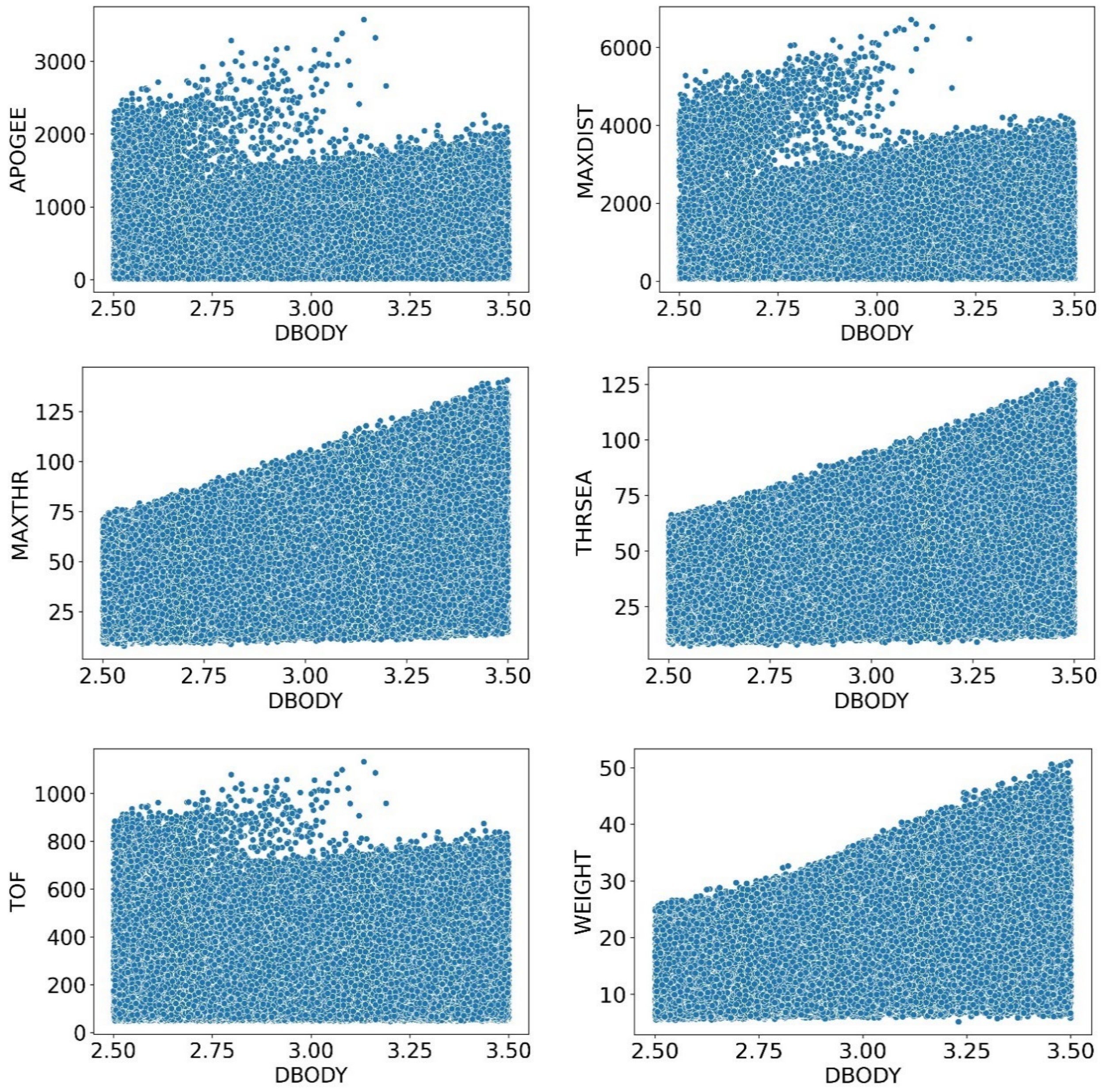


Fig. 97 (Top-Left) Apogee vs DBODY, (Top-Right) MAXDIST vs DBODY, (Mid-Left) MAXTHR vs DBODY, (Mid-Right) THRSEA vs DBODY, (Bottom-Left) TOF vs DBODY, (Bottom-Right) WEIGHT vs DBODY.

Appendix H: Output vs PC Scatter Plots

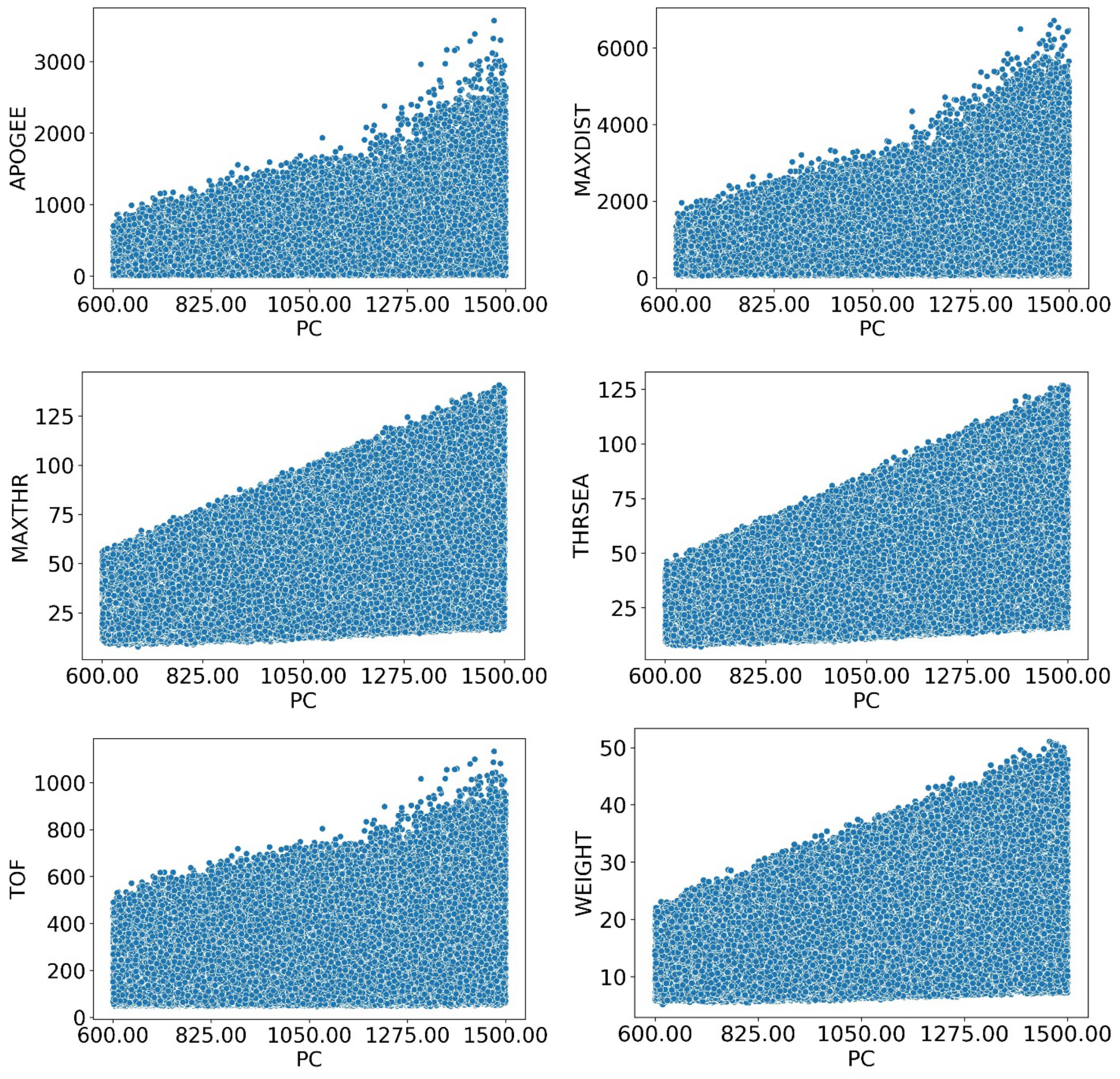


Fig. 98 (Top-Left) Apogee vs PC, (Top-Right) MAXDIST vs PC, (Mid-Left) MAXTHR vs PC, (Mid-Right) THRSEA vs PC, (Bottom-Left) TOF vs PC, (Bottom-Right) WEIGHT vs PC.

Appendix I: Output vs TAILB2 Scatter Plots

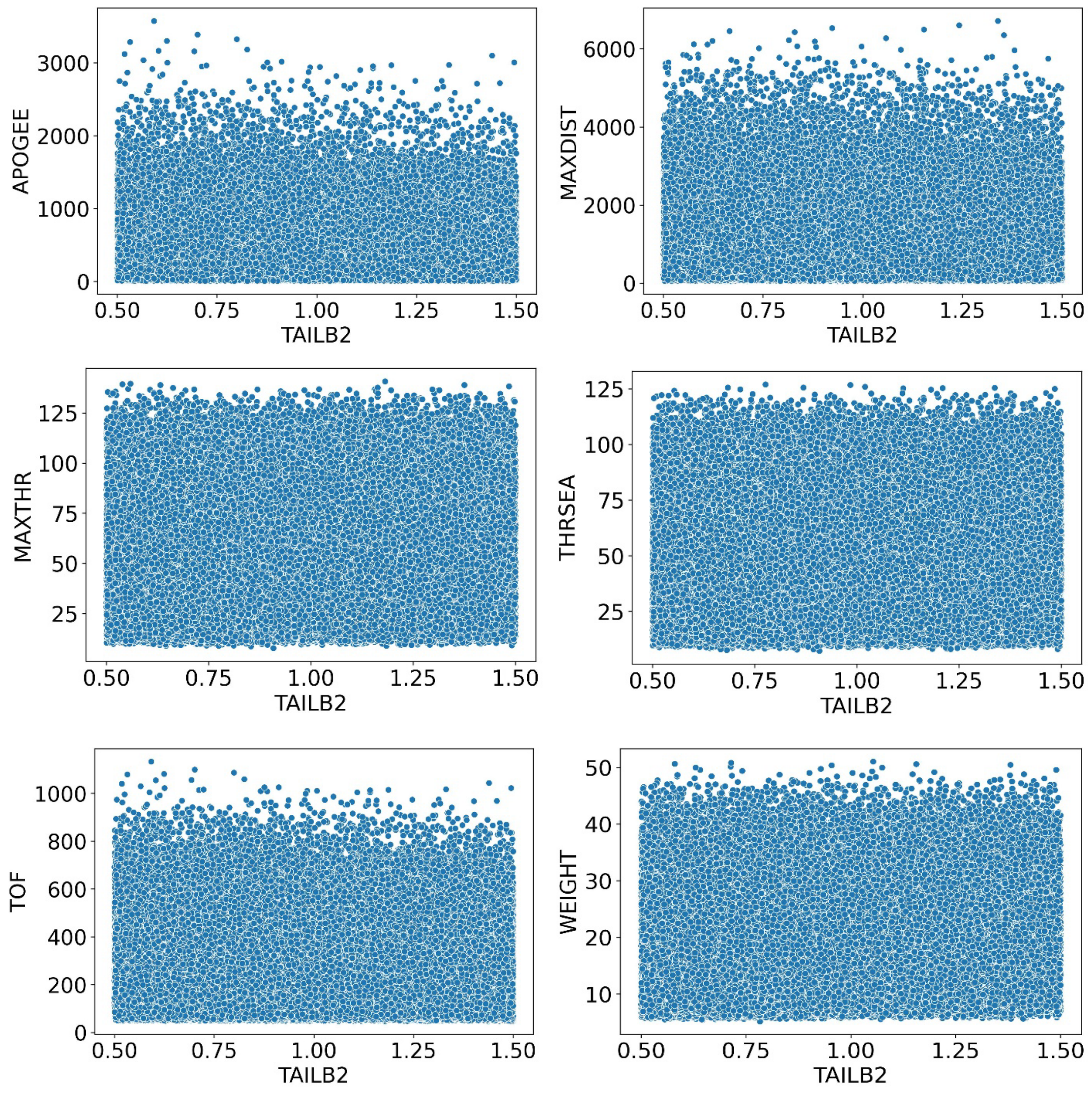


Fig. 99 (Top-Left) Apogee vs TAILB2, (Top-Right) MAXDIST vs TAILB2, (Mid-Left) MAXTHR vs TAILB2, (Mid-Right) THRSEA vs TAILB2, (Bottom-Left) TOF vs TAILB2, (Bottom-Right) WEIGHT vs TAILB2.

Appendix J: Output vs ILAUNCH Scatter Plots

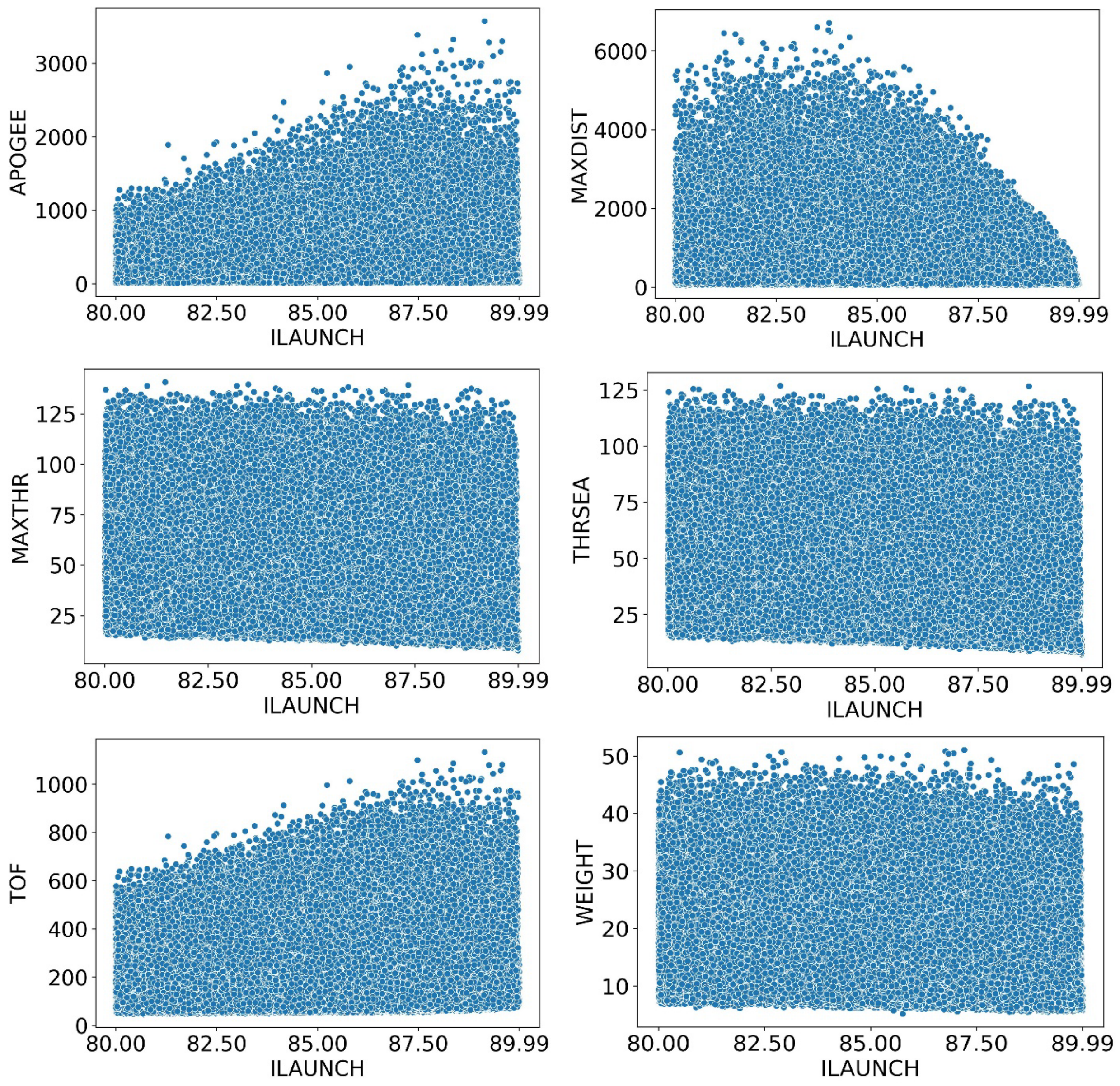
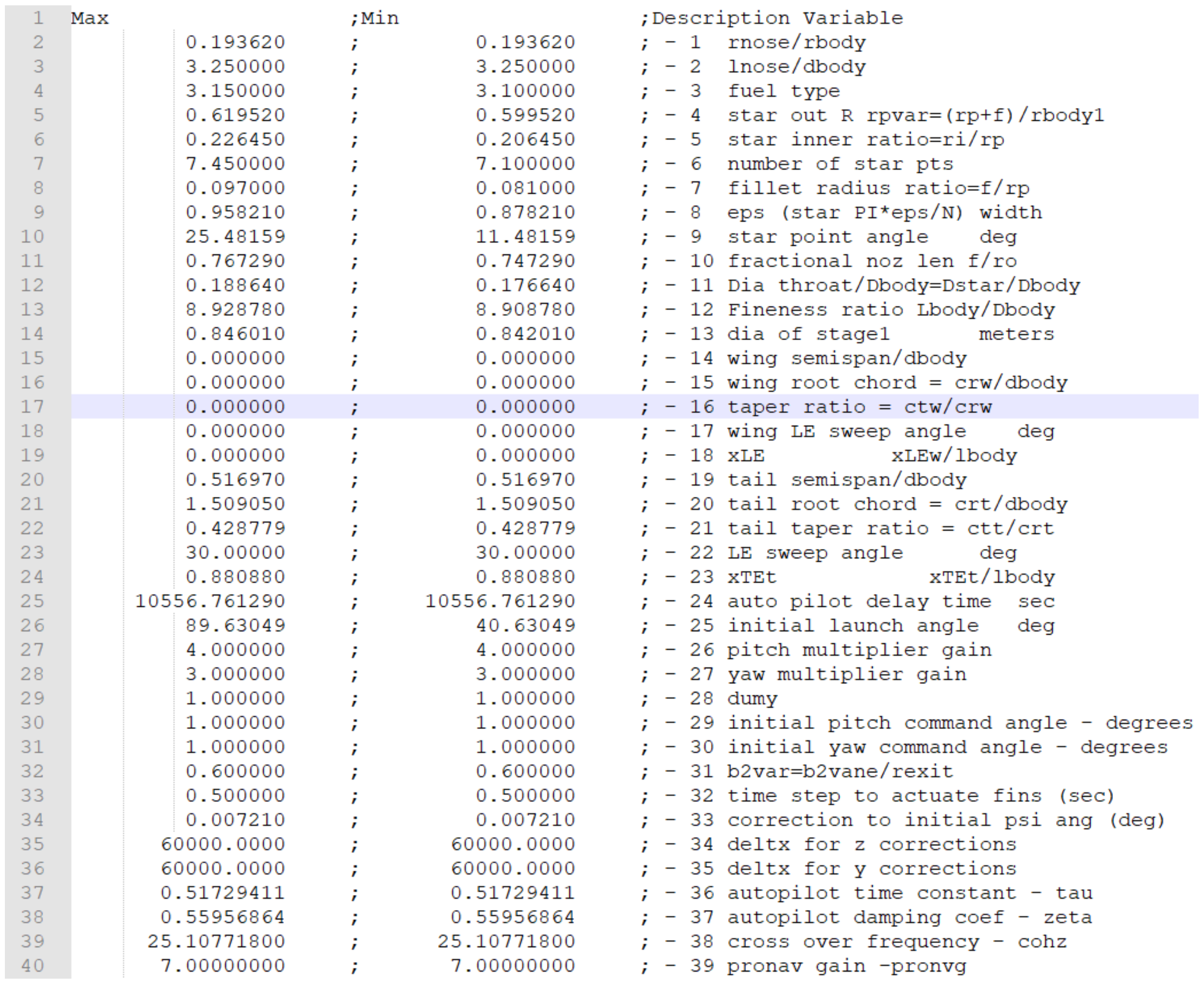


Fig. 100 (Top-Left) Apogee vs ILAUNCH, (Top-Right) MAXDIST vs ILAUNCH, (Mid-Left) MAXTHR vs ILAUNCH, (Mid-Right) THRSEA vs ILAUNCH, (Bottom-Left) TOF vs ILAUNCH, (Bottom-Right) WEIGHT vs ILAUNCH.

Appendix K: Class 10 LHC Input



```
Max                  ;Min                  ;Description Variable
        0.193620     ;        0.193620     ; - 1  rnose/rbody
        3.250000     ;        3.250000     ; - 2  lnose/dbody
        3.150000     ;        3.100000     ; - 3  fuel type
        0.619520     ;        0.599520     ; - 4  star out R rpvar=(rp+f)/rbody1
        0.226450     ;        0.206450     ; - 5  star inner ratio=ri/rp
        7.450000     ;        7.100000     ; - 6  number of star pts
        0.097000     ;        0.081000     ; - 7  fillet radius ratio=f/rp
        0.958210     ;        0.878210     ; - 8  eps (star PI*eps/N) width
        25.48159     ;        11.48159     ; - 9  star point angle    deg
        0.767290     ;        0.747290     ; - 10 fractional noz len f/ro
        0.188640     ;        0.176640     ; - 11 Dia throat/Dbody=Dstar/Dbody
        8.928780     ;        8.908780     ; - 12 Fineness ratio Lbody/Dbody
        0.846010     ;        0.842010     ; - 13 dia of stage1       meters
        0.000000     ;        0.000000     ; - 14 wing semispan/dbody
        0.000000     ;        0.000000     ; - 15 wing root chord = crw/dbody
        0.000000     ;        0.000000     ; - 16 taper ratio = ctw/crw
        0.000000     ;        0.000000     ; - 17 wing LE sweep angle    deg
        0.000000     ;        0.000000     ; - 18 xLE          xLEw/lbody
        0.516970     ;        0.516970     ; - 19 tail semispan/dbody
        1.509050     ;        1.509050     ; - 20 tail root chord = crt/dbody
        0.428779     ;        0.428779     ; - 21 tail taper ratio = ctt/crt
        30.00000     ;        30.00000     ; - 22 LE sweep angle      deg
        0.880880     ;        0.880880     ; - 23 xTEt           xTEt/lbody
    10556.761290     ;    10556.761290     ; - 24 auto pilot delay time   sec
        89.63049     ;        40.63049     ; - 25 initial launch angle   deg
        4.000000     ;        4.000000     ; - 26 pitch multiplier gain
        3.000000     ;        3.000000     ; - 27 yaw multiplier gain
        1.000000     ;        1.000000     ; - 28 dumy
        1.000000     ;        1.000000     ; - 29 initial pitch command angle - degrees
        1.000000     ;        1.000000     ; - 30 initial yaw command angle - degrees
        0.600000     ;        0.600000     ; - 31 b2var=b2vane/rexit
        0.500000     ;        0.500000     ; - 32 time step to actuate fins (sec)
        0.007210     ;        0.007210     ; - 33 correction to initial psi ang (deg)
      60000.0000     ;      60000.0000     ; - 34 deltx for z corrections
      60000.0000     ;      60000.0000     ; - 35 deltx for y corrections
      0.51729411     ;      0.51729411     ; - 36 autopilot time constant - tau
      0.55956864     ;      0.55956864     ; - 37 autopilot damping coef - zeta
     25.10771800     ;     25.10771800     ; - 38 cross over frequency - cohz
      7.00000000     ;      7.00000000     ; - 39 pronav gain -pronvg
```

Fig. 101 Input File "gannlDIST.dat" for Class 10.

Appendix L: 3rd Order Linear Regression Model

There are 1770 coefficients required for this model and include non-distinct parameters. Can see that the 3rd order model still cannot capture the second distribution of data.

Table 45 Individual & Overall Linear Model Metrics for Degree = 3

Metric	TOF	MAXTHR	MAXDIST	APOGEE	THRSEA	WEIGHT	Overall
R^2	0.98033	0.99974	0.91911	0.95961	0.99998	0.99930	0.97635
MSE	438.635	0.11007	44485.97	4087.39	0.00480	0.02936	8168.69
MAPE	5.77355	0.54726	36.43451	44.53611	0.15105	0.45511	14.6496
MMAPE	1.11014	0.16178	2.02188	0.89445	0.03869	0.15974	0.73111

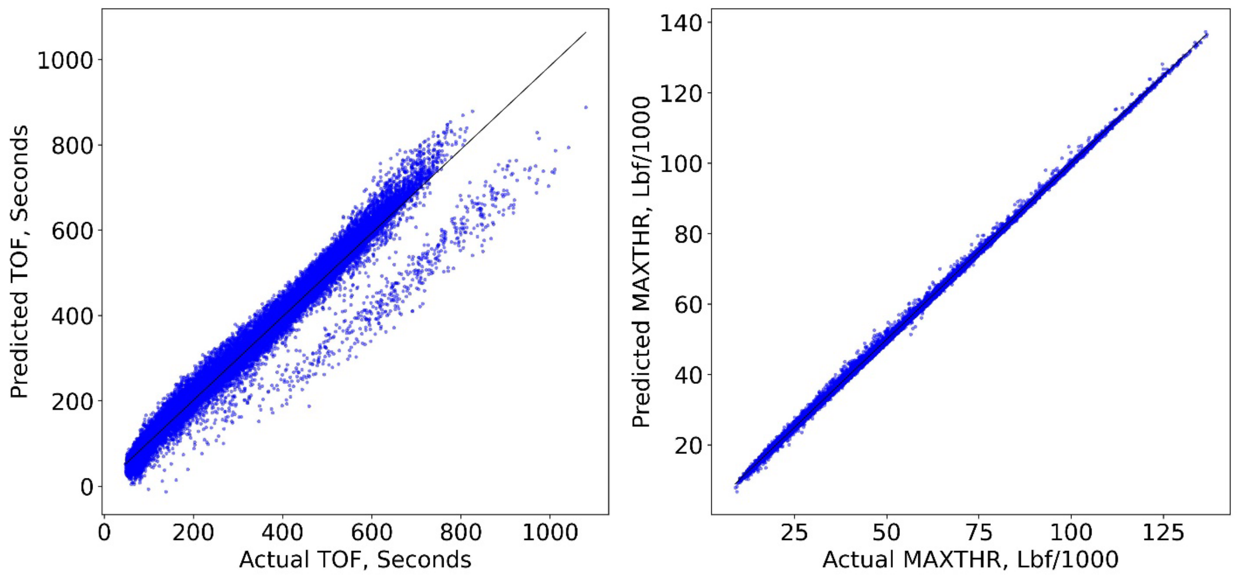


Fig. 102 Linear Three-Way Models: Predicted vs. Actual for TOF (Left) & MAXTHR (Right).

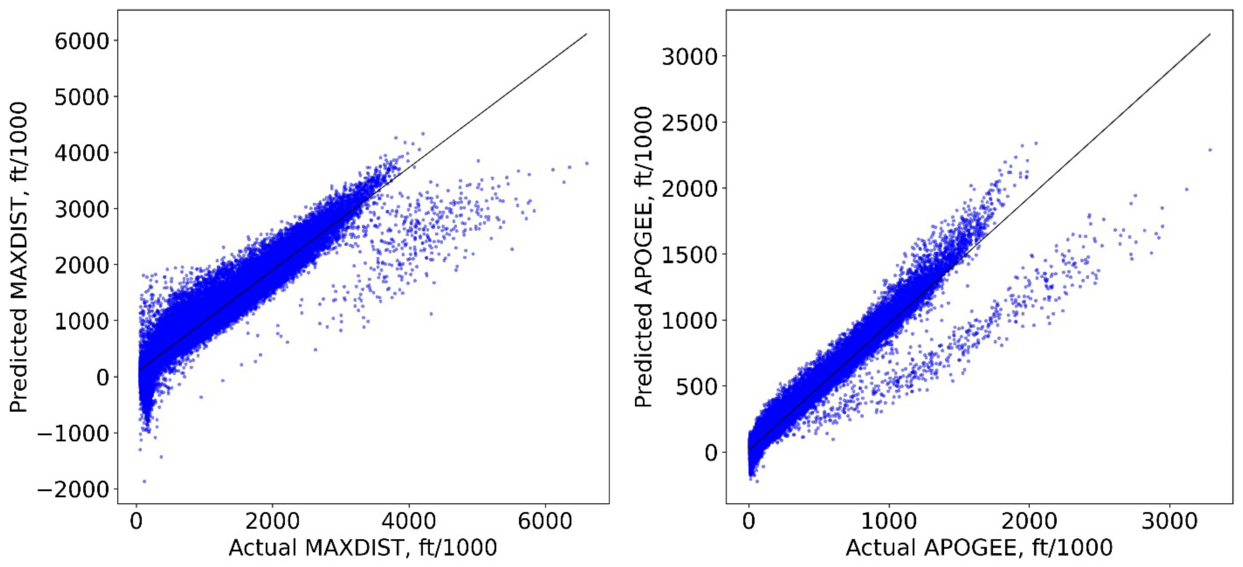


Fig. 104 Linear Three-Way Models: Predicted vs. Actual for MAXDIST (Left) & APOGEE (Right).

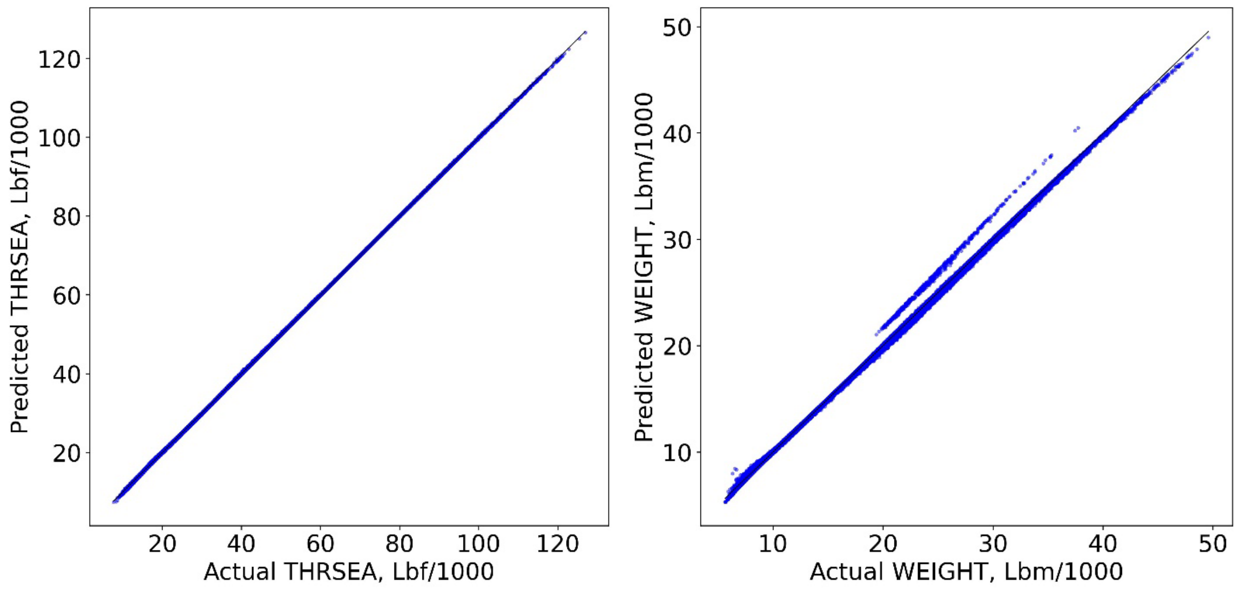


Fig. 103 Linear Three-Way Models: Predicted vs. Actual for THRSEA (Left) & WEIGHT (Right).

Appendix M: Hypothetical Shapley Value Example

Equation (21.1) shows the expanded summation calculation of the Shapley value for PC.

$$\begin{aligned}\phi(PC) = {} & \gamma(\varnothing)\left[v(PC) - v(\varnothing)\right] + \\ & \gamma(DBODY)\left[v(PC, DBODY) - v(DBODY)\right] + \\ & \gamma(BURNTIME)\left[v(PC, BURNTIME) - v(BURNTIME)\right] + \\ & \gamma(DBODY, BURNTIME)\left[v(DBODY, PC, BURNTIME) - \right. \\ & \qquad \left. v(DBODY, BURNTIME)\right]\end{aligned} \tag{21.1}$$

Equation (21.2) shows the expanded summation calculation of the Shapley value for BURNTIME.

$$\begin{aligned}\phi(BURNTIME) = {} & \gamma(\varnothing)\left[v(BURNTIME) - v(\varnothing)\right] + \\ & \gamma(DBODY)\left[v(BURNTIME, DBODY) - v(DBODY)\right] + \\ & \gamma(PC)\left[v(BURNTIME, PC) - v(PC)\right] + \\ & \gamma(DBODY, PC)\left[v(DBODY, PC, BURNTIME) - \right. \\ & \qquad \left. v(DBODY, PC)\right]\end{aligned} \tag{21.2}$$