# Overview

Derive a tangent plane equation to represent Drag as an approximation in the velocity and density dimensions. This might be expanded by linearizing around the CD domain as well using alpha as a state value. This linearization is only valid for the fall stage. It is to serve as part of a program to either calculate an LQR controller, or simulate a model for TensorFlow neural net training.

# Expected Flight Regime

* Velocity: 40 – 110 m/s (78 – 214 knots)
* Height: 0 – 4268 m (0 – 14,000 ft)
  + Implicit: Air density: ρSL(1.00 – 0.65)
* Angle of attack: (-10) – (10) deg

# Calculations – 1 Dimensional

The equation for a tangent plane is generalized below, where the finite series repeats for every dimension X, Y, Z, and so on…:

Where:

* f\_approx – The approximate value and linearized function
* f(~) – The original function being approximated
* vec(P0) – A vector with all of the initial dimension points (x0,y0,z0,…etc.)

For the necessary calculations it will be easier to treat the height variable as a density ratio relative to sea level. This will allow me to vary the sea level density for different weather and ASL values. The original drag equation and its’ derivation is as follows:

Where:

* D – Drag force
* S\_A – Surface area
* C\_D – drag coefficient
* ρ\_SL – Sea level density
* ρ/ρ0 – density ratio at some height relative to a reference density (In our case, ISA sea level density)
* v – Velocity

The derived version of the initial equation lumps all of the constant values into one variable (for now we will assume C\_D to be constant and re-evaluate it later). The air density ratio is one dimension, which varies with height, and v^2 is the 2nd dimension we are linearizing around. For simplicity, the density ratio will furthermore be denoted as “S”. For further simplicity the constant values will be lumped into a variable denoted as “β”. See below…

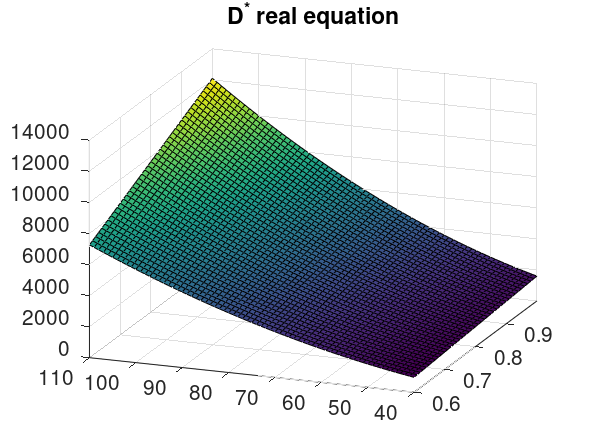
We can ascend one more level of abstraction by the following:

,

Now we must only linearize D\*, and plug it into the drag equation to approximate drag given a craft designed with β parameters.

,

The flight regime is already planar, shown below. High speeds are more dominant in non-linearizing the function than density changes due to the velocity value exponent.



We can add a bias variable B to the equation to better capture this nonlinearity in our inevitable approximation. We can computationally calculate the B variable using a method similar to the least squared regression algorithm. Thus, the planar equation is as follows…

Given the flight regime, it is most appropriate to select vec(P0) as the middle of all dimensions. Vec(P0) = <0.825, 75>. See below.

|  |  |
| --- | --- |
| Variable | Value |
| Vec(P0) | <0.825, 75> |
| df/dS | 5625 |
| df/dv | 123.75 |
| F(P0) | 4641 |

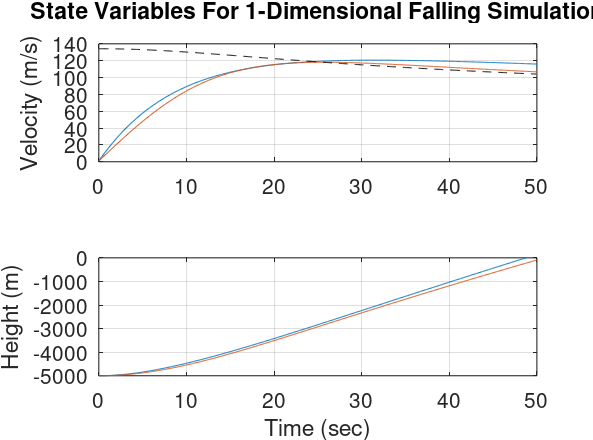
This approximation is decent as-is, but if we calculate B by finding the value which minimizes the square of the local error values, we get B = +367. Therefore…

Because our state variables include height (h) and not S, it would be smart to also linearize S around the variable h. Instead of calculating the tangent line for S, the least squares regression is used. Additionally, under the North-East-Down convention, any altitude above sea level is actually considered negative, and a descent towards the ground is considered to be positive velocity. We will change the signs of the equation to accommodate this convention. The linearized equation plugged into S in units of meters and distributed resulting in the equation:

Where:

* Dapprox\* - The approximated function
* h – height above sea level in meters
* v – velocity in meters per second

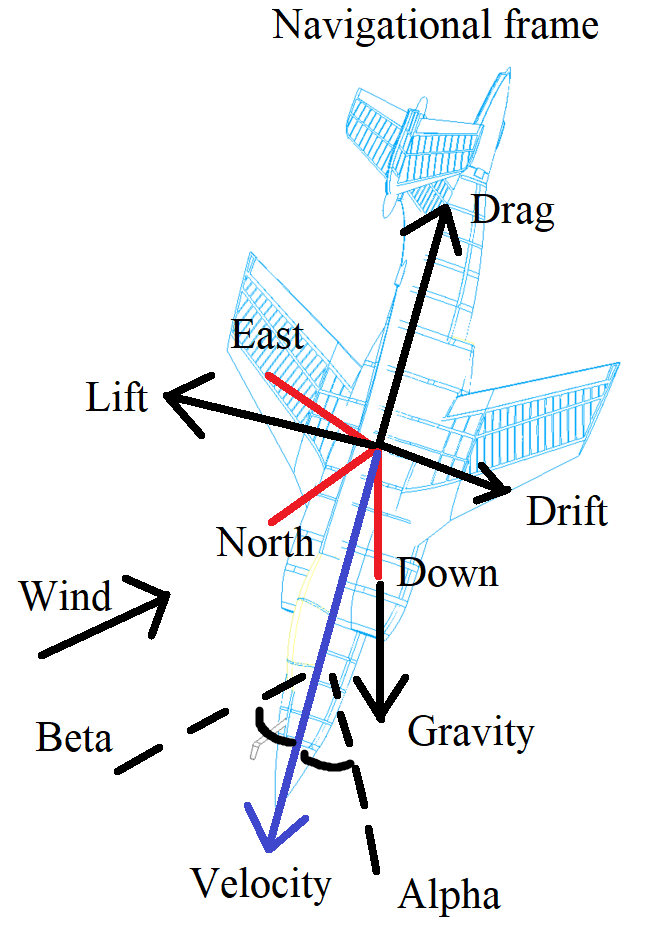
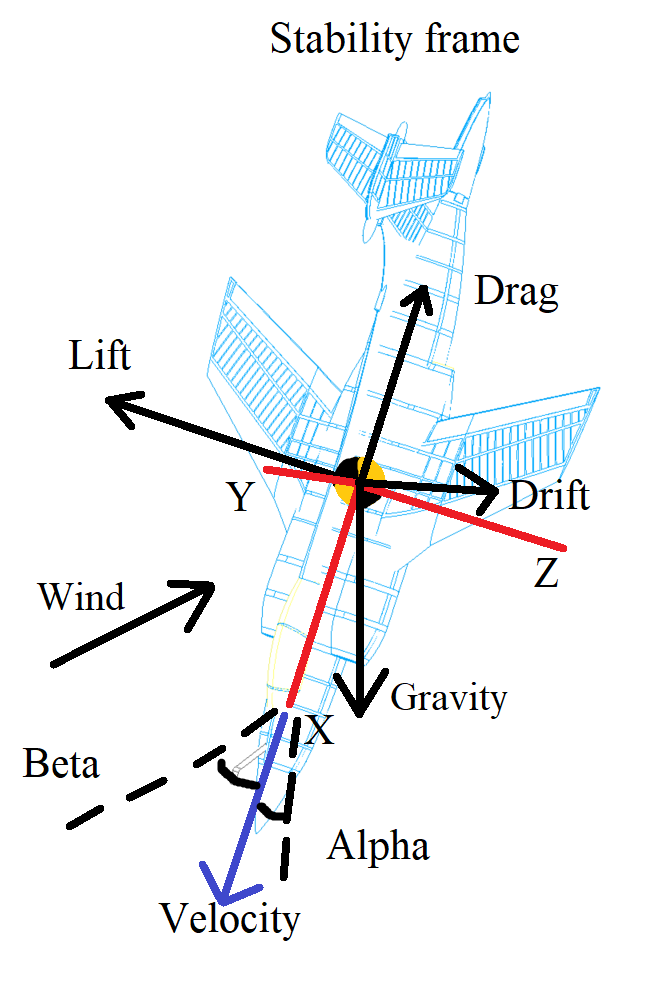
This is an important step in our function! We have linearized around the dimensions of height and velocity. If we stopped here, this would be enough to create a 1 dimensional falling simulation. Such a simulation however, is not very useful, because it disregards wind and an initial horizontal velocity. However, for the sake of testing our equation, here is a graph showing velocity over time using the linearized equation.



Notice how the initial condition for height is negative and the velocity is increasing in value. As a reminder, this is because of the North-East-Down convention. The blue line represents our approximation function. The red line is a more realistic Euler’s method ODE solution using the original drag equation. The dashed black line is the real terminal velocity given the flight parameters for the height of Euler’s method solution. We can see very clearly that the approximation is quite close to Euler’s method, and varies only by at most ~10 m/s. This error can propagate in the form of significant height error over time, but our free-fall is not expected to last longer than 1 minute, so this is of little concern. The state space form of our approximated model is shown below. This overly simplistic model is simply an A matrix due to a lack of disturbance or control input.

# Calculations – Full Spatial

To expand to the full spatial equations, we have a few challenges to adopt. We need to include the drag coefficient in our linearization so that we can account for varying angles of attack. At the same time we must also model lift very similarly to how we modelled drag. We must also model wind, a seemingly random 3-dimensional force vector that, for all intents and purposes, disturbs the angle of attack and sideslip states. All of this must be accomplished with compensation for reference frame conversions, as the lift and drag vectors act in the stability axis whereas the gravity and wind act in the navigational axis. For starters let’s create a free body diagram in the stability and navigational frames showing all of the forces and angles in play.



It is in my humble opinion that the simulation is best kept in the stability frame. This is because the lift, drag, and drift vectors are more complex and approximated, so finding conversions would only lead to more ambiguity in the linearization. Additionally, the wind is not a calculated vector, it stems from a lookup table by a weather forecast or external data, so it can be converted actively in the lookup table while the orientation changes in the navigational frame. The only vector to be concerned about converting between frames then, is the gravity vector. We will run into problems during the pitch over portion of the flight when transitioning from horizontal to vertical flight, but if we can overcome those problems the vertical portion of the fall stage can be easily kept under control with modelling constraints and small angle theory.

Given the use of the stability frame, we already have our approximation for the drag force. However, we need to again linearize it, this time around a new dimension: CD. Because in these three dimensions, the drag coefficient will have a major impact on the change in state variables it is inevitable that we must assume some variability with angle of attack.