Developing an Advanced Model Rocket Flight Simulation Software

Jorge Diaz Chao

Index

Introduction

Model rocketry is an excellent approach to the intricacies and applications of physics that spark curiosity and entertainment. It concerns insightful investigation, extensive experimentation and creative design to maximize reliability, a key factor within rocket science. Therefore, simulation software is frequently utilized to predict the flightpath of a rocket to minimize risks.

My interest in this topic originates from the development of an advanced model rocket, *Apex Beta*, I have been working on for over a year already, which uses thrust vector to control its attitude, along with a set of complex systems including a reaction wheel, avionics and a recovery system. An adventurous project I embarked on inspired by my keen enthusiasm towards engineering and space and the recent outstanding developments in aerospace technologies. However, no available simulation software is compatible with such complex systems featured in advanced model rocketry, which sets a perfect opportunity to deeply understand the physics of rockets and contribute to the community by developing simulation software for advanced model rockets, namely *Apex Dream*.

Research question

Therefore, this work aims to explore the extent to which different forces influence the flightpath of a rocket, accounting for thrust, weight and even aerodynamic forces, and more importantly, how to estimate them. This sets a fun physics challenges due to the lack of documentation on the chaotic nature of aerodynamics. Hence the research question:

How may the flightpath of a thrust vectored controlled model rocket be predicted through the consideration of thrust, weight and the aerodynamic forces acting on it?

Action plan:

The development of flight simulation software requires a detailed understanding of aerodynamics and reliable estimates. Therefore, the forces acting on a rocket will be first discuss at a theoretical level, and later estimated with empirical formulae, retrieved from experimental data and theoretical approaches. Finally, flight simulations will be compared to experimental data to verify the reliability of the product. Extensive research has been made within this topic; however, there is no defined mathematical model that describe the complex nature of aerodynamics. Thus, different models were selected and combined based on their reliability and similarity to experimental data.

Gravitational force

Weight

Weight is the resultant force exerted by the gravitational attraction on every component of a rocket. When added together, the gravitational forces may be viewed as a single force originating from the centre of gravity (CG ◔), whose location varies with the weight distribution of an object, as depicted by fig. 1. Calculating a rocket's gravitational force is therefore a simple matter of determining its total mass and CG.

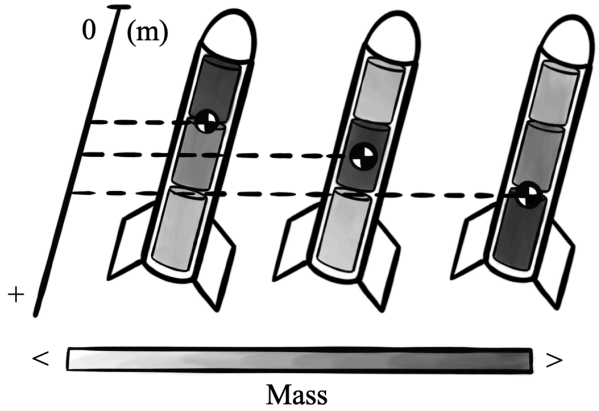

Fig. 1 - Variation of a rocket's CG location with weight distribution.

Torque

Moreover, a rocket's pivot point, around which every rotation will occur, is denoted by its CG. Any force applied to an object, if it does not pass through this point, will not only cause a translation, but also a rotational force referred to as torque τ, which may be used to calculate the angular acceleration α. Note that rotational forces do not interfere with linear forces.

$$\tau = d \cdot F sin(\theta) \quad (1)$$

$$\tau = I \cdot \alpha \quad (2)$$

Where I is the rotational inertia, which is the opposition exhibited by an object to the change in angular velocity, just like mass defines the resistance to linear acceleration. Its value is expressed in $kg \cdot m^2$, and may be calculated for each component of a rocket through standard formulae presented in ref. [1] (Nave, 2017) and combined to obtain the total moment of inertia.

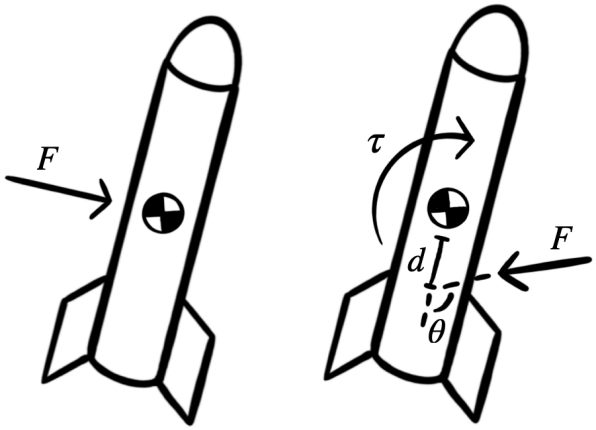

Fig. 2 - Torque exerted around a rocket's pivot point due to unaligned forces.

Propulsive force

Thrust

Thrust refers to the rocket's propulsive force, generated by exhausting gas or liquid in the opposite direction to that of the desired translation. Thus, the thrust of a motor is directly proportional to the velocity v_t of the working fluid and the mass per time unit $\dot{m}$ that is exhausted.

$$T = \dot{m} v_t + A_t(p_t - p_o) \quad (3)$$

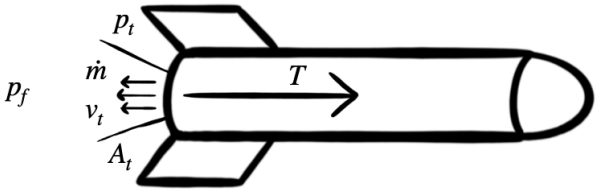

Fig. 3 - Thrust of a rocket and its components.

Were A_t is the exhaust area, p_t is the pressure of the working fluid, and p_o is the free stream pressure.

The thrust of commercial model rocket motors as a function of time have been measured in static fire tests and are readily available online at ref. [2] (Coker, 2022). This data may be used by the flight simulations to calculate the thrust at different times of the flight.

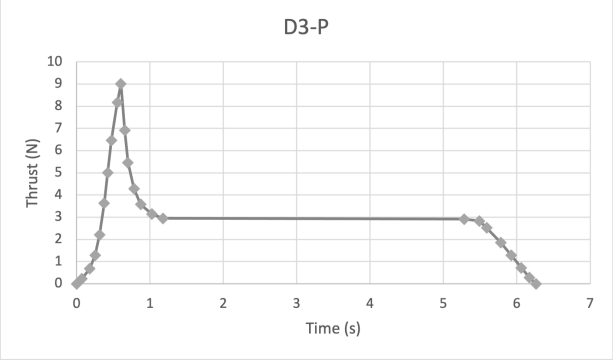

Fig. 4 - Thrust of a D3-P motor over time.

Thrust vectoring

Directing a rocket's thrust to control its angular velocity is known as thrust vectoring. Nominally, the thrust vector is aligned with the rocket's centreline, and thus, exerts no torque.

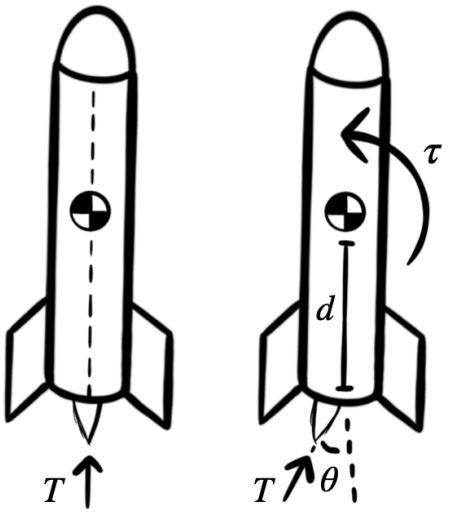

Fig. 5 - Torque exerted by thrust vectoring.

Otherwise, a pitching moment may be created by deflecting the thrust vector from the CG.

Aerodynamic Forces

In contact, fluids will exert a force on an object whenever there is a relative motion between both, and may be divided into drag D, tangent to the flow, and lift L, perpendicular to the flow. These forces are aerodynamic if the fluid is a gas, or hydrodynamic if it is a liquid.

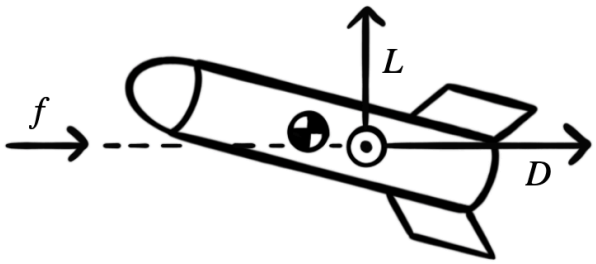

Fig. 6 - Components of aerodynamic forces relative to the flow.

Drag

Pressure drag is the outcome of forcing a fluid's flow around an object, detaching the boundary layer from the body generating a wake of recirculating flow. Whenever a fluid is deflected by an object, it will attempt to maintain contact with its surface, hindered by sharper edges. This leads to different fluid accelerations at diverse points perpendicular to the surface, and perhaps, flow separation if the changes of the fluid's velocity prevent it from adhering to the surface.

Acceleration is higher at the impact point and lower where the fluid tries to fill up the spaces, as seen in fig. 7. This creates a difference in pressure between the front and rear of an object, proportional to the resulting net force, retarding forward motion. Flow separation may

also cause vortex shedding, an oscillating flow that takes place when fluids flow as opposed to streamlined, which may cause undesired vibrations and instability.

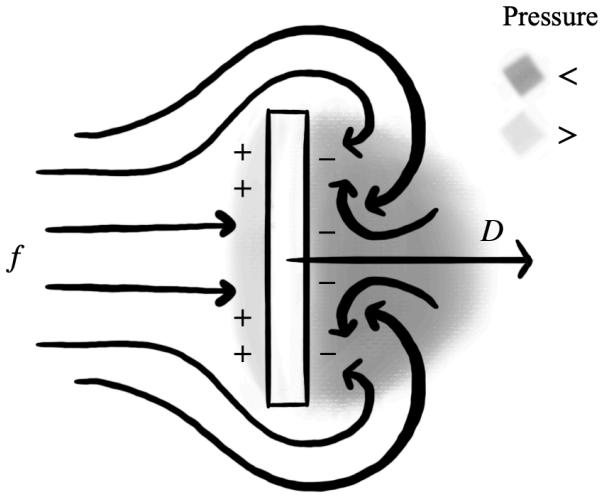

Fig. 6 - Pressure drag as a result of pressure variation and flow deflection.

Friction drag is the result of the friction between a fluid and an object's surface. Shear stresses create a region where flow varies from rest, near the walls of an object, to a maximum velocity at the main stream of the flow as shown by fig. 7. These shearing forces, that push different parts of a body or fluid in opposite directions simultaneously, rely on the fluid's viscosity to exert drag forces tangential to the surface and deductively, proportional to the exposed or wetted area. Greater shear stresses or velocity gradients translate into higher friction forces.

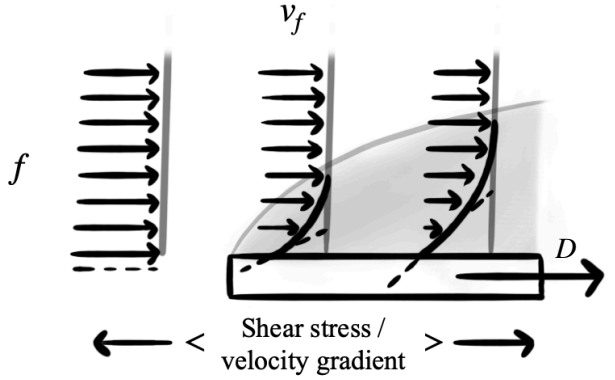

Fig. 7 - Friction drag as a result of greater shear stresses and velocity gradients.

The nearest fluid layer to a surface is identified as the boundary layer and might be laminar or turbulent, characterized by smooth paths and chaotic movement respectively. The Reynolds number, calculated using eq. (10), helps identify the type of flow and predict flow patterns, which will be looked at in more detail later in the paper. Drag forces under turbulent flow, associated with higher Re, are greater to those encountered in laminar flow, attributed to low Re, due to the more irregular interactions with an object's surface.

The resultant drag force may be obtained as suggested by ref. [3] (Nakayama, 2018) by integrating the pressure p and friction f contributions over the surface area A_w, if the distribution of these forces is known the drag. Hence:

$$D = \int_A (-p_d cos(\varphi) + f_d sin(\varphi)) dA \qquad (4)$$

Where φ is the angle between the flow vectors and the normal to the surface, such that the components of the pressure and friction tangential to the flow are obtained. However, this scenario is highly unlikely and requires complex computational fluid dynamics, so the following equation derived in ref. [4] (Maxemow, 2013) may be used instead:

$$D = \frac{1}{2} p_f v_f^2 C_D A_r \qquad (5)$$

The equation above requires determining a value for the drag coefficient C_D, which varies with each set of fluid conditions and object's shape and angle of attack. Experimentation demonstrates that for fixed drag coefficients the resultant drag appears to be proportional to

the pressure of the fluid p_f, the flow's velocity v_f squared and the reference area A_{ref}, commonly defined as the cross sectional area. Correlations from which eq. (5) may be obtained. Nonetheless, determining the C_D of an object becomes complex due to the multiple factors this coefficient accounts for and the chaotic nature of fluids. Therefore, it is typically done experimentally; however, for the purpose of this work a method to calculate a plausible approximation will be presented later.

Lift

Newton's third law of motion states that, for every action, there is an equal and opposition reaction, and explains how an object may be forced in the opposite direction to the deflected fluid. Variations in the angle of attack lead to different fluid deflection patterns, and thus, diverse lift contributions. Additionally, as the flow of a fluid is deflected over the surface of a body, its pressure varies at different points. Flow will be faster wherever there is more surface area to cover, leading to lower pressures. This uneven distribution of pressure across an object will force an object and contribute to the lift.

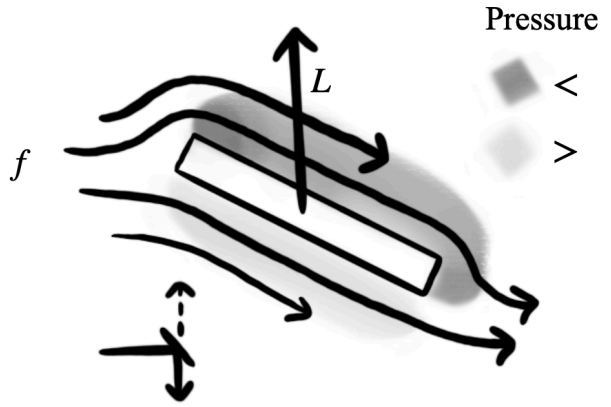

Fig. 8 - Lift due to a variation of pressure and a counter-action to the flow deflection.

Rockets generally have low lift to drag ratios as they already use thrust instead of lift to oppose weight, making this aerodynamic force undesirable in rocketry. However, lift should not be neglected given its significant variation upon the angle of attack and considerable effect on a rocket's flightpath.

Similar to drag calculations, lift force may be obtained by integrating the pressure p and friction f contributions over the surface area, if the distribution of these forces is known. Such that:

$$L = \int_A (-p_d sin(\varphi) + f_d cos(\varphi)) dA \qquad (6)$$

Where φ is utilized to obtain the components of the pressure and friction perpendicular to the flow. Otherwise, lift may be calculated with the the more pragmatic equation:

$$L = \frac{1}{2} p_f v_f^2 C_L A_r \qquad (7)$$

Formulae to estimate the lift coefficient C_L will be provided later on as an alternative to experimental calculations.

Aerodynamic Properties

As noted previously, drag and lift are oriented in respect to the flow's direction, being tangential and perpendicular to it respectively. However, for simplicity, these forces may be divided into an alternate set of perpendicular components, axial drag A, tangential to the body, and normal force N, perpendicular to the

body, for which their respective coefficients will be empirically calculated.

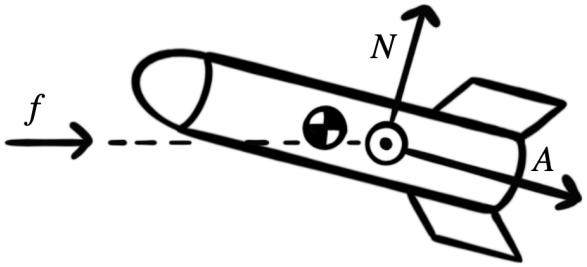

Fig. 9 - Components of aerodynamic forces relative to the rocket.

Therefore:

$$D = N sin(\alpha) + A\, cos(\alpha) \quad (8)$$

$$L = N cos(\alpha) - A\, sin(\alpha) \quad (9)$$

Flow

Note that the flow f of a fluid is relative to the velocity of an object. Despite the apparent original velocity v_w of the fluid to a stationary object or the wind for instance, the flow will tend to oppose the velocity of the object v_o as it gains speed and overweights the fluid's apparent velocity to an observer in rest. The flow's velocity v_f may be simply calculated by adding up the velocity vector of the wind and the inverse velocity vector of the rocket.

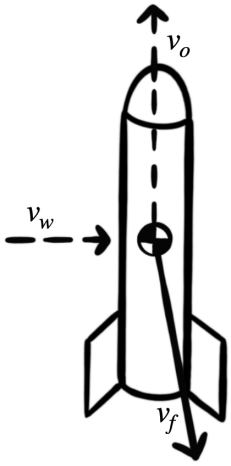

Fig. 10 - Relative flow obtained from the velocities of the wind and the rocket.

$$\overrightarrow{v_f} = \overrightarrow{v_w} - \overrightarrow{v_o} \quad (10)$$

Mach and Reynolds numbers

Aerodynamic properties may be calculated using the object's Mach number or the fluid's Reynolds number, due to the more apparent relations between aerodynamic forces and these ratios that describe the speed of an object or fluid with respect to other factors.

The Mach number M is the ratio of the speed of an object to the speed of sound c, which varies with the medium's density and temperature, assumed to be constant in this work given that model rockets typically fly at low altitudes, where differences are not significant. At higher speeds, a body will locally compress the fluid and alter its density, on which the aerodynamic forces depend. While compressibility factors may be ignored at subsonic speeds, they become important at transonic and supersonic speeds as discussed in ref. [5] (Hall, 2021).

$$M = \frac{v_o}{c} \quad (11)$$

The parameter β frequently appears in aerodynamical equations , characterizing the flow speed in subsonic and supersonic flow, such that as the flow speed tends to the transonic region at $M = 1$, β approaches 0.

$$\beta = \sqrt{M^2 - 1} \quad (12)$$

The Reynolds number Re is the ratio of inertial forces to viscous forces. This similarity parameter is used to identify flow patterns whenever a fluid is disturbed and forced across

an object, commonly categorized as laminar or turbulent, attributed to low and high Re respectively.

$$M = \frac{v_o}{c} \tag{13}$$

Angle of Attack

Aerodynamic properties are subject to the angle between the flow and an object't centreline, namely the angle of attack α.

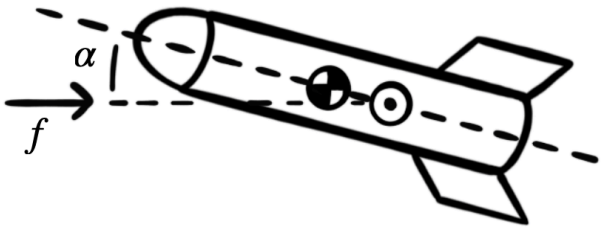

Fig. 11 - Angle fo attack relative to the flow.

This angle ranges from *0º* to *180º*, whenever the rocket's velocity opposes that of the flow and follow the same direction respectively.

Variations in the angle of attack translate into different fluid pressure patterns around an object as the flow's path is altered, and therefore, drag and lift forces as well.

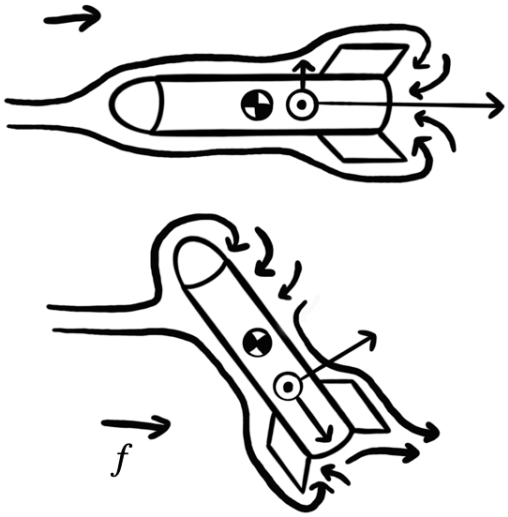

Fig. 12 - Variation of pressure with angle of attack.

Calculations may require a correction in the angle of attack α' to maintain values absolute:

$$\alpha' = \begin{cases} \alpha & \text{if } 0^\circ < \alpha \leq 90^\circ \\ 180 - \alpha & \text{if } 90^\circ < \alpha \leq 180^\circ \end{cases} \tag{14}$$

Rocket stability

Similar to an object's CG, the individual aerodynamic or hydrodynamic forces may be combined into a single force acting on the centre of pressure (CP ⊙). This point is rarely located where the CG is, so aerodynamic forces will induce a torque around the rocket's pivot point. Integrating the pressure over the surface area of an object will determine the position of this point. An approximation to calculate the CP's location is presented in Appendix A.

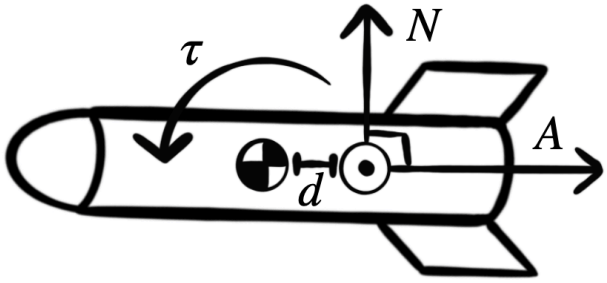

Fig. 13 - Torque exerted around the rocket's CG by the aerodynamic forces acting on the CP.

This pitching moment is mostly exerted by the aerodynamic forces perpendicular to the rocket's centreline, otherwise identified as the normal forces, greater at angles attack nearer to 90˚. Stable rockets tend to fly at minimum angles of attack opposing the flow, as otherwise they would be accelerating towards undesirable directions. Aerodynamic forces acting on the CP of a rocket flying at an angle of attack will produce a pitch so that the CG is always in front of the CP with respect to the flow. Therefore, if the CP of a rocket is above

or before its CG, the rocket is naturally unstable, as at the aerodynamic forces will tilt the rocket to a position where the thrust is acceleration the rocket in a undesired direction, and perhaps, lead to a loss of control and make it spin uncontrollably, as pictured in fig. 14. However, if the CP is brought below or behind the CG by adding some fins at the base of the rocket, increasing the pressure at their location, for example, the corrective pitching moment produced by the aerodynamic forces will keep the rocket facing against the flow and fly in the desired orientation as depicted by fig. 15.

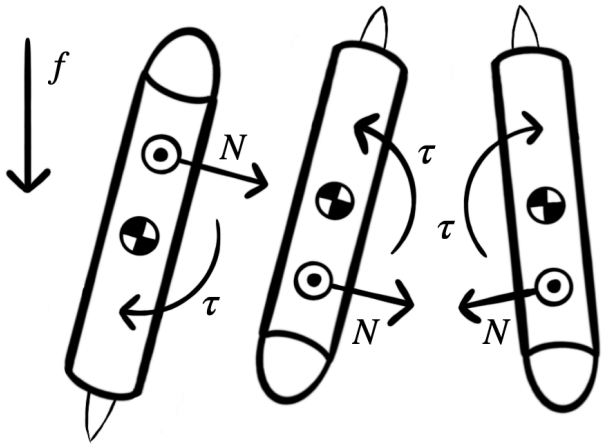

Fig. 14 - Pitching moment due to aerodynamic forces on a naturally unstable rocket.

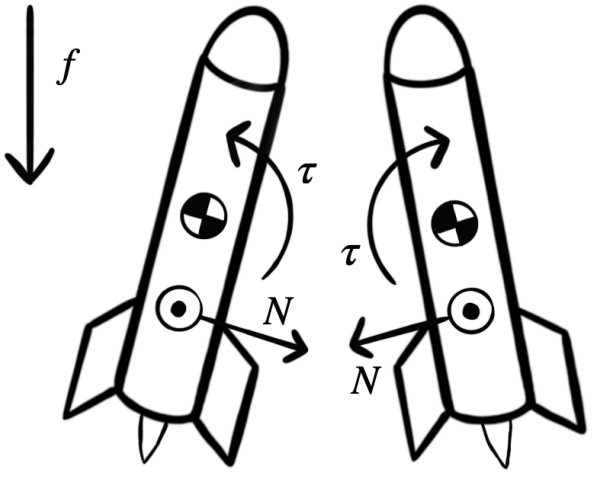

Fig. 15 - Corrective pitching moment due to aerodynamic forces on a stable rocket.

Note that the torque is proportional to the distance between the pivot point and wherever the force is applied, that is, the CG and CP respectively. Thus, the farther this two points are located from each other, the greater the pitching moment.

Axial Coefficient

The axial drag A is the component of the aerodynamic forces tangential to the rocket's centreline, for which eq. (5) may be used interchangeably by replacing the drag coefficient with the axial drag coefficient C_A, expressed as the combination of the friction drag C_{A_f}, body C_{A_p}, base C_{A_b} and fin C_{A_n} pressure coefficients at an angle of attack i.

$$C_{A_{\alpha=i}}(C_{A_f} + C_{A_p} + C_{A_b} + C_{A_n})_{\alpha=i} \qquad (15)$$

The approximation of the Ca value at various angles of attack requires its calculation at angles of attack of 0^o and 180^o as estimated by ref. [6] (Jorgensen, 1973). This increases reliability since the coefficient of the closest angle is used, such that:

$$C_A = \begin{cases} C_{A_{\alpha=0°}} \cdot cos(\alpha')^2 & \text{if } 0° < \alpha \leq 90° \\ C_{A_{\alpha=180°}} \cdot cos(\alpha')^2 & \text{if } 90° < \alpha \leq 180° \end{cases} \qquad (16)$$

Friction drag coefficient

The friction coefficient is the contribution to the axial drag coefficient due to friction drag. Methods for estimating its value under laminar and turbulent flow are determined by the different scopes of Reynolds number. These ranges are somewhat denoted by the rocket's critical Reynolds number Re_c, the point at which the flow becomes turbulent, defined by the approximate roughness height R_s of its

surface and the length of the rocket in ref. [7] (Barrowman, 1967):

$$Re_c = 51 \left(\frac{R_s}{l} \right)^{-1.039} \tag{17}$$

Some approximate roughness heights for different surfaces retrieved from ref. [8] are listed hereunder:

Type of surface	Height (μm)
Average glass	0.1
Finished and polished surface	0.5
Optimum paint-sprayed surface	5
Planed wooden boards	15
Paint in aircraft mass production	20
Dip-galvanized metal surface	150
Incorrectly sprayed aircraft paint	200
Raw wooden boards	500

Tab. 1 - Approximate roughness heights for common surfaces.

From theoretical and experimental data presented in ref. [8] (Cheeseman, 1976) the following piecewise function may be retrieved to calculate the skin friction coefficient C_f:

$$C_f = \begin{cases} 1.48 \cdot 10^{-2} & \text{if } Re < 10^4 \\ \frac{1}{(1.5\, ln(Re) - 5.6)^2} & \text{if } 10^4 \leq Re \leq Re_c \\ 0.032(\frac{R_s}{L})^{0.2} & \text{if } Re_c < Re \end{cases} \tag{18}$$

Where the coefficient at Re below 10^4, for which the experimental formulae are not applicable, typically encountered at velocities below $1\ m/s$, is assumed to be constant. The critical Reynolds number defines the point at which the skin friction coefficient can be considered independent of Re.

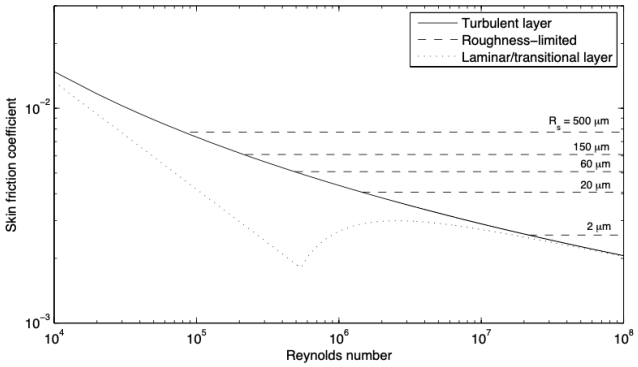

Fig. 16 - Skin friction coefficient of turbulent, laminar and roughness-limited boundary layers.

Someone would expect more friction drag on turbulent flow; however, higher Reynolds numbers translate into lower ratios of viscous to inertial forces. Higher velocities, to which greater Reynolds numbers are mostly related, will increase inertial forces while decreasing frictional forces.

Compressibility corrections to account for the deviation of the thermodynamic properties of a fluid and its modified density as an object moves faster through it may be applied.

At subsonic speeds the corrected coefficient C_f' is independent from Re. In supersonic flow however, the coefficient for Reynolds numbers lower and higher than critical, turbulent and roughness limited layers respectively, may be corrected differently.

$$C_f' = \begin{cases} C_f(1 - 0.1M^2) & \text{if } M < 1 \\ \frac{C_f}{(1 + 0.15M^2)^{0.58}} & \text{if } 1 \leq M \text{ and } Re \leq Re_c \\ \frac{C_f}{1 + 0.18M^2} & \text{if } 1 \leq M \text{ and } Re_c \leq Re \end{cases} \tag{19}$$

The optimized coefficient can then be used to calculate the resultant friction coefficient by scaling it to the appropriate wetted area common to a reference area. The body and fins wetted area are corrected for its cylindrical

geometry and finite thickness respectively. Interactions with the fluid will be higher with greater wetted areas, deductively proportional to the friction drag coefficient C_{A_f}:

$$C_{A_f} = C'_f \frac{(1 + \frac{1}{2f_B}) \cdot A_{w_b}}{A_r} + \frac{(1 + \frac{2t}{\zeta}) \cdot A_{w_f}}{A_r} \tag{20}$$

Where f_B is the fineness ratio of the rocket, t the thickness and ζ the mean aerodynamic chord length of the fins.

Base pressure drag coefficient

The base pressure coefficient C_b accounts for the low pressure area generated at the base of the rocket, where the body radius declines briskly. The coefficient may be estimated using the following empirical formula from ref. [9] (Moore, 1994), obtained from the experimental data referenced in ref. [10] (Moore, 1992):

$$C_b = \begin{cases} 0.12 + 0.13M^2 & \text{if } M < 1 \\ 0.25 / M & \text{if } 1 \leq M \end{cases} \tag{21}$$

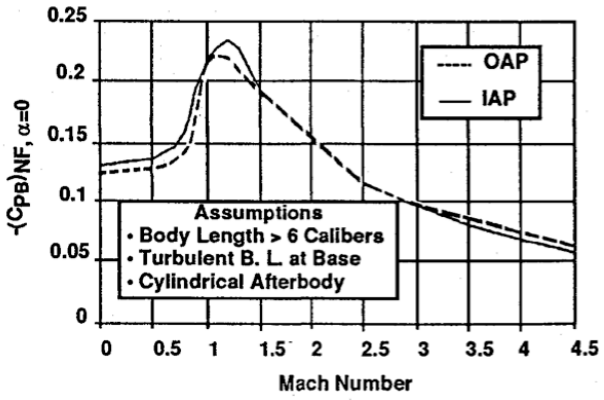

Fig. 17 - Empirical data of the base pressure drag coefficient variation with the Mach number.

The coefficient is at its peak throughout the transonic range because of the shock waves formation and general flow instabilities.

The base pressure coefficient can then be scaled to a common reference area to obtain the base drag contribution C_{A_b}.

$$C_{A_b} = \frac{A'_b}{A_r} C_b \tag{22}$$

Where the amended base area A'_b accounts for the disruption from the exhaust of a motor into the area. An approximation is achieved by subtracting the area of the thrusting propulsive systems A_m from the base area A_b. Therefore, if the base is the same size as the motor itself, there will be no base drag. In the contrary, if the base area is significantly greater than the motor area, the base drag is similar to whenever the rocket is coasting.

$$A'_b = A_b - A_m \tag{23}$$

Tip pressure drag coefficient

The pressure drag coefficients of streamlined rocket tips have been measured by ref. [11] (Hoerner, 1965) and are presented by the following figure:

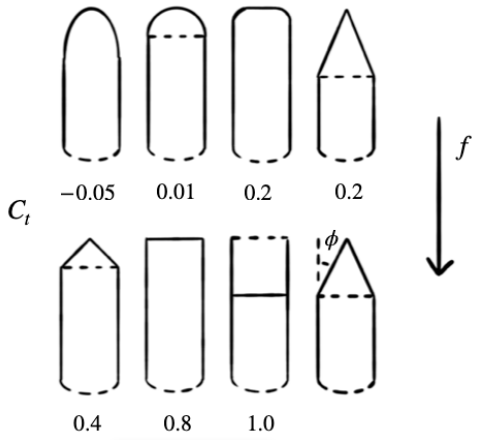

Fig. 18 - Experimental data of the pressure drag of various rocket tip geometries.

The slightest rounding at the connection between the tip and the body, defined by the

joint angle ϕ, reduces flow separation, and thus, the drag coefficient considerably. Appreciable pressure drag is encountered if the transition is not smooth due to flow separation.

The pressure drag C_T of a rocket tip may therefore be calculated with the piecewise function obtained from the additional experimental data discussed in ref. [12] (Convict & Faro, 1961):

$$C_T = \begin{cases} a \cdot M^b + C_{T_{M=0}} & \text{if } M < 1 \\ \frac{C_{T_{M=1.3}} - C_{T_{M=1}}}{0.3}(M-1) + C_{T_{M=1}} & \text{if } 1 \le M \le 1.3 \\ 2.1\, sin^2(\varepsilon) + \frac{0.5\, sin(\varepsilon)}{\beta} & \text{if } 1.3 < M \end{cases} \quad (24)$$

Where a and b are computed to interpolate the drag coefficient between the low subsonic and transonic flow and its derivatives, following the constraints:

$$C_{T_{M=0}} = 0.8 \cdot sin^2(\phi) \quad (25)$$

$$C_{T_{M=1}} = sin(\varepsilon) \quad (26)$$

$$\frac{\partial C_T}{\partial M} \Big|_{M=1} = \frac{4 - 2C_{t_{M=1}}}{\gamma + 1} \quad (27)$$

The value for the specific heat ratio of air is introduced by $\gamma = 1.4$, and ε is the angle between the conical body and the body centreline:

$$tan(\varepsilon) = \frac{d}{2\,l} \quad (28)$$

Where lower fineness or diameter d to length l ratios translate into higher pressure drag forces.

For blunt cylinders or rocket tips, which may be accounted for at angles of attack over 90° where the rocket's aft is facing against the flow, the pressure drag coefficient may be considered the same as the stagnation pressure, or the pressure at areas perpendicular to the flow, defined by ref. [11] (Hoerner, 1961) as:

$$C_q = 0.85 \cdot \begin{cases} 1 + \frac{M^2}{4} + \frac{M^4}{40} & \text{if } M < 1 \\ 1.84 - \frac{0.76}{M^2} + \frac{0.166}{M^4} + \frac{0.035}{M^6} & \text{if } 1 \le M \end{cases} \quad (29)$$

The coefficient C_T should then be scaled to the tip's front area A_t, common to the rocket's reference area, to get the resultant contribution C_t to the axial drag coefficient:

$$C_t = \frac{A_t}{A_r} C_T \quad (30)$$

Transition and boattail pressure drag

The pressure drag C_j of transitional rocket components, like the one identified in fig. 19, is assumed to be the same as that of a tip, scaled to the difference in area between the fore A_{j_f} and aft A_{j_a} ends of the transition.

$$C_j = \frac{A_{j_f} - A_{j_a}}{A_r} C_T \quad (31)$$

The pressure drag coefficient C_v of a boattail, like the one in fig. 19, is deductively related to the base drag and the length to height ratio λ, calculated from the boattail's length l_v and fore d_{v_f} and aft d_{v_a} diameters. Data from ref. [7] (Barrowman, 1967) suggests that:

$$\lambda = \frac{l_v}{d_{v_f} - d_{v_a}} \quad (32)$$

$$C_v = \frac{A_b}{A_r} C_b \cdot \begin{cases} 1 & \text{if } \lambda < 1 \\ \frac{3 - \lambda}{2} & \text{if } 1 \le \lambda \le 3 \\ 0 & \text{if } 3 < \lambda \end{cases} \quad (33)$$

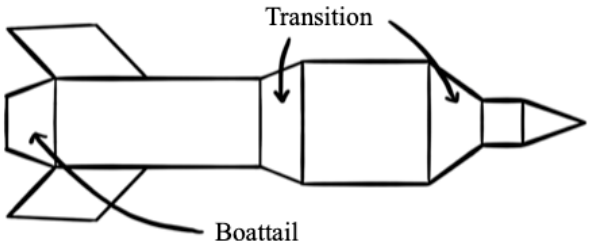

Fig. 19 - Rocket with transitional components and a boattail.

Body pressure drag coefficient

The total pressure drag coefficient may then be calculated by adding the tip C_t, transition C_j and boattail C_v pressure drag coefficients.

$$C_{A_p} = C_t + C_j + C_v \tag{34}$$

Fin pressure drag coefficient

The fin pressure drag is highly dependent on its profile geometry, typically classified as rectangular ⊓, tapered ∧, or rounded ∩.

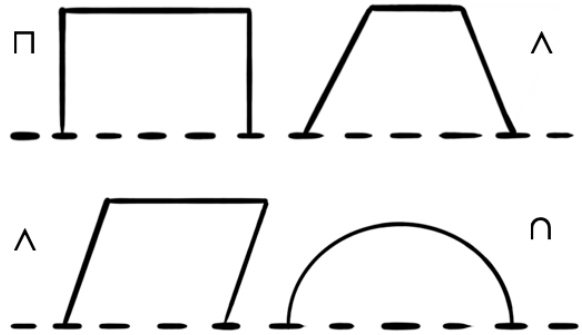

Fig. 20 - Typical rectangular, tapered and rounded fin geometries.

The coefficient C_n may be obtained by adding the pressure drag of the leading C_{n_l} and trailing C_{n_t} edges, scaled up to the fin frontal area A_{n_f} common to the reference area, where pressure drag is encountered.

$$C_{A_n} = \frac{A_{n_f}}{A_r}(C_{n_l} + C_{n_t}) \tag{35}$$

From experimentation, empirical formulae may be derived to calculate the pressure drag of different geometry leading and trailing edges, as approached in ref. [7] (Barrowman, 1967).

$$C_{n\cap_l} = \begin{cases} (1-M^2)^{-0.417} - 1 & \text{if } M < 0.9 \\ 1 - 1.785(M - 0.9) & \text{if } 0.9 \le M \le 1 \\ 1.214 - \frac{0.502}{M^2} + \frac{0.1095}{M^4} & \text{if } 1 < M \end{cases} \tag{36}$$

$$C_{n_l} = \begin{cases} C_s & \text{for } \sqcap \\ C_{n\cap_l} \cdot cos^2\mu & \text{for } \wedge \\ C_{n\cap_l} & \text{for } \cap \end{cases} \tag{37}$$

$$C_{n_t} = \begin{cases} C_b & \text{for } \sqcap \\ 0 & \text{for } \wedge \\ \frac{C_b}{2} & \text{for } \cap \end{cases} \tag{38}$$

Where μ is the average leading edge angle, identified in fig. 24. The pressure drag is higher for lower values of μ, as they translate into sharper transitions, and therefore, increased flow separation. The pressure drag is greater at transonic velocities, where shock waves form due to the compression of the fluid and create chaotic flow which does not stabilize until supersonic flow is established.

Normal Coefficient

The normal force N is the component of the aerodynamic forces perpendicular to the rocket's centreline and may be calculated using eq. (7) by substituting the lift coefficient for the normal coefficient C_N, defined as the combination of the lift exerted by symmetrical body components $C_{N_a} + C_{N_l}$ and the fins C_{N_n}.

$$C_N = C_{N_a} + C_{N_l} + C_{N_n} \tag{39}$$

Symmetrical body components normal coefficient

The normal force N_x at a point x for an axially symmetric body may be computed with the cross sectional area A_x of the body at position x and the local downwash $w_x = v_f \, sin(\alpha)$. According to ref. [7] (Barrowman, 1967) based on the potential flow theory and with reliable results accurate to about 6%:

$$\begin{aligned} N_x &= p v_f \frac{\partial}{\partial x}[A_x w_x] \\ &= p v_f^2 \, sin(\alpha) \frac{\mathrm{d}A_x}{\mathrm{d}x} \end{aligned} \tag{40}$$

Substituting values into eq. (7) and integrating the cross sectional area derivative over the component length the normal coefficient C_{N_a} can be obtained:

$$\begin{aligned} C_{N_{a_x}} &= \frac{N_x}{\frac{1}{2} p v_f^2 A_r} \\ &= p v_f^2 \, sin(\alpha) \frac{\mathrm{d}A_x}{\mathrm{d}x} \end{aligned} \tag{41}$$

$$\begin{aligned} C_{N_a} &= \frac{2 sin\,(a)}{A_{ref}} \int_0^l \frac{\mathrm{d}A(x))}{\mathrm{d}x} dx \\ &= \frac{2 \, sin(\alpha)}{A_{ref}} [A_{x=l} - A_{x=0}] \end{aligned} \tag{42}$$

These equations demonstrate that the normal force coefficient will depend on the difference of the of cross sectional area at the aft and fore ends. However, the lift due to cylindrical bodies with constant cross sectional areas, even at low angles of attack, noted by experiments in ref. [13] (Vallini, 2015) counterclaims this approach. Thus a correction term is added accounting for body lift ref. [6] (Jorgensen, 1973):

$$C_{N_l} = \eta C_d \frac{A_p}{A_r} sin^2(\alpha) \tag{43}$$

Where C_d is the crossflow drag coefficient for a section of an undefined length cylinder placed normal to an air stream, and η is the crossflow drag proportionality factor, denoted by the ratio of the crossflow drag coefficient for a finite length cylinder to that of a infinite length cylinder. The variation of C_d is well documented by fig. 21:

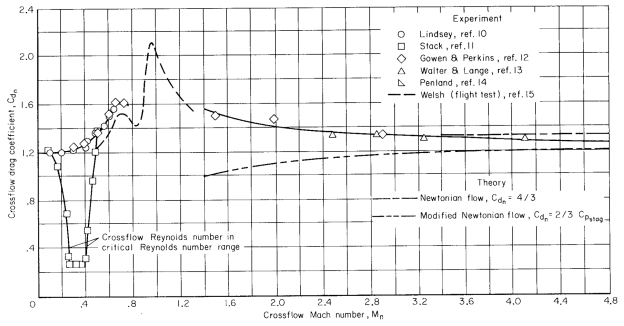

Fig. 21 - Variation of the crossflow drag coefficient with Mach number.

However, at low subsonic speeds below $M \approx 0.5$, the Reynolds number may have a significant effect on the coefficient, for which additional experimental data was also recorded and is shown in fig. 22:

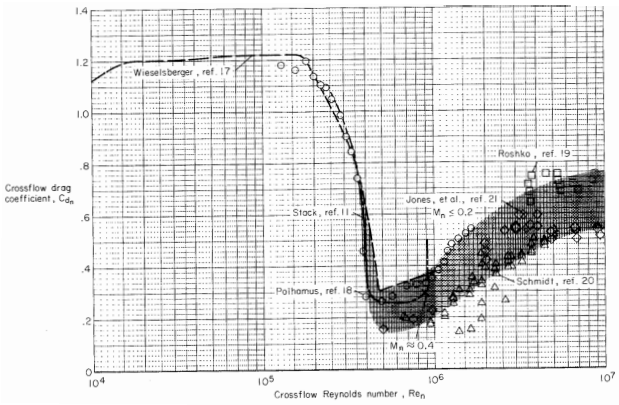

Fig. 22 - Variation of the crossflow drag coefficient with Reynolds number.

The following regression line can be derived from the experimental data, such that:

$$C_d' = \begin{cases} 1.2 & \text{if } Re < 10^5.25 \\ -2.575 \cdot 10^{-6} \\ \cdot (Re - 10^{5.25}) + 1.2 & \text{if } 10^5.25 \leq Re \leq 10^5.75 \\ \frac{1}{6} ln(Re) - 2 & \text{if } 10^{5.75} < Re \end{cases} \tag{44}$$

$$C_d = \begin{cases} C_d' & \text{if } M < 0.5 \\ 2^{0.9M} + 0.2 & \text{if } 0.5 \leq M < 1 \\ -0.1\, ln(0.1(M-1)) + 1.1 & \text{if } 1 \leq M \leq 5 \\ 1.2 & \text{if } M < 5 \end{cases} \quad (45)$$

The experimental variation of η with the length l to diameter d ratio of a circular cylinder is presented by the following figure:

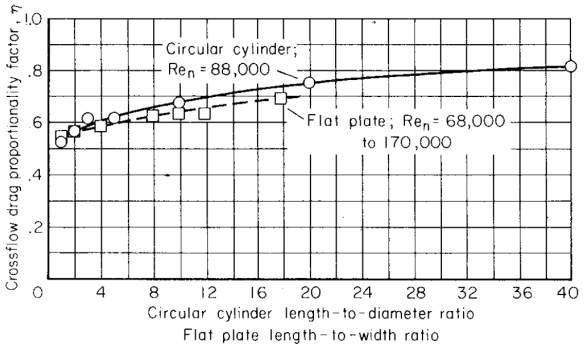

Fig. 23 - Variation of the crossflow drag proportionality with length to diameter ratio.

From which the following equation may be obtained to estimate its value:

$$\eta = \frac{1}{12}\, ln\left(\frac{l}{d} + 1\right) + 0.5 \quad (46)$$

Fins normal coefficient

The normal force coefficient derivative C_N for one fin at subsonic velocities may be calculated using a semi-empirical approach introduced in ref. [7] (Barrowman, 1967) and the thin airfoil theory of potential flow [ref. 14] (Gaunaa, 2006).

$$C_N = \frac{\frac{2\pi}{\beta} F_D \frac{A_{fin}}{A_{ref}} cos(\vartheta)}{2 + F_D \sqrt{1 + \frac{4}{F_D^2}}} \alpha \quad (47)$$

Where ϑ is the mid chord sweep angle, as seen in fig. 24, and F_D is the corrective value accounting for the sweep of the fin given by:

$$F_D = \frac{\frac{2s^2}{A_{ref}}}{\frac{1}{2\pi} C_{N_{\alpha 0}} cos\,\vartheta} \quad (48)$$

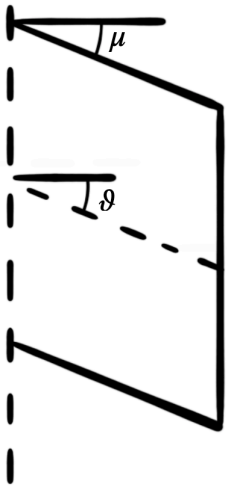

Fig. 24 - Average leading edge angle and mid chord sweep angle of a fin.

$$C_n = \frac{2\pi \frac{s^2}{A_{ref}} \alpha}{1 + \sqrt{1 + \left(\frac{\beta s^2}{A_{fin} cos(\psi)}\right)^2}} \quad (49)$$

By substitution, the coefficient becomes:
Otherwise, the normal force coefficient of a fin at supersonic velocities can be calculated through a third-order series expansion used in ref. [15] (Barrowman, 1966) that defines the local pressure coefficient C_P.

$$C_P = K_1\alpha + K_2\alpha^2 + K_3\alpha^3 \quad (50)$$

$$K_1 = \frac{2}{\beta} \quad (51)$$

$$K_2 = \frac{(\gamma+1)M^4 - 4\beta^2}{4\beta^4} \quad (52)$$

$$K_3 = \frac{(\gamma+1)M^8}{6\beta^7} + \frac{(2\gamma^2 - 7\gamma - 5)M^6}{6\beta^7} + 10(\gamma+1)M^4 + 8 \quad (53)$$

The lift force N of a fin is given by:

$$N = C_P \cdot \frac{1}{2} p_f v_f^2 A_f \tag{54}$$

Which may the be substituted into eq. (7) to obtain the normal force coefficient $C_n$of a fin:

$$\begin{aligned} C_n &= \frac{N}{\frac{1}{2} p_f v_f^2 A_r} \\ &= \frac{A_f}{A_r} C_P \end{aligned} \tag{55}$$

Altogether, the normal force coefficient C_{N_n} for a number of fins y is given by the sum of their individual force coefficient at an angle of attack Λ, depicted by fig. 24.

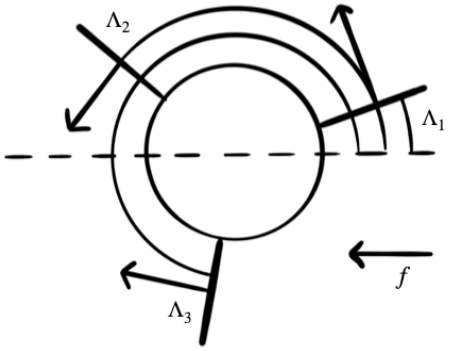

Fig. 24 - Normal force of fins and their individual angle of attack.

$$C_{N_n} = C_n \left(\sum_{i=1}^{y} sin^2(\Lambda_i) \right) \cdot \Xi \tag{56}$$

Where Ξ is a correction term for interference between fins, suggested by ref. [16] (United States, 1968):

$$\Xi = \begin{cases} 1 & \text{for } 1 \leq y \leq 4 \\ 0.984 & \text{for } y = 5 \\ 0.913 & \text{for } y = 6 \\ 0.854 & \text{for } y = 7 \\ 0.810 & \text{for } y = 8 \\ 0.750 & \text{for } 8 \leq y \end{cases} \tag{57}$$

Simulation

The designed flight simulation software was built on Swift. The code developed may be found in Appendix C, and its architecture is further explained in Appendix B. Comparisons between simulated and experimental flight data were made to validate the reliability of the software, *Apex Dream*. These include a simple model rocket and an advanced or thrust vectored controlled model rocket. The designs of the rockets from the experiments were introduced like a controlled variable into the flight simulation software, including the alternative software OpenRocket and RockSim for further comparison. Flight profiles are outlined by the following figures:

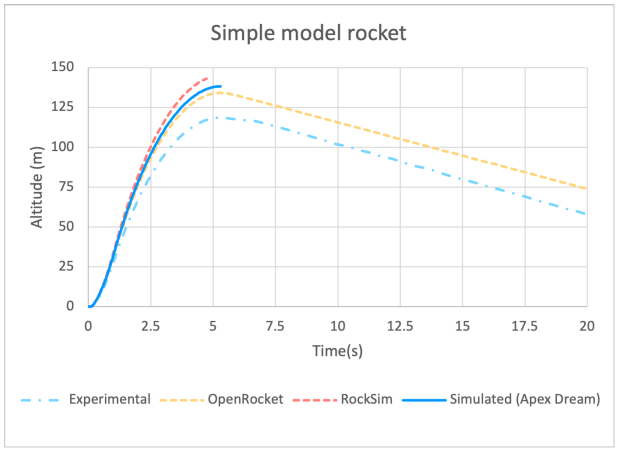

Fig. 25 - Experimental and simulated flight comparison of a simple model rocket.

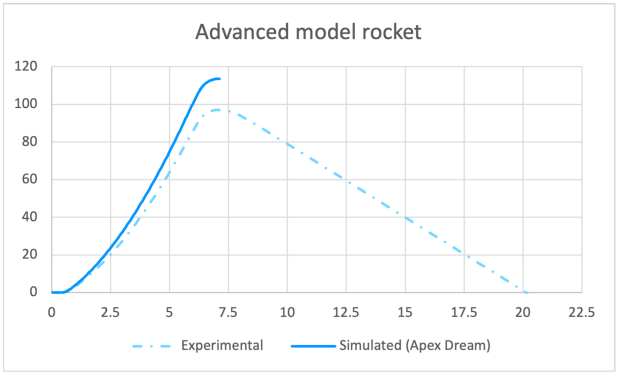

Fig. 26 - Experimental and simulated flight comparison of an advanced model rocket.

The descent phase of a rocket flight is not developed, and thus, the simulated data is interrupted at apogee, the highest altitude. Moreover, fig. 26 only shows the simulated data by *Apex Dream* as it is the only simulation software compatible with the thrust vector control system featured by the experimental advanced model rocket.

All simulations appear to be notably optimistic, predicting slightly higher apogees, from the experimental flights. Errors are presented by the following tables:

	Apogee (m)	Error (%)
Experimental	118.53	- %
OpenRocket	134.08	13%
RockSim	143.02	21%
Simulated (Apex Dream)	138.10	17%

Tab. 2 - Errors for comparison between experimental and simulated data of a simple model rocket.

	Apogee (m)	Error (%)
Experimental	97.12	- %
Simulated (Apex Dream)	113.63	17%

Tab. 3 - Errors for comparison between experimental and simulated data of an advanced model rocket.

Conclusion

Flight simulations are key to an optimized rocket design and risk minimization; however, there is no available flight simulation software compatible with commonly featured systems in advanced model rocketry. Therefore, the forces acting on a rocket during its flight were deeply studied to develop an advance model rocket flight simulation software.

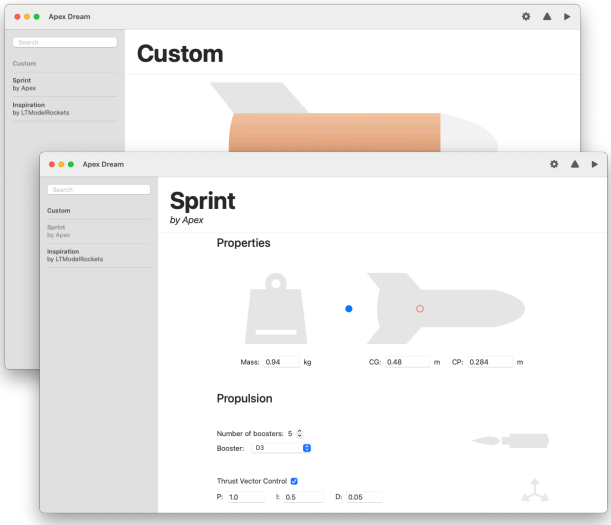

Fig. 27 - *Apex Dream* design interface.

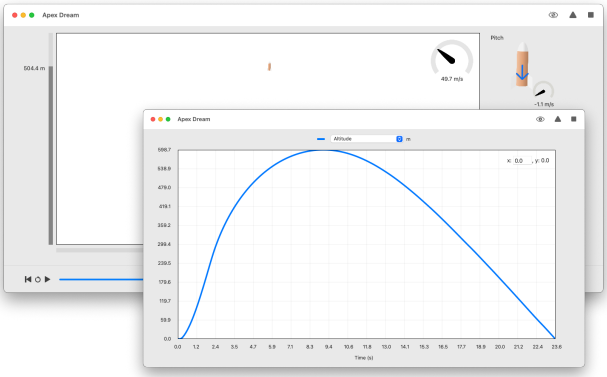

Fig. 28 - *Apex Dream* simulation interface.

This work not only provides detailed estimates for aerodynamic properties, for which there is no defined mathematical models, but also introduces a powerful tool that may be used by any advanced model rocketeer. Results are a bit optimistic, probably due to:

Assumptions were made in the calculation of aerodynamic properties. Estimations for the coefficients of a component with a determined shape were applied to a more varied range of shapes based on their similarity. While these assumptions had a solid justification and demonstrated not to be very off, given the error of the simulation, they may be corrected through more extense calculation that adjust to a greater variation of components and their forms.

Assumptions were made in the calculation of aerodynamic properties. Estimations for the coefficients of a component with a determined shape were applied to a more varied range of shapes based on their similarity. While these assumptions had a solid justification and demonstrated not to be very off, given the error of the simulation, they may be corrected through more extense calculation that adjust to a greater variation of components and their forms.
Wind **disturbances** were not taken into account, which may have a considerable effect on real rocket flight and therefore the experimental data as well, probably contributing to the error. Wind disturbances may be coded and accounted by flow calculations. Perhaps, the characteristics of such disturbances could be manually introduced by the user.
Empirical formulae was derived from experimental data under **limited conditions**, including different ranges of Mach and Reynolds numbers. The equations extrapolated observed relations to greater ranges which may have led to little inaccuracies in calculation, again, contributing to the small error in the simulation.
Sources were carefully selected based on their reliability. Note that the bibliography only makes reference to this selection of sources. The models that best predicted the aerodynamic properties in comparison to wind tunnel data and other simulation software, line OpenRocket in ref. [17] (Niskanen, 2013), were used. References include a primary and secondary sources, that present experimental data and empirical formulae. Many come from prestigious organizations or individuals, such as NASA.
Introduced rocket **characteristics** may have been slightly off due to inaccurate measurement, which would have led to miscalculations of the rocket's aerodynamic properties and therefore greater errors. This could by retrieving more accurate data, perhaps obtained with more precise tools.
Aerodynamic phenomenon with almost negligible effects on ideal conditions, such as the roll of a rocket exerted by the aerodynamic forces, were not approximated nor taken into account in any way. Future version of the simulation software will include these and hopefully give more accurate results, for which additional research will have to be done.

Tab. 4 - Evaluation and discussion.

Overall, the flight simulation software is reliable enough with errors below 20% and similar in accuracy to other popular flight simulation software, which also proves the reliability of the referenced sources. Thus, may be used to optimise the rocket design. Approximations for aerodynamic forces may be further develop by accounting for roll, additional sources of drag which were not considered in this work due to their small contribution to the resultant drag and lift of a rocket, or even by performing more wind tunnels experimentation to retrieve more accurate formulae. Despite the effectiveness of *Apex Dream*, the software may still be utilized to tune the systems of thrust vectored controlled model rockets, as these are not dependant on the flight profile, but rather the variation of aerodynamic properties.

In conclusion, this investigation answers the initial research question to a great extent, yet, more investigation could and will be made to improve the reliability of the simulation software and include more features. This work brought me a step closer to the flight of the advanced model rocket I am currently developing and only dreamt of to this date. Hopefully, this work contributes to the model rocketry community by redefining the boundaries of available flight simulation software and expanding possibilities, allowing to design reliably advanced model rockets featuring thrust vector control, tune their systems, and estimate their aerodynamic properties and flightpaths.

Appendix A

Center of pressure

The document cited in ref. [18] (Barrowman & Barrowman, 1966) presents algebraic methods to determine the location of the CP:

$$X_{CP} = \frac{C_{P_t}X_t + C_{P_j}X_j + C_{P_n}X_n}{C_P} \qquad (A.1)$$

The coefficient of the denominator and each adding term, corresponding to the rocket tip, any transition and the fins respectively, can be calculated from:

$$C_P = C_{P_t} + C_{P_j} + C_{P_n} \qquad (A.2)$$

$$C_{P_t} = 2 \qquad (A.3)$$

$$C_{P_t} = \begin{cases} \frac{2}{3}l_t & \text{for } \wedge \\ \frac{7}{15}l_t & \text{for } \cap \end{cases} \qquad (A.4)$$

$$C_{P_t} = 2\left(\left(\frac{d_{j_a}}{d_{t_b}}\right)^2 - \left(\frac{d_{j_f}}{d_{t_b}}\right)^2\right) \qquad (A.5)$$

$$X_j = X_p + \frac{l_j}{3}\left(1 + \frac{1 - \frac{d_{j_f}}{d_{j_a}}}{1 - \left(\frac{d_{j_f}}{d_{j_a}}\right)^2}\right) \qquad (A.6)$$

$$C_{P_n} = \left(1 + \frac{r_b}{h_n + r_b}\right)\left(\frac{4y\left(\frac{h_n}{d_b}\right)^2}{1 + \sqrt{1 + \left(\frac{2l_{n_m}}{l_{n_r} + l_{n_t}}\right)^2}}\right) \qquad (A.7)$$

$$C_{P_n} = \left(1 + \frac{r_b}{h_n + r_b}\right)\left(\frac{4y\left(\frac{h_n}{d_b}\right)^2}{1 + \sqrt{1 + \left(\frac{2l_{n_m}}{l_{n_r} + l_{n_t}}\right)^2}}\right) \qquad (A.8)$$

$$X_n = X_b + \frac{X_r(l_{n_r} + 2l_{n_t})}{3(l_{n_r} + l_{n_t})} + \frac{1}{6}\left((l_{n_r} + l_{n_t}) - \frac{l_{n_r}l_{n_t}}{l_{n_r} + l_{n_t}}\right) \qquad (A.9)$$

Where the variables:

l_t = tip length
d_{t_b} = tip base diameter
d_{j_f} = transition fore diameter
d_{j_a} = transition aft diameter
l_j = transition length
X_p = distance from tip to transition fore
l_{n_r} = fin root chord length
l_{n_t} = fin tip chord length
l_{n_m} = fin mid chord length
h_n = fin height
r_b = base radius
X_r = distance from root to tip leading edge
X_b = distance from tip to fin root
y = number of fins

Appendix B

Simulation architecture

The simulation was built on Swift. Simple individual characteristics of a rocket, limited to the range of variations in from, are introduced by the user. Additional characteristics, like the chord length of a fin or the joint angle of a rocket tip, are calculated automatically by the simulation software from the simpler inputted data. Designing a rocket in *Apex Dream* is made easier because of this, however, it may also lead to some inaccuracies. Values for the conditions are also introduced, including the angle of the wind and its velocity. Note however that the simulation software does not account for disturbances in wind yet. The user also sets up the simulation by defining the running frequency, used to calculate the period between each set of calculations for an instance in the flight.

A loop runs from liftoff to recovery; however, aerodynamic properties for the descent phase of the rocket flight weren't appropriately coded as of now, which makes the simulation only valuable up to apogee, the highest point of a rocket flightpath. For every instance of the loop the simulation first codes all flow properties, including the angle of attack, derived from the values from the previous instance. Then, all forces are calculated, including weight, thrust and aerodynamic forces. Model rocket often fly at low altitudes were the difference in weight is almost negligible, but it is still accurately calculated based on the rocket's altitude using basic gravitational equations. The thrust is computed by linearly interpolating the different points of the data base of a determined rocket motor. The direction of the thrust is also accounted for depending on the rocket's orientation and the thrust vectoring, which differs this simulation software from others. Then, all aerodynamic properties are calculated for every component of the rocket with the equations presented by this work, later used with the drag and lift equations.

All this forces are stored as two-dimensional vectors, although the simulation software is soon expected to expand to the three dimensions. This however, may not compromise the effectiveness of the simulations. The linear and angular accelerations of the rocket are then calculated with the force vectors using force and torque equations. With previous and actual acceleration new velocities and positions can be computed. If the rocket has not landed the loop runs again.

Appendix C

Simulation code

```
import Foundation
import simd

func degreesToRadians(_ degrees: Double) -> Double {
    return degrees * .pi / 180
}

func radiansToDegrees(_ radians: Double) -> Double {
    return radians * 180 / .pi
}

struct Motor {
    var id = UUID()
    var name = String()
    var manufacturer = String()
    var data = [Double:Double]()
    var totalMass = Double()
    var propellantMass = Double()
}

enum FinType {
    case slanted, tapered, rectangular, rounded
}

struct Rocket {
    var id = UUID()
    var name = "Custom Rocket"
    var manufacturer: String = "User"

    var mass = 0.0
    var massCentre = 0.0
    var pressureCentre = 0.0

    var length = 0.0
    var diameter = 0.0

    var roughnessHeight = 0.00

    var jointAngle = 0.0

    var tipChord = 0.0
    var rootChord = 0.0
    var height = 0.0
    var thickness = 0.0
    var finsNumber = 0.0
    var tipSweepAngle = degreesToRadians(0.0)
    var midSweepAngle = degreesToRadians(0.0)
    var finType = FinType.slanted

    var volume: Double {
        return .pi * pow((diameter / 2), 2) * length
    }
    var coneRatio: Double {
        return atan(((diameter / 2) / length) * 180 / Double.pi)
```

```
    }
    var finArea: Double {
        return (tipChord + rootChord) / 2 * height
    }
    var averageChord: Double {
        return (tipChord + rootChord) / 2
    }
    var wettedBodyArea: Double {
        return .pi * length * diameter
    }
    var wettedFinsArea: Double {
        return finArea * 2 * Double(finsNumber)
    }
    var frontFinArea: Double {
        return thickness * length * Double(finsNumber)
    }
    var baseArea: Double {
        return .pi * pow(diameter / 2, 2)
    }
    var referenceArea: Double {
        return .pi * pow(diameter / 2, 2)
    }
    var planformArea: Double {
        return diameter * length
    }
    var planformCentroid: Double {
        return length / 2 + 0.05 * length
    }
    var rotationalInertia: Double {
        return (mass * pow(diameter / 2, 2)) / 4 + (mass * pow(length / 2, 2)) / 12
    }

    var motor: Motor = Motor(id: UUID(),
                             name: "Custom Motor",
                             manufacturer: "User",
                             data: [0.000: 0.000],
                             totalMass: 0.0163,
                             propellantMass: 0.003)
}

struct Conditions {
    var windSpeed = 0.0
    var windOrientation = simd_double2(x: 0, y: 0)

    var wind: simd_double2 {
        return windSpeed * simd_normalize(windOrientation)
    }
}

struct Settings {
    var frequency = 50.0

    var period: Double {
        return 1 / self.frequency
    }
}


struct FlightData {
    var id = UUID()
    var title = "Flight Simulation"
```

```
    var date = Date()

    var time = [Double]()

    var rocket: Rocket
    var conditions: Conditions

    var position = [simd_double2]()
    var orientation = [simd_double2]()

    var velocity = [simd_double2]()
    var angularVelocity = [Double]()
    var acceleration = [simd_double2]()
    var angularAcceleration = [Double]()

    var axialDrag = [simd_double2]()
    var axialDragCoefficient = [Double]()

    var frictionDragCoefficient = [Double]()
    var baseDragCoefficient = [Double]()
    var pressureDragCoefficient = [Double]()

    var normalForce = [simd_double2]()
    var normalForceCoefficient = [Double]()

    var pitchMomentCoefficient = [Double]()

    var thrust = [simd_double2]()
    var weight = [simd_double2]()

    var attackAngle = [Double]()
    var reynoldsNumber = [Double]()
    var pressureCentre = [Double]()
}

func flightSim(rocket: Rocket, conditions: Conditions, settings: Settings) -> FlightData {
    var data = FlightData(rocket: rocket, conditions: conditions)

    var running = true
    var loop = 0

    while running {
        let time = Double(loop) * settings.period
        data.time.append(time)

        guard loop > 0 else {
            data.position.append(simd_double2())
            data.orientation.append(simd_double2(x: 0, y: 1))
            data.velocity.append(simd_double2())
            data.angularVelocity.append(Double())
            data.acceleration.append(simd_double2())
            data.angularAcceleration.append(Double())
            data.axialDrag.append(simd_double2())
            data.axialDragCoefficient.append(Double())
            data.frictionDragCoefficient.append(Double())
            data.baseDragCoefficient.append(Double())
            data.baseDragCoefficient.append(Double())
            data.pressureDragCoefficient.append(Double())
            data.normalForce.append(simd_double2())
            data.normalForceCoefficient.append(Double())
            data.pitchMomentCoefficient.append(Double())
```

```
            data.thrust.append(simd_double2())
            data.weight.append(simd_double2())
            data.attackAngle.append(Double())
            data.reynoldsNumber.append(Double())
            data.pressureCentre.append(Double())

            loop += 1
            continue
        }

        let airDensity = 1.225
        let airViscosity = 1.81 * pow(10, Double(-5))
        let soundSpeed = 340.0

        let flow = -data.velocity[loop - 1] + data.conditions.wind

        let machNumber = simd_length(data.velocity[loop - 1]) / soundSpeed

        let reynoldsNumber = airDensity * simd_length(flow) * rocket.length / airViscosity
        let criticalReynoldsNumber = 51 * pow(rocket.roughnessHeight / rocket.length, -1.039)
        data.reynoldsNumber.append(reynoldsNumber)

        //check
        var attackAngle = acos(simd_dot(data.orientation[loop - 1], flow) / simd_length(data.orientation[loop - 1]
* simd_length(flow)))
        data.attackAngle.append(attackAngle)

        var correctedAttackAngle: Double {
            if abs(attackAngle) < 90{
                return abs(attackAngle)
            } else {
                return 180 - abs(attackAngle)
            }
        }

        let keyTimes = Array(rocket.motor.data.keys).sorted(by: <)

        let previousTime = keyTimes.lastIndex(where: {$0 <= time})

        var thrust: simd_double2 {
            if previousTime != (keyTimes.count - 1) {
                let lastKey = simd_double2(x: keyTimes[previousTime ?? 0],
                                           y: rocket.motor.data[keyTimes[previousTime ?? 0]] ?? 0.0)
                let nextKey = simd_double2(x: keyTimes[(previousTime ?? 0) + 1],
                                           y: rocket.motor.data[keyTimes[(previousTime ?? 0) + 1]] ?? 0.0)

                let slope = (nextKey.y - lastKey.y) / (nextKey.x - lastKey.x)

                return (slope * (time - lastKey.x) + lastKey.y) * simd_normalize(data.orientation[loop - 1])
            } else { return simd_double2() }
        }

        data.thrust.append(thrust)

        var motorMass: Double {
            if time > keyTimes[keyTimes.count - 1] {
                return rocket.motor.totalMass - rocket.motor.propellantMass
            } else {
                return ((rocket.motor.totalMass - rocket.motor.propellantMass) - rocket.motor.totalMass) /
(keyTimes[0] - keyTimes[keyTimes.count - 1]) * time + rocket.motor.totalMass
            }
```

```
        }

        let totalMass = rocket.mass + motorMass

        var weight: simd_double2 {
            if data.position[loop - 1].y > 0 {

                let earthMass = 5.972 * pow(10, Double(24))
                let earthRadius = 6.371 * pow(10, Double(6))
                let G = 6.67 * pow(10, Double(-11))

                let g = G * earthMass / pow(earthRadius + Double(data.position[loop - 1].y), 2)

                return (totalMass * Double(g)) * simd_normalize(simd_double2(x: 0, y: -1))
            } else { return simd_double2() }
        }

        data.weight.append(weight)

        var frictionCoefficient: Double {
            var approximation: Double {
                if reynoldsNumber < pow(10, 4) {
                    return 1.48 * pow(10, -2)
                } else if reynoldsNumber > pow(10, 4) && reynoldsNumber < criticalReynoldsNumber {
                    return 1 / pow(1.5 * log(reynoldsNumber) - 5.6, 2)
                } else {
                    return 0.032 * pow(rocket.roughnessHeight / rocket.length, 0.2)
                }
            }
            if simd_length(flow) < soundSpeed {
                return approximation * (1 - 0.1 * pow(machNumber, 2))
            } else {
                return approximation / pow(1 + 0.15 * pow(machNumber, 2), 0.58)
            }
        }

        var frictionDragCoefficient: Double {
            if rocket.finsNumber > 0 {
                return (frictionCoefficient * (((1 + (1 / (2 * rocket.length / rocket.diameter))) *
rocket.wettedBodyArea + (1 + (2 * rocket.thickness) / rocket.averageChord) * rocket.wettedFinsArea) /
rocket.baseArea)) * pow(abs(cos(attackAngle)), 2)
            } else {
                return (frictionCoefficient * (((1 + (1 / (2 * rocket.length / rocket.diameter))) *
rocket.wettedBodyArea) / rocket.referenceArea)) * pow(abs(cos(attackAngle)), 2)
            }
        }

        data.frictionDragCoefficient.append(frictionCoefficient)

        var baseDragCoefficient: Double {
            if machNumber < 1 {
                return ((rocket.baseArea / rocket.referenceArea) * (0.12 + 0.13 * pow(machNumber, 2))) *
pow(abs(cos(attackAngle)), 2)
            } else {
                return ((rocket.baseArea / rocket.referenceArea) * 0.25 / machNumber) * pow(abs(cos(attackAngle)),
2)
            }
        }

        data.baseDragCoefficient.append(baseDragCoefficient)
```

```
        var leadingFinPressure: Double {
            switch rocket.finType {
            case .rectangular:
                var stagnationPressure: Double {
                    if machNumber < 1 {
                        return 1 + (pow(machNumber, 2) / 4) + (pow(machNumber, 4) / 40)
                    } else {
                        return 1.84 - (0.76 / pow(machNumber, 2)) + (0.166 / pow(machNumber, 4)) + (0.035 /
pow(machNumber, 6))
                    }
                }
                return 0.85 * stagnationPressure
            default:
                var leadingPerpendicularFinPressure: Double {
                    if machNumber < 0.9 {
                        return pow(1 - pow(machNumber, 2), -0.417) - 1
                    } else if machNumber > 0.9 && machNumber < 1 {
                        return 1 - 1.785 * (machNumber - 0.9)
                    } else {
                        return 1.214 - (0.502 / pow(machNumber, 2)) + (0.1095 / pow(machNumber, 4))
                    }
                }
                switch rocket.finType {
                case .slanted:
                    return leadingPerpendicularFinPressure * pow(cos(rocket.tipSweepAngle * Double.pi / 180), 2)
                default:
                    return leadingPerpendicularFinPressure
                }
            }
        }

        //check
        var trailingFinPressure: Double {
            switch rocket.finType {
            case .rectangular:
                return baseDragCoefficient
            case  .rounded:
                return baseDragCoefficient / 2
            default:
                return 0
            }
        }

        let finPressureCoefficient: Double = (rocket.frontFinArea / rocket.referenceArea) * (leadingFinPressure +
trailingFinPressure) * pow(abs(cos(attackAngle)), 2)

        //check
        var conePressureCoefficient: Double {
            if machNumber <= 1 {
                let startPoint = 0.8 * pow(sin(rocket.jointAngle * Double.pi / 180), 2)
                let startSlope = 0.0
                let finishPoint = sin(rocket.coneRatio * Double.pi / 180)
                let finishSlope = (4 / (1.4 + 1)) * (1 - 0.5 * finishPoint)

                return ((finishSlope - startSlope) / 2 * pow(machNumber, 2) + startPoint) *
pow(abs(cos(attackAngle)), 2)
            } else if machNumber > 1 && machNumber < 1.3 {
                return 1 * pow(abs(cos(attackAngle)), 2)
            } else {
                return (2.1 * pow(sin(rocket.coneRatio * Double.pi / 180), 2) + 0.5 * (sin(rocket.coneRatio *
Double.pi / 180) / sqrt(pow(machNumber, 2) - 1))) * pow(abs(cos(attackAngle)), 2)
```

```
            }
        }

        //boattail

        let pressureDragCoefficient = conePressureCoefficient + finPressureCoefficient
        data.pressureDragCoefficient.append(pressureDragCoefficient)

        let axialDragCoefficient = pressureDragCoefficient + baseDragCoefficient + frictionDragCoefficient
        data.axialDragCoefficient.append(axialDragCoefficient)

        let axialDrag = (axialDragCoefficient * airDensity * pow(simd_length(data.velocity[loop - 1]), 2) *
rocket.referenceArea / 2) * simd_normalize(-data.orientation[loop - 1])
        data.axialDrag.append(axialDrag)

        let crossflowDragProportionalityFactor = 0.7
        let crossflowDragCoefficient = 1.2

        var bodyNormalForceCoefficient: Double {
            if time > 0.25 {
                return 2 * (rocket.baseArea / rocket.referenceArea) * sin(attackAngle) +
crossflowDragProportionalityFactor * crossflowDragCoefficient * 1.1 * pow(sin(attackAngle), 2)
            } else {
                return 0
            }
        }

        let specificHeatRatio = 1.4

        var singleFinNormalForceCoefficient: Double {
            if machNumber <= 0 {
                return 0
            } else if machNumber > 0 && machNumber < 1 {
                return 2 * .pi * (pow(rocket.height, 2) / rocket.referenceArea) / (1 + sqrt(1 + pow((sqrt(1 -
pow(machNumber, 2)) * pow(rocket.height, 2)) / (rocket.finArea * cos(rocket.midSweepAngle)), 2)))
            } else {
                let k1 = 2 / sqrt(abs(pow(machNumber, 2) - 1))
                let k2 = ((specificHeatRatio + 1) * pow(machNumber, 4) - 4 * pow(sqrt(abs(pow(machNumber, 2) -
1)), 2)) / (4 * pow(sqrt(abs(pow(machNumber, 2) - 1)), 4))
                let k3 = ((specificHeatRatio + 1) * pow(machNumber, 8) + (2 * pow(specificHeatRatio, 2) - 7 *
specificHeatRatio - 5) * pow(machNumber, 6) + 10 * (specificHeatRatio + 1) * pow(machNumber, 4) + 8) / (6 *
pow(sqrt(abs(pow(machNumber, 2) - 1)), 7))

                return (rocket.finArea / rocket.referenceArea) * (k1 * degreesToRadians(attackAngle) + k2 *
degreesToRadians(attackAngle) + k3 * pow(degreesToRadians(attackAngle), 2))
            }
        }

        var finNormalForceCoefficient: Double {
            var interference: Double {
                switch rocket.finsNumber {
                case 0:
                    return 0
                case 1...4:
                    return 1.0
                case 5:
                    return 0.948
                case 6:
                    return 0.913
                case 7:
                    return 0.854
```

```
                case 8:
                    return 0.810
                default:
                    return 0.750
                }
            }
            let contribution = Double(rocket.finsNumber) / 2

            return (singleFinNormalForceCoefficient * contribution * interference) * (1 + (360 /
Double(rocket.finsNumber)) / (rocket.height + 360 / Double(rocket.finsNumber)))

        }


        let normalForceCoefficient = bodyNormalForceCoefficient + finNormalForceCoefficient
        data.normalForceCoefficient.append(normalForceCoefficient)

        var normal: simd_double2 {
            let perpendicular = simd_normalize(simd_double2(x: data.orientation[loop - 1].y, y:
-data.orientation[loop - 1].x))

            if acos(simd_dot(data.orientation[loop - 1], perpendicular) / simd_length(data.orientation[loop - 1] *
simd_length(perpendicular))) < 90 {
                return simd_normalize(simd_double2(x: data.orientation[loop - 1].y, y: -data.orientation[loop -
1].x))
            } else {
                return simd_normalize(simd_double2(x: -data.orientation[loop - 1].y, y: data.orientation[loop -
1].x))
            }
        }

        let normalForce = (normalForceCoefficient * airDensity * pow(simd_length(data.velocity[loop - 1]), 2) *
rocket.referenceArea / 2) * normal
        data.axialDrag.append(axialDrag)

        var pitchMomentCoefficient: Double {
            if abs(attackAngle) < 90 {
                return ((rocket.volume - rocket.baseArea * (rocket.length - rocket.massCentre)) /
rocket.referenceArea * rocket.diameter) * sin(2 * correctedAttackAngle) * cos(correctedAttackAngle / 2) +
crossflowDragProportionalityFactor * crossflowDragCoefficient * (rocket.planformArea / rocket.referenceArea) *
((rocket.massCentre - rocket.planformCentroid) / rocket.diameter) * pow(sin(correctedAttackAngle), 2)
            } else {
                return -((rocket.volume - rocket.baseArea * rocket.massCentre) / rocket.referenceArea *
rocket.diameter) * sin(2 * correctedAttackAngle) * cos(correctedAttackAngle / 2) +
crossflowDragProportionalityFactor * crossflowDragCoefficient * (rocket.planformArea / rocket.referenceArea) *
((rocket.massCentre - rocket.planformCentroid) / rocket.diameter) * pow(sin(correctedAttackAngle), 2)
            }
        }

        data.pitchMomentCoefficient.append(pitchMomentCoefficient)

        var pressureCentre: Double {
            if normalForceCoefficient != 0 {
                return ((rocket.massCentre / rocket.diameter) - (pitchMomentCoefficient / normalForceCoefficient))
* rocket.diameter
            } else {
                return rocket.massCentre
            }
        }
        data.pressureCentre.append(pressureCentre)
```

```
        let acceleration = (weight + thrust + axialDrag + normalForce) / totalMass
        data.acceleration.append(acceleration)

        let angularAcceleration = simd_cross((rocket.pressureCentre - rocket.massCentre) *
-simd_normalize(data.orientation[loop - 1]), normalForce).z / rocket.rotationalInertia
        data.angularAcceleration.append(angularAcceleration)

        let velocity = data.velocity[loop - 1] + (data.acceleration[loop - 1] + acceleration) / 2 *
settings.period
        data.velocity.append(velocity)

        let angularVelocity = data.angularVelocity[loop - 1] + (data.angularAcceleration[loop - 1] +
angularAcceleration) * settings.period
        data.angularVelocity.append(angularVelocity)

        let translation = ((data.velocity[loop - 1] + velocity) / 2) * settings.period
        let position = data.position[loop - 1] + translation
        data.position.append(position)

        var rotation: Double {
            if position.y > 0 {
                return ((data.angularVelocity[loop - 1] + angularVelocity) / 2) * settings.period
            } else {
                return 0
            }
        }
        let orientation = simd_normalize(simd_double2(x: data.orientation[loop - 1].x * cos(rotation) -
data.orientation[loop - 1].y * sin(rotation), y: data.orientation[loop - 1].x * sin(rotation) +
data.orientation[loop - 1].y * cos(rotation)))
        data.orientation.append(orientation)

        loop += 1

        let limit = 500

        if loop > limit {
            running = false
        }
    }

    return data
}

flightSim(rocket: Rocket(), conditions: Conditions(), settings: Settings())
```

Bibliography

[1] - Nave, C. R. (2017). *Moment of Inertia*. Hyperphysics. Retrieved from http://hyperphysics.phy-astr.gsu.edu/hbase/mi.html

[2] - Coker, J. (2022). *Thrustcurve Hobby Rocket Motor Data*. ThrustCurve. Retrieved from https://www.thrustcurve.org/

[3] - Nakayama, Y. (2018). Drag and Lift. In Introduction to Fluid Mechanics (pp. 177–201). Elsevier. https://doi.org/10.1016/b978-0-08-102437-9.00009-7

[4] - Maxemow, S. (2013). That's a Drag: The Effects of Drag Forces. In Undergraduate Journal of Mathematical Modeling: One + Two (Vol. 2, Issue 1). University of South Florida Libraries. https://doi.org/10.5038/2326-3652.2.1.4

[5] - Hall, N. (2021, May 13). *Similarity Parameters*. NASA. Retrieved from https://www.grc.nasa.gov/www/k-12/airplane/airsim.html

[6] - Jorgensen, L. H. (1973, January 1). Prediction of static aerodynamic characteristics for space-shuttle-like and other bodies at angles of attack from 0 deg to 180 deg. NASA. Retrieved from https://ntrs.nasa.gov/citations/19730006261

[7] - Barrowman, J. S. (1967, March 1). The practical calculation of the aerodynamic characteristics of slender finned vehicles. NASA. Retrieved from https://ntrs.nasa.gov/citations/20010047838

[8] - Cheeseman, I. (1976). Fluid-Dynamic Drag: Practical Information on Aerodynamic Drag and Hydrodynamic Resistance. S. F. Hoerner. Hoerner Fluid Dynamics, Brick Town, New Jersey. 1965. 455 pp. Illustrated. $24.20. *The Aeronautical Journal (1968), 80*(788), 371-371. doi:10.1017/S0001924000034187

[9] - Moore, F. G., Wilcox, F., & Hymer, T. (1994). Base drag prediction on missile configurations. In Journal of Spacecraft and Rockets (Vol. 31, Issue 5, pp. 759–765). American Institute of Aeronautics and Astronautics (AIAA). https://doi.org/10.2514/3.26509

[10] - Moore, F.G., Wilcox, F.J., & Hymer, T.C. (1992). Improved Empirical Model for Base Drag Prediction on Missile Configurations Based on New Wind Tunnel Data.

[11] - Hoerner, S. F. (1965). Fluid-dynamic drag: theoretical, experimental and statistical information. Retrieved from http://ftp.demec.ufpr.br/disciplinas/TM240/Marchi/Bibliografia/Hoerner.pdf

[12] - Convict, L. L., & Faro, I. D. V. (1961). Handbook of Supersonic Aerodynamics. section 8. Bodies of Revolution. Defense Technical Information Center. Retrieved from https://apps.dtic.mil/sti/pdfs/ADA279187.pdf

[13] - Vallini, L. (2015). Static and dynamic analysis of the aerodynamic stability and trajectory simulation of a student sounding rocket. Retrieved from https://etd.adm.unipi.it/t/etd-04012015-151632/

[14] - Gaunaa, M. (2006). Unsteady 2D potential-flow forces on a thin variable geometry airfoil undergoing arbitrary motion. Denmark. Forskningscenter Risoe. Risoe-R No. 1478(EN) Retrieved from https://orbit.dtu.dk/en/publications/unsteady-2d-potential-flow-forces-on-a-thin-variable-geometry-air

[15] - Barrowman, J. S. (1966, April 1). FIN - A computer program for calculating the aerodynamic characteristics of fins at supersonic speeds. NASA. Retrieved from https://ntrs.nasa.gov/citations/19660021056

[16] - United States. (1968). Design of aerodynamically stabilized free rockets. Washington, D.C: Headquarters, U.S. Army Materiel Command. Retrieved from http://mae-nas.eng.usu.edu/MAE_5900_Web/5900/USLI_2010/PDF_files/rocket_handbook.pdf

[17] - Niskanen, S. (2013, May 10). OpenRocket technical documentation. OpenRocket. Retrieved from https://openrocket.info/documentation.html

[18] - Barrowman, J. S. & Barrowman, J. A. (1966, August 18). The theoretical prediction of the center of pressure. Retrieved from http://www.nar.org/wp-content/uploads/2016/01/barrowman_cp_extended_edition.pdf