Design Tool Report – Performance Analysis Tool

Justin Bradford, Sam Coyle, Peter Kim

Background

Our tool's objective is to simulate the airplane's performance throughout a designated route. The code incorporates both the engine and weather models into the overall performance model, utilizing information from the sizing and drag polar tools as key inputs. Through integration with other project groups, our tool enhances model fidelity, enabling the validation of a design's ability to break Voyager's record.

The tool's output is the airplane state throughout the user designated mission. This includes information such as remaining fuel, altitude, distance, groundspeed, true airspeed, power, specific fuel consumption, lift coefficient, and drag coefficient. Beyond performance simulation, our tool serves a crucial role in optimization. Configured for rapid testing of diversified designs and routes, it opens the door for identification of optimal parameters and flight plans.

Methodology

The methodology involves the utilization of two primary input categories: aircraft parameters and mission parameters. Aircraft parameters inputs encompass takeoff weight, drag polar, wing area, propeller efficiency, engine minimum specific fuel consumption, engine power, and engine type. Mission parameters include elements like weather collection step, control points, flight date, and flight modes. The main script reads these inputs and initializes data through the weather and engine models for simulation purposes. Subsequently, different flight modes are executed based on the user's specified mission parameters. The program incorporates built-in warnings to alert users of undesired states during the simulation, resulting in an organized matrix presenting the airplane's state and corresponding mission parameters. Key parameters within the airplane state include time, distance, weight, altitude, airspeed, ground speed, power, SFC, lift coefficient, drag coefficient, and flight mode.

The tool encompasses four distinct flight modes: a full throttle climb for efficient ascent during takeoff, a constant rate of climb useful for reaching cruise altitude, complying with air traffic control, or avoiding adverse weather conditions, a constant altitude and airspeed cruise mode, which predominates during the majority of the mission, and a cruise climb function, allowing for a gradual ascent while fuel weight decreases. Additionally, the constant rate of climb function can be used for descent.

All models within the tool employ Euler integration to update the aircraft's state. The best climb rate function determines the most efficient climb rate at each time step during a full throttle climb. The fixed climb function specifies an airspeed and rate of climb, solving for the power required to maintain these conditions. The cruise climb function, given a constant airspeed and lift coefficient, solves for density in the lift equation to determine altitude and climb rate. The constant altitude and airspeed function finds the power that equalizes thrust and drag to prevent acceleration.

The accuracy of results hinges on the weather and engine functions. The engine function represents an average gas or diesel engine, resulting in specific fuel consumption estimation. Users can enhance fidelity by inputting precise engine deck details. The weather function, an

average wind model with interpolated data, introduces uncertainty in wind effects. Another source of error arises from the assumption of no crosswind. However, the weather function indicates an average crosswind of about 5 knots, leading to only a 0.1% difference in distance for a record distance flight. Additionally, the fidelity of the model can be adjusted by changing the time step of the aircraft state calculations. This allows the user to rapidly test with a larger time step or have a higher fidelity model with a small time step.

Limitations and Assumptions

Our model operates as a point mass 2-degree-of-freedom simulation. Distance traveled and altitude are the two degrees of freedom. Designed as a mission performance tool, it does not include a takeoff or landing simulation, it begins with a full throttle climb and concludes with a constant rate of climb descent. Wind considerations are limited to the flight direction. This is because an average crosswind of 5 knots would only increase the still air distance by 0.1%. Additionally, our model operates under the assumption of international standard atmospheric conditions.

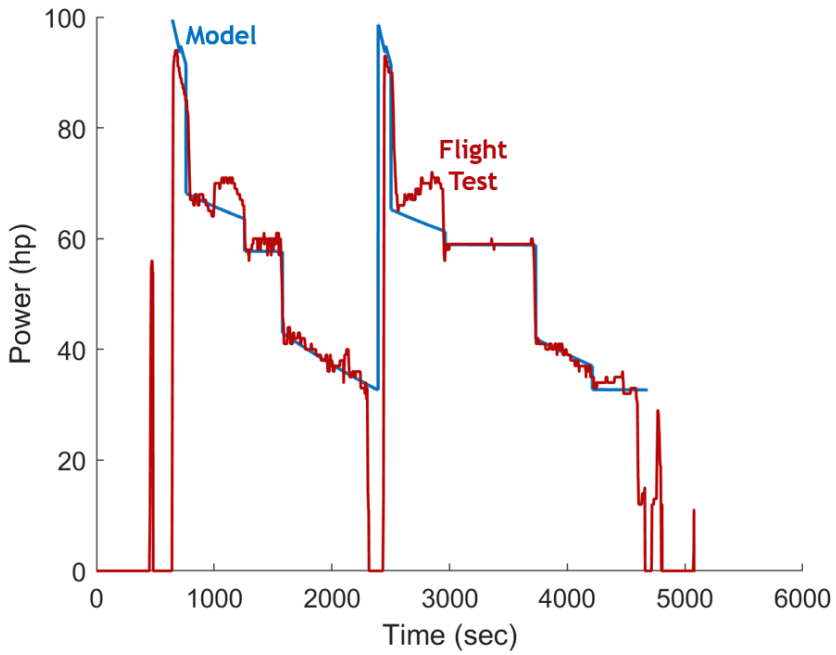
Possible discontinuities may arise between flight modes. For instance, if a user configures a constant rate of climb resulting in a 90-knot velocity and subsequently sets up a constant velocity cruise at 110 knots, the code may experience instantaneous jumps in velocity, specific fuel consumption, and power. Given the relatively brief duration of this scenario within the 7 to 8-day mission timeframe, its impact on overall fuel consumption and flight time is deemed negligible. Users are notified of these discontinuities and have the option to address them if necessary.

The model is constrained in its analysis to gas and diesel-powered aircraft. Nevertheless, due to the incorporation of drag polar and sizing capabilities, the model is versatile and can analyze a broad spectrum of aircraft categories and types.

Validation

The validation of this tool was conducted in two parts. Firstly, it was compared with real data from a test flight, providing confidence in the tool's ability to predict performance across various flight modes. Secondly, it was compared with existing performance calculations, solidifying confidence in the tool's ability to predict overall performance of intended missions.

The flight data used was collected from an RV7 which performed various flight modes. The tool took in the RV7's aircraft and mission parameters. The engine's performance was based off an existing model and the wind was assumed to be negligible. Key metrics in this comparison included power draw and fuel consumption, and the tool correctly predicted the RV7's performance, showing matching trends and closely aligned data points. This outcome provided confidence that the flight modes were working and ready for further testing.



Plot of RV7's real and simulated engine power output over the duration of the flight

To validate the tool's overall performance, a simulated around-the-world flight was conducted, and its results were compared with ones derived from the Breguet range equation. Despite its simplicity, the Breguet range equation is a reliable and widely accepted method for calculating ballpark estimations. Comparing the tool against this equation provided quick reality checks and confidence that predictions were in the right range. Additionally, this comparison allowed for wind effects to be considered, with an average tailwind factored into the Breguet calculation and modeled within the simulation. The resulting comparison showed that the performance tool and the previous calculations aligned closely, confirming that the tool was valid for overall mission performance analysis as well.

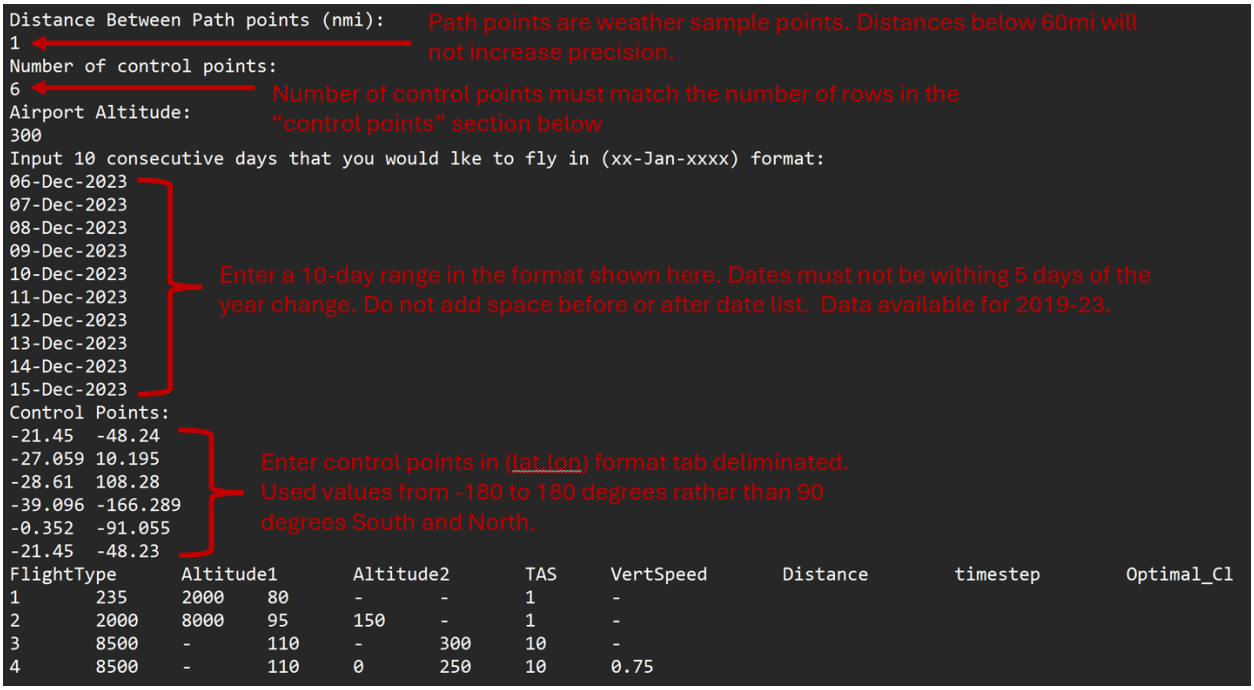
	Design Point	Simulation Result
Fuel Used	7000 lbs	6408 lbs
Flight Time	7 days 2 hours	7 days 5 hours
Average Cruise Power	85 HP	72 HP
Average Cruise C_L	0.69	0.71
Average Ground Speed	117 knots	115 knots

Table of calculated and simulated around-the-world flight results

The combination of these two validation methods provided confidence in the tool's capability to perform effectively for its intended purpose.

User Guide

To use this software, it is important to understand its inputs, outputs, and limitations. The input to this tool is a pair of text files with specific formatting. One defines the parameters of the flight and the other the aircraft. In both inputs, the formatting is extremely important. This program reads these text files line by line so it will not function as intended if formatting is altered. One notable exception to this is the matrices for control points and flight sectors in the "Aircraft Sector Split" file.



The last section of this "Aircraft Sector Split" file is the sector info that will be imported as the "Sectors" variable. This section can be as long or as short as the user would like, with columns being tab delineated. The content of these columns can be seen above the inputs with, but no flight mode requires all of them. For information that is not needed for a given sector leave "-" as can be seen above. The "FlightType" column on the left denotes the type of flight mode for the sector and has four possible options: best climb, level change climb/descent, constant altitude cruise, and cruise climb. Additionally, the "timestep" input represents the update rate for the aircraft's state vector. Although the time step can be set at a constant value, varying this input depending on flight type can greatly reduce the run time of this software.

Flight Mode One

The first flight mode represents a full throttle climb at the airplane's best climb rate. This flight type only requires a starting and ending altitude and the airspeed desired for the climb. It is assumed that this will primarily be used for takeoff and is not recommended for climbing in flight. For this section it is important that "altitude1" is lower than "altitude2" and that the

timestep is small enough to fully capture the climb. It is recommended this section has the smallest time step of any mode.

Flight Mode Two

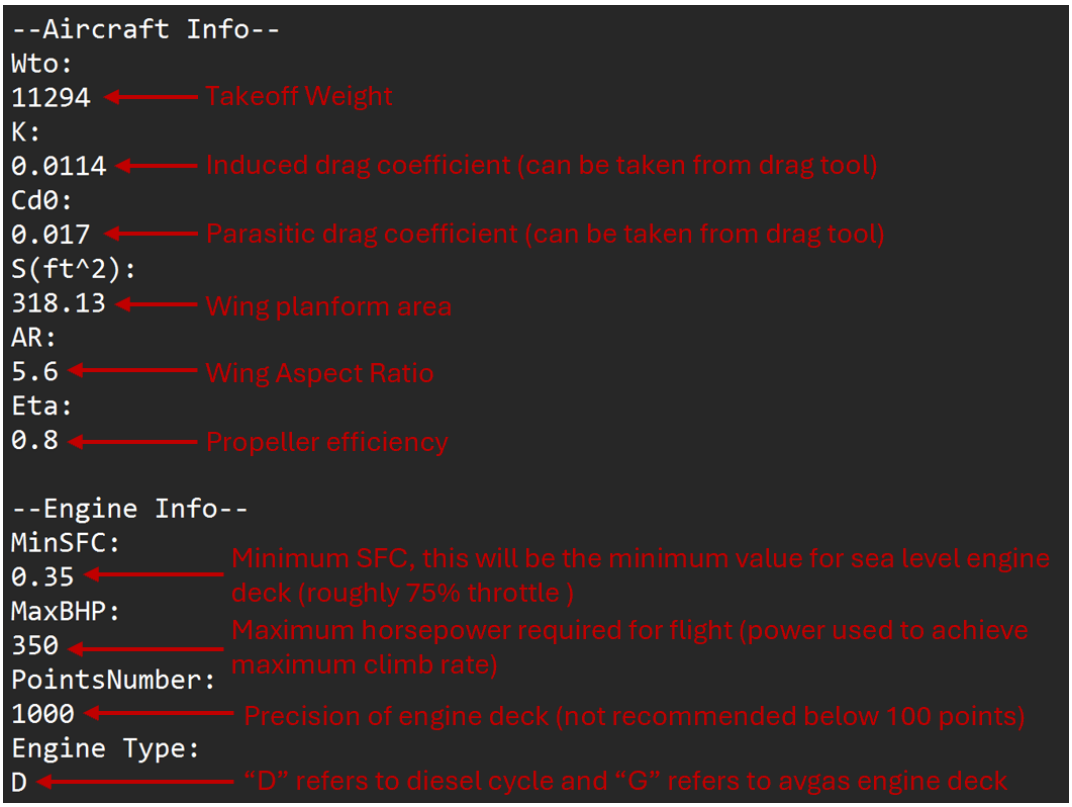
Flight mode two represents a level change climb with a constant climb rate. This mode is designed to mimic the "lvl/ch" autopilot mode used by airbus to change flight levels at a constant vertical speed. This mode requires a starting and ending altitude like the best climb, but now has a climb rate input called "VertSpeed" in ft/min. This mode can be used for descents as well by inputting a larger starting altitude than ending altitude. Descents will require the user to input a negative vertical speed.

Flight Mode Three

Flight mode represents a constant altitude and constant velocity cruise. This flight mode takes inputs of altitude put in the "altitude1" column, airspeed, distance, and time step. Both cruise modes use distance to define the time spent in cruise conditions. For this section there is more room to increase timestep to decrease run time without compromising the output.

Flight Mode Four

The final flight mode is a "cruise climb" keeping airspeed and lift coefficient constant. This flight mode takes the same inputs as flight mode three but with the addition of a cruise lift coefficient. The cruise lift coefficient can be taken from the drag tool. In most cases, the input lift coefficient should coincide with the lift coefficient for the maximum lift-to-drag ratio. This mode will likely not be used for the entire cruise portion but may offer a good option for climbing throughout the route.



The “Aircraft Info” text file contains aircraft constants to be used throughout the model. This file is broken into two distinct sections: aircraft info and engine info. The former primarily defines the aircraft’s initial states and constants. The latter defines the inputs to the engine matrix tool.

This software is broken into three parts for the calculation of the aircraft state vector. The first section loads the aircraft information, the second initiates meteorological information, and the final and main section integrates the aircraft state vector along the prescribed flight profile.

Aircraft State

The first section takes inputs from the “Aircraft Info” file to get the aircraft’s constants and initial states. This section first initialized the values in the aircraft state vector “AS” that will eventually be the primary output after the integration is performed. This state vector contains the aircraft’s flight time, flight distance, weight, altitude, airspeed, groundspeed, power, SFC, lift coefficient, drag coefficient, and flight mode.

This section also utilized the “BuildEngineDeck” function developed by team E. This engine deck defines the SFC for every throttle setting and requires the “ChangeEngineAlt” function to be run within the flight mode functions to alter the engine deck depending on the aircraft’s altitude.

Weather Model

The second section of this program utilizes a function called “profileTeamF” to load NOAA global weather data. This section loads the entire global wind profile as the “all_tailwind” variable. This data is loaded outside the loop to allow for computational efficiency. The weather data is taken over a ten-day period of the user’s choosing and a set of control points for the flight route.

Integration Scheme

The main part of this is the looped integration scheme in which the aircraft’s state vector is updated along the flight path. This section consists of a loop that steps through the “sectors” matrix pulled from the flight path text file. This is one of the four functions used to iterate through the selected flight mode. The selection of the flight mode type depends on the mode number in the first column of the “sectors” matrix before pulling the rest of the matrix to define the given section of the flight.

Outputs

The primary output of this function is the same aircraft state vector “AS” that was initialized before the main loop. This output shows the state of the aircraft for the entire flight, meaning the data within can be plotted and analyzed by the user. Some sample plots can be found in the final section.

Appendix

Main Script

```
clc; close all; clear all;

% Aircraft Performance MAIN

% Perfromance script to integrate:
% > Aircraft Configuration/Engine Selection
% > Route
% > Weather: Wing vectors along route
% > Flight profile
%% Aircraft Configuration
aircraft_info = readlines('Aircraft Info.txt');
Wto = str2double(aircraft_info(3)); % Takeoff Weight
AP = [str2double(aircraft_info(9)),str2double(aircraft_info(11)),str2double(aircraft_info(13))]; %
AP = [wing area (ft^2),AR, Eta]
k = str2double(aircraft_info(5)); % Induced Drag Constant
Cd0 = str2double(aircraft_info(7)); % Parasitic Drag

%% Engine Selection
MinSFC = str2double(aircraft_info(17));
MaxBHP = str2double(aircraft_info(19));
EngineType = aircraft_info(23);
n = str2double(aircraft_info(21));
[SeaLevelEngine] = BuildEngineDeck(EngineType, MinSFC, MaxBHP, n);

%% Route and Weather
% Import Flight Profile from text file
sector_filename = "Aircraft Sector Split 6.txt"
route = readlines(sector_filename); %%%
point_dist = str2double(route(2));
n_control = str2double(route(4));
alt_airport = str2double(route(6));
dates = route(8:17);
route = readmatrix(sector_filename); %%%
control_points_latlong = route(1:n_control,1:2);
sectors = route(2+n_control:length(route),1:8);
sectors_column_labels = {'FlightType', 'Altitude1 [ft]', 'Altitude2 [ft]', 'TAS [knots]',
        'VertSpeed', 'Distance [NM]', 'timestep [sec]', 'Optimal_Cl'};
sectors_table = array2table(sectors, 'VariableNames', sectors_column_labels);


%% Weather Initialization
global WindX
global WindY
global WindTime
```

```
global WindPAlt
global speedU
global speedV
initWind()
control_points_longlat(:,1) = control_points_latlong(:,2);
control_points_longlat(:,2) = control_points_latlong(:,1);
% Define the number of days of data you want to store
numArrays = length(dates);
% Initialize a cell array to store all the tail wing data arrays
all_tailwind = cell(1, numArrays);
for i = 1:length(dates)
   [all_tailwind{i},cross,u,v,distance,point_distance,latitudes,longitudes,unitx,unity] =
profileTeamF(control_points_longlat, point_dist, dates(i));
   disp("Weather Data for Day " + i + " Collected")
end
lat_long = [latitudes',longitudes'];

% load("Dec1-Dec10_all_tailwind.mat")
% load("SouthHem-Team1-100NM.mat")
%% Flight Profile
% Aircraft State Vector:
% AS = [time(s), distance(nmi), weight(lb), altitude(ft), TAS(knots), ground speed (knots),
Power, SFC Cl, Cd, mode]
AS(1,:) = zeros(1,11);
AS(1,3) = Wto;
sizeSEC = size(sectors);

for i = 1:sizeSEC(1)
% Flight Sector Analysis
   % 1: Full Throttle Climb
   if sectors(i,1) == 1
      sizeAS = size(AS);
      j = sizeAS(1);
      [time,x,W,alt,P,v,x_dot,sfc,Cl,Cd] =
best_climb(AS(j,3),sectors(i,2),sectors(i,3),AP(3),0,sectors(i,7),EngineType,SeaLevelEngine,Mi
nSFC,n,AP(1),k,Cd0);
      newAS =
[max(AS(:,1))+time',max(AS(:,2))+x',W',alt',v',x_dot',P',sfc',Cl',Cd',sectors(i)*ones(length(time),
1)];
      AS = [AS;newAS];
   % 2: Level Change Climb/Descent
   elseif sectors(i,1) == 2
      sizeAS = size(AS);
      j = sizeAS(1);
```

```
        [time,x,W,P,alt,v,x_dot,sfc,Cl,Cd] =
NavLvlChange(AS(j,3),sectors(i,4),sectors(i,2),sectors(i,3),AP(3),0,sectors(i,7),sectors(i,5),Engi
neType,SeaLevelEngine,MinSFC,n,AP(1),k,Cd0);
        newAS =
[max(AS(:,1))+time',max(AS(:,2))+x',W',alt',v',x_dot',P',sfc',Cl',Cd',sectors(i)*ones(length(time),
1)];
        AS = [AS;newAS];
    % 3: Cruise, constant alt, constant TAS
    elseif sectors(i,1) == 3
        [AS] =
cruise_cnst_v_h_final(AS,AP,sectors(i,6),sectors(i,4),sectors(i,7),all_tailwind,distance,EngineTy
pe,SeaLevelEngine,MinSFC,n,k,Cd0);

    % 4: Cruise, constant Cl, constant TAS
    elseif sectors(i,1) == 4
        Optimal_Cl = sectors(i,8);
        Optimal_Cd = Cd0 + k*(Optimal_Cl^2);
        [AS] = cruise_cnst_CL_v2(sectors(i,7), AP,
AS,EngineType,SeaLevelEngine,MinSFC,n,all_tailwind,distance,sectors(i,4),sectors(i,6),Optima
l_Cl,Optimal_Cd);

        disp("Cruise Climb ended at iteration: " + length(AS))
    else
        error('Input Valid Sector Type (1-4)')
    end
end

column_labels = {'Time [sec]','Distance [NM]', 'Weight [lbf]', 'Altitude [ft]', 'Airspeed [knots]',
'Ground Speed [knots]', 'Power [hp]', 'SFC', 'CL', 'CD', 'Mode #'};
AS_table = array2table(AS, 'VariableNames', column_labels);

%% plotting

figure
plot(AS(:,2),AS(:,4))
title('Altitude')
xlabel('dist (nmi)')
ylabel('alt (ft)')

% figure
% plot(AS(:,2),AS(:,3))
% title('weight')
% xlabel('dist (nmi)')
% ylabel('total weight (lbs)')

% figure
```

```
% plot(AS(:,2),AS(:,7))
% title('Power')
% xlabel('dist (nmi)')
% ylabel('power (hp)')

figure
hold on
groundspeedAVG = mean(AS(2782:end,6))
plot(AS(2782:end,2),AS(2782:end,5),LineWidth=1.5)
plot(AS(2782:end,2),AS(2782:end,6),LineWidth=1.1, Color=[0.722 0.027 0.027])
yl1 = yline(groundspeedAVG,"--",LineWidth=1.2);
yl1.Color = [0.722 0.027 0.027];
xlabel('Distance (nautical miles)')
ylabel('Speed (knots)')
xlim([0,AS(end,2)])

%% Quick Maths
day6 = all_tailwind{6};
for i = 1:4
   average_tailwind_start(i,1) = mean(day6(i,1:145));
   average_tailwind_end(i,1) = mean(day6(i,145:end));
end

disp("Flew " + AS(end,2) + " NM")
disp("Average Cruise CL: " + mean(AS(3924:end,9)))
disp("Average Cruise Power: " + mean(AS(3924:end,7)))
```

Flight Mode One - Best Climb

```
function [time,x,W,alt,P,v,x_dot,sfc,Cl,Cd] =
best_climb(Wi,alt1,alt2,eta,delta_T,delta_time,EngineType,SeaLevelEngine,MinSFC,n,S,k,Cd0)
% INPUTS:
% Wi - initial weight in lbf or lbm*(ft/s2)
% alt1 - Initial Altitude
% alt1 - Final Altitude
% speed - cruise speed in knots
% eta - propeller efficiency
% delta_T - off standard temperature difference in Fahrenheit
% delta_time - time step in seconds
% SeaLevelMatrix - engine matrix
% MinSFC - engine function input
% n - engine function input

% OUTPUTS
% x - distance in nautical miles
% W - final weight in lbf
% P - power in hp
```

```
% time - seconds
% alt - change in altitude

% conversions
ft2meter = 0.3048; % feet to meters
mile2ft = 6076.12; % nautical mile to feet
kgm3_2_slugft3 = 0.00194032; % kg per meter cubed to slug per foot cubed
knots2ftps = 1.68781; % knots to feet per second

% constants
[T_isa_SI,~,~,rho_SI] = atmosisa(alt1*ft2meter); % matlab isa
Ti_isa = (T_isa_SI - 273.15)*(9/5)+32; % Kelvin to Fahrenheit
Ti = Ti_isa + delta_T; % Fahrenheit
rhoi = rho_SI*kgm3_2_slugft3 ; % slug/ft3
g0 = 32.174; % ft per second squared (assuming gravity is constant with altitude)

% initialize state vector
x(1) = 0;
W(1) = Wi;
time(1) = 0;
rho(1) = rhoi;
alt(1) = alt1;
path(1) = 2; % initial path angle (degrees)

% Euler
i = 1;
while alt < alt2
   v(i) = sqrt(2*W(i))/(rho(i)*S) * ((k/(3*Cd0))^0.25);
   Cl(i) = (2*(W(i)*cosd(path(i))))/(rho(i)*S*(v(i))^2);
   Cd(i) = Cd0 + (k*(Cl(i)^2)); %from drag polar given
   D(i) = 0.5*rho(i)*Cd(i)*S*(v(i))^2; % lbf
   T = D(i);

   % Power
   [AdjEngineDeck] = ChangeEngineAlt(EngineType,SeaLevelEngine,MinSFC,alt(i),n); %
change Power Check
   p_avail(i) = max(AdjEngineDeck(:,1)); % max available shaft power
   P_req(i) = (((D(i)*v(i))/eta)/550)*eta; % divide by 550 to put into hp
   if P_req(i) > p_avail(i)
      error('Required Power Exceeds Available Power')
   end
   percent_power(i) = (P_req(i)/p_avail(i))*100;
   sfc(i) = AdjEngineDeck(round(percent_power(i)),2);

   % Current State
   x_dot(i) = v(i)*cosd(path(i)); % ft/s
```

```
    W_dot = (-sfc(i)*P_req(i))/3600; % divide by 3600 to get in lbf/s
    v_dot = (T-D(i))/((W(i)*cosd(path(i)))/g0); % ft/s2

    % Update State
    x(i+1) = x_dot(i)*delta_time + x(i);
    W(i+1) = W_dot*delta_time + W(i);
    v(i+1) = v_dot*delta_time + v(i);
    time(i+1) = time(i) + delta_time;
    path(i+1) = -asind(((p_avail(i)*eta)/(v(i)*W(i)))-(D(i)/W(i)));
    alt(i+1) = alt(i) + x_dot(i)*sind(path(i));

    % Atmosphere at updated altitude
    [T_isa_SI,~,~,rho_SI] = atmosisa(alt(i)*ft2meter); % matlab isa
    Ti_isa = (T_isa_SI - 273.15)*(9/5)+32; % Kelvin to Fahrenheit
    T(i+1) = Ti_isa + delta_T; % Fahrenheit
    rho(i+1) = rho_SI*kgm3_2_slugft3 ; % slug/ft3
    Cl(i+1) = Cl(i);
    Cd(i+1) = Cd(i);
    i = i + 1;
end
x_dot = x_dot/knots2ftps;
v = v/knots2ftps; % convert velocity vector back to knots
x = x/mile2ft; % convert feet to nautical miles
x_dot(length(x_dot)+1) = x_dot(length(x_dot));
time = 1:(length(alt)*delta_time);
P = p_avail;
P(length(P)+1) = P(length(P));
sfc(length(sfc)+1) = sfc(length(sfc));
end
```

Flight Mode Two – Constant Rate Climb

```
function [time,x,W,P,alt,v,x_dot,sfc,Cl,Cd] =
NavLvlChange(Wi,Vi,alt1,alt2,eta,delta_T,delta_time,climb_rate,EngineType,SeaLevelEngine,M
inSFC,n,S,k,Cd0)

% INPUTS:
% Wi - initial weight in lbf or lbm*(ft/s2)
% alt1 - Initial Altitude
% alt1 - Final Altitude
% speed - cruise speed in knots
% eta - propeller efficiency
% delta_T - off standard temperature difference in Fahrenheit
% delta_time - time step in seconds
% SeaLevelMatrix - engine matrix
% MinSFC - engine function input
% n - engine function input
```

```
% OUTPUTS
% Cl - lift coefficient
% Cd - drag coefficient
% x - distance in nautical miles
% W - final weight in lbf
% v - TAS in knots
% P - power in hp
% time - seconds

% conversions and constants
ft2meter = 0.3048; % feet to meters
mile2ft = 6076.12; % nautical mile to feet
kgm3_2_slugft3 = 0.00194032; % kg per meter cubed to slug per foot cubed
knots2ftps = 1.68781; % knots to feet per second
min2sec = 1/60; % sec
HPconv = 550; % lb*ft/s

% initialize state vector
x(1) = 0;
W(1) = Wi; % Initial weight
time(1) = 0;
alt(1) = alt1; % Initial altitude (ft)
v(1) = Vi*knots2ftps; % Initial velicity (ft/s)
climb_rate = climb_rate*min2sec;
path(1) = real(asin(climb_rate/(v(1)))); % Path Angle (degrees)
Cl = [];
Cd = [];

if alt1 < alt2
   % Climb Euler
   i = 1;
   while alt < alt2
      % Atmosphere Model
      [T_isa_SI,~,~,rho_SI] = atmosisa(alt(i)*ft2meter); % matlab isa
      Ti_isa = (T_isa_SI - 273.15)*(9/5)+32; % Kelvin to Fahrenheit
      T(i) = Ti_isa + delta_T; % Fahrenheit
      rho(i) = rho_SI*kgm3_2_slugft3 ; % slug/ft3

      % Lift and climb
      Cl(i) = (2*W(i))/(rho(i)*v(i)^2*S);
      Cd(i) = Cd0 + (k*(Cl(i)^2)); %from drag polar given
      D(i) = 0.5*rho(i)*Cd(i)*S*(v(i))^2; % lbf
      path(i) = real(asin(climb_rate/v(i))); % path angle
      rate_climb(i) = v(i) * sin(path(i)); % climb rate
```

```
        % Climb Power
        [AdjEngineDeck] = ChangeEngineAlt(EngineType,SeaLevelEngine,MinSFC,alt(i),n); %
change power matrix
        power_req_level(i) = (D(i)*v(i))/(HPconv*eta); % sea level power
        power_req(i) = power_req_level(i) + ((rate_climb(i)*W(i))/HPconv); % required climb
power
        p_avail(i) = max(AdjEngineDeck(:,1)); % max available shaft power
        if power_req(i) > p_avail(i)
           warning('Required Power Exceeds Available Power')
        end
        percent_power(i) = power_req(i)/p_avail(i)*100;
        sfc(i) = AdjEngineDeck(round(percent_power(i)),2);

        % Current State
        x_dot(i) = v(i)*cos(path(i)); % ft/s
        W_dot = (-sfc(i)*power_req(i))/3600; % divide by 3600 to get in lbf/s

        % Update State
        x(i+1) = x_dot(i)*delta_time + x(i);
        W(i+1) = W_dot*delta_time + W(i);
        time(i+1) = time(i) + delta_time;
        alt(i+1) = alt(i) + v(i)*sin(path(i));
        Cl(i+1) = Cl(i);
        Cd(i+1) = Cd(i);

        % Velocity Change (should not be required)
        if rate_climb < climb_rate
          v(i+1) = v(i) + 1;
        elseif rate_climb > climb_rate
          v(i+1) = v(i) - 1;
        elseif rate_climb == climb_rate
          v(i+1) = v(i);
        end
        i = i + 1;
    end
elseif alt1 > alt2
        % Descent Euler
    i = 1;
    while alt > alt2
        % Atmosphere Model
        [T_isa_SI,~,~,rho_SI] = atmosisa(alt(i)*ft2meter); % matlab isa
        Ti_isa = (T_isa_SI - 273.15)*(9/5)+32; % Kelvin to Fahrenheit
        T(i) = Ti_isa + delta_T; % Fahrenheit
        rho(i) = rho_SI*kgm3_2_slugft3 ; % slug/ft3

        % Lift and Descent
```

```
        Cl(i) = (2*W(i))/(rho(i)*v(i)^2*S);
        Cd(i) = Cd0 + (k*(Cl(i)^2)); %from drag polar given
        D(i) = 0.5*rho(i)*Cd(i)*S*(v(i))^2; % lbf
        path(i) = real(asin(climb_rate/v(i))); % path angle
        rate_climb(i) = v(i) * sin(path(i)); % climb rate

        % Descent Power
        [AdjEngineDeck] = ChangeEngineAlt(EngineType,SeaLevelEngine,MinSFC,alt(i),n); %
change
        power_req_level(i) = (D(i)*v(i))/(HPconv*eta); % sea level power
        power_req(i) = power_req_level(i) + ((rate_climb(i)*W(i))/HPconv); % required climb
power
        p_avail(i) = max(AdjEngineDeck(:,1)); % max available shaft power
        if power_req(i) > p_avail(i)
           warning('Required Power Exceeds Available Power')
        end
        percent_power(i) = power_req(i)/p_avail*100;
        sfc(i) = AdjEngineDeck(round(percent_power(i)),2);

        % Current State
        x_dot(i) = v(i)*cos(path(i)); % ft/s
        W_dot = (-sfc(i)*power_req(i))/3600; % divide by 3600 to get in lbf/s

        % Update State
        x(i+1) = x_dot(i)*delta_time + x(i);
        W(i+1) = W_dot*delta_time + W(i);
        time(i+1) = time(i) + delta_time;
        alt(i+1) = alt(i) + v(i)*sin(path(i));
        Cl(i+1) = Cl(i);
        Cd(i+1) = Cd(i);

        % Velocity Change (should not be required)
        if rate_climb < climb_rate
          v(i+1) = v(i) + 1;
        elseif rate_climb > climb_rate
          v(i+1) = v(i) - 1;
        elseif rate_climb == climb_rate
          v(i+1) = v(i);
        end
        i = i + 1;
    end
elseif alt1 == alt2
    error('Input altitudes must differ (0-15k ft)')
end
v = v/knots2ftps; % convert velocity vector back to knots
x = x/mile2ft; % convert feet to nautical miles
```

```
time = (1:length(alt))*delta_time;
P = power_req;
P(length(P)+1) = P(length(P));
sfc(length(sfc)+1) = sfc(length(sfc));
x_dot(length(x_dot)+1) = x_dot(length(x_dot));
x_dot = x_dot/knots2ftps;
end
```

Flight Mode Three – Const. Alt & Airspeed Cruise

```
function [time,x,W,P,alt,v,x_dot,sfc,Cl,Cd] =
NavLvlChange(Wi,Vi,alt1,alt2,eta,delta_T,delta_time,climb_rate,EngineType,SeaLevelEngine,M
inSFC,n,S,k,Cd0)

% INPUTS:
% Wi - initial weight in lbf or lbm*(ft/s2)
% alt1 - Initial Altitude
% alt1 - Final Altitude
% speed - cruise speed in knots
% eta - propeller efficiency
% delta_T - off standard temperature difference in Fahrenheit
% delta_time - time step in seconds
% SeaLevelMatrix - engine matrix
% MinSFC - engine function input
% n - engine function input

% OUTPUTS
% Cl - lift coefficient
% Cd - drag coefficient
% x - distance in nautical miles
% W - final weight in lbf
% v - TAS in knots
% P - power in hp
% time - seconds

% conversions and constants
ft2meter = 0.3048; % feet to meters
mile2ft = 6076.12; % nautical mile to feet
kgm3_2_slugft3 = 0.00194032; % kg per meter cubed to slug per foot cubed
knots2ftps = 1.68781; % knots to feet per second
min2sec = 1/60; % sec
HPconv = 550; % lb*ft/s

% initialize state vector
x(1) = 0;
W(1) = Wi; % Initial weight
time(1) = 0;
```

```
alt(1) = alt1; % Initial altitude (ft)
v(1) = Vi*knots2ftps; % Initial velicity (ft/s)
climb_rate = climb_rate*min2sec;
path(1) = real(asin(climb_rate/(v(1)))); % Path Angle (degrees)
Cl = [];
Cd = [];

if alt1 < alt2
   % Climb Euler
   i = 1;
   while alt < alt2
      % Atmosphere Model
      [T_isa_SI,~,~,rho_SI] = atmosisa(alt(i)*ft2meter); % matlab isa
      Ti_isa = (T_isa_SI - 273.15)*(9/5)+32; % Kelvin to Fahrenheit
      T(i) = Ti_isa + delta_T; % Fahrenheit
      rho(i) = rho_SI*kgm3_2_slugft3 ; % slug/ft3

      % Lift and climb
      Cl(i) = (2*W(i))/(rho(i)*v(i)^2*S);
      Cd(i) = Cd0 + (k*(Cl(i)^2)); %from drag polar given
      D(i) = 0.5*rho(i)*Cd(i)*S*(v(i))^2; % lbf
      path(i) = real(asin(climb_rate/v(i))); % path angle
      rate_climb(i) = v(i) * sin(path(i)); % climb rate

      % Climb Power
      [AdjEngineDeck] = ChangeEngineAlt(EngineType,SeaLevelEngine,MinSFC,alt(i),n); %
change power matrix
      power_req_level(i) = (D(i)*v(i))/(HPconv*eta); % sea level power
      power_req(i) = power_req_level(i) + ((rate_climb(i)*W(i))/HPconv); % required climb
power
      p_avail(i) = max(AdjEngineDeck(:,1)); % max available shaft power
      if power_req(i) > p_avail(i)
         warning('Required Power Exceeds Available Power')
      end
      percent_power(i) = power_req(i)/p_avail(i)*100;
      sfc(i) = AdjEngineDeck(round(percent_power(i)),2);

      % Current State
      x_dot(i) = v(i)*cos(path(i)); % ft/s
      W_dot = (-sfc(i)*power_req(i))/3600; % divide by 3600 to get in lbf/s

      % Update State
      x(i+1) = x_dot(i)*delta_time + x(i);
      W(i+1) = W_dot*delta_time + W(i);
      time(i+1) = time(i) + delta_time;
      alt(i+1) = alt(i) + v(i)*sin(path(i));
```

```
        Cl(i+1) = Cl(i);
        Cd(i+1) = Cd(i);

        % Velocity Change (should not be required)
        if rate_climb < climb_rate
          v(i+1) = v(i) + 1;
        elseif rate_climb > climb_rate
          v(i+1) = v(i) - 1;
        elseif rate_climb == climb_rate
          v(i+1) = v(i);
        end
        i = i + 1;
    end
elseif alt1 > alt2
        % Descent Euler
    i = 1;
    while alt > alt2
        % Atmosphere Model
        [T_isa_SI,~,~,rho_SI] = atmosisa(alt(i)*ft2meter); % matlab isa
        Ti_isa = (T_isa_SI - 273.15)*(9/5)+32; % Kelvin to Fahrenheit
        T(i) = Ti_isa + delta_T; % Fahrenheit
        rho(i) = rho_SI*kgm3_2_slugft3 ; % slug/ft3

        % Lift and Descent
        Cl(i) = (2*W(i))/(rho(i)*v(i)^2*S);
        Cd(i) = Cd0 + (k*(Cl(i)^2)); %from drag polar given
        D(i) = 0.5*rho(i)*Cd(i)*S*(v(i))^2; % lbf
        path(i) = real(asin(climb_rate/v(i))); % path angle
        rate_climb(i) = v(i) * sin(path(i)); % climb rate

        % Descent Power
        [AdjEngineDeck] = ChangeEngineAlt(EngineType,SeaLevelEngine,MinSFC,alt(i),n); %
change
        power_req_level(i) = (D(i)*v(i))/(HPconv*eta); % sea level power
        power_req(i) = power_req_level(i) + ((rate_climb(i)*W(i))/HPconv); % required climb
power
        p_avail(i) = max(AdjEngineDeck(:,1)); % max available shaft power
        if power_req(i) > p_avail(i)
           warning('Required Power Exceeds Available Power')
        end
        percent_power(i) = power_req(i)/p_avail*100;
        sfc(i) = AdjEngineDeck(round(percent_power(i)),2);

        % Current State
        x_dot(i) = v(i)*cos(path(i)); % ft/s
        W_dot = (-sfc(i)*power_req(i))/3600; % divide by 3600 to get in lbf/s
```

```
        % Update State
        x(i+1) = x_dot(i)*delta_time + x(i);
        W(i+1) = W_dot*delta_time + W(i);
        time(i+1) = time(i) + delta_time;
        alt(i+1) = alt(i) + v(i)*sin(path(i));
        Cl(i+1) = Cl(i);
        Cd(i+1) = Cd(i);

        % Velocity Change (should not be required)
        if rate_climb < climb_rate
          v(i+1) = v(i) + 1;
        elseif rate_climb > climb_rate
          v(i+1) = v(i) - 1;
        elseif rate_climb == climb_rate
          v(i+1) = v(i);
        end
        i = i + 1;
    end
elseif alt1 == alt2
    error('Input altitudes must differ (0-15k ft)')
end
v = v/knots2ftps; % convert velocity vector back to knots
x = x/mile2ft; % convert feet to nautical miles
time = (1:length(alt))*delta_time;
P = power_req;
P(length(P)+1) = P(length(P));
sfc(length(sfc)+1) = sfc(length(sfc));
x_dot(length(x_dot)+1) = x_dot(length(x_dot));
x_dot = x_dot/knots2ftps;
end
```

Flight Mode Four – Cruise Climb

```
function [AS] = cruise_cnst_CL_v2(dt_sec, AP, AS, EngineType, SeaLevelEngine, MinSFC, n,
all_tailwind, distance, v_cruise_knots, distance_target_nm, CL_cruise, CD_cruise)
% AS [ time(sec), distance(nm), weight(lbf), altitude(ft), airspeed(knots), ground speed(knots),
power(hp), sfc, CL, CD, mode]
% AP [ ]
% SeaLevelMatrix - baseline engine performance produced by engine model
% all_tailwind - cell containing all tailwind data
% distance - matrix produced by the weather function that has the cumulative distnace traveled
per path point
% v_cruise_knots
% distance_target_nm
% CL_cruise
% CD_cruise
```

```
mode_number = 4;

% Make sure that the plane "catches up" and TODO

%Optional Allocation
guessLength = 10000;
mass = nan(guessLength,1);
density = nan(guessLength,1);           % density
D = nan(guessLength,1);
thrust_required = nan(guessLength,1);
power_required = nan(guessLength,1);
power_required_hp = nan(guessLength,1);
path_angle = nan(guessLength,1);
tailwind_fts = nan(guessLength,1);
power_shaft = nan(guessLength,1);
power_output = nan(guessLength,1);
dv = nan(guessLength,1);
dx = nan(guessLength,1);
dy = nan(guessLength,1);
dW = nan(guessLength,1);
t = nan(guessLength,1);
airspeed_fts = nan(guessLength,1);
groundspeed_fts = nan(guessLength,1);
x = nan(guessLength,1);
y = nan(guessLength,1);
W = nan(guessLength,1);
sfc = nan(guessLength,1);
fuel_consumption = nan(guessLength,1);

% Conversions
nm2ft = @(x) x*6076.11549;
ft2nm = @(x) x/6076.11549;
knots2fts = @(x) x*1.68780986;
fts2knots = @(x) x/1.68780986;

% Time Step
dt = dt_sec;

% Take in Airplane State
    % AS [ 1 time, 2 distance, 3 weight, 4 altitude, 5 airspeed, 6 ground speed, 7 power(hp), 8 sfc,
9 CL, 10 CD, 11 mode]
t(1) = AS(end,1);               % [sec]
x(1) = nm2ft(AS(end,2));        % [ft]
W(1) = AS(end,3);               % Starting weight [lbf]
y(1) = AS(end, 4);              % Starting Altitude [ft]
```

```
   density(1) = stdAtmosphere_imperial(y(1),0);
airspeed_fts(1) = knots2fts(AS(end, 5));   % [ft/s]

% Aircraft Parameters
S = AP(1);      % Wing Area [ft^2]
eta = AP(3);    % Propeller Efficiency


%% Mission Parameters
   % Target Cruise Velocity
v_cruise_fts = knots2fts(v_cruise_knots);      % [ft/s]
   % CL & CD
CL = CL_cruise;
CD = CD_cruise;
   % Mission Distance
distance_target_ft = nm2ft(distance_target_nm);   % Distance to travel [ft]

i = 1;
i_AS = size(AS,1);                % Sets the starting point of the function at the end of the Airplane
State (AS)
while x(i) <= x(1)+distance_target_ft
   % House Keeping
   mass(i) = W(i)/32.2;            % [lbm]power

   % Weather
   tailwind_fts(i) = knots2fts(windFinder(AS(i_AS,:),all_tailwind,distance));              % [ft/s]
   groundspeed_fts(i) = v_cruise_fts + tailwind_fts(i);                    % [ft/s]

   % Density Climb
   density(i+1) = 2*W(i) / (S * CL * v_cruise_fts^2);                    % [slugs/ft^3]
      % nextdensity [slug/ft^3] needed for the plane to be in cruise
   y(i+1) = equivalent_std_density_alt_finder(density(i+1));   % next altitude [ft],


   % Solving for Power Required
   dy(i) = y(i+1) - y(i);             % if there is no density change (weight change), this should be
zero
   airspeed_fts(i) = v_cruise_fts;                                % [ft/s]
   path_angle(i) = atan2(dy(i),airspeed_fts(i));                    % [rad]
   D(i) = 0.5 * density(i) * v_cruise_fts^2 * S * CD;      % [lbf]
   thrust_required(i) = W(i)*sin(path_angle(i)) + D(i);    % [lbf]

   power_required(i) = thrust_required(i) * v_cruise_fts;      % [lbf * ft/s]
   power_required_hp(i) = power_required(i)/550;          % [hp]
   % find required shaft power
   power_shaft(i) = (power_required_hp(i)/eta);           % [hp]
```

```
    % Solving for Power Output and SFC
    [AdjEngineDeck] = ChangeEngineAlt(EngineType, SeaLevelEngine, MinSFC, y(i), n);   % y(i) because we are solving for power needed NOW to solve for power needed to get to next state
    solve = abs(AdjEngineDeck(:,1) - power_shaft(i));
    [~,I] = min(solve);
    power_output(i) = AdjEngineDeck(I,1);
    sfc(i) = AdjEngineDeck(I,2);

    % Fuel Consumption
    fuel_consumption(i) = sfc(i) * power_output(i)/3600;                % lbf/sec

    % Update Derivative
    dv(i) = (power_required_hp(i) - power_output(i))/mass(i)/airspeed_fts(i);  % [ft/s^2] should be zero most of the time as the power output is adjusted to be close to the power
    dx(i) = groundspeed_fts(i);                                 % [ft/s]
    dW(i) = -fuel_consumption(i);                                  % [lbf]

    % Update State
    t(i+1) = dt+t(i);
    x(i+1) = dx(i)*dt + x(i);   %[ft]
    W(i+1) = dW(i)*dt + W(i);

    % Update Airplane State
    % AS [ 1 time, 2 distance, 3 weight, 4 altitude, 5 airspeed, 6 ground speed, 7 power(hp), 8 sfc, 9 CL, 10 CD, 11 mode]
    AS(i_AS+1, 1) = t(i+1);
    AS(i_AS+1, 2) = ft2nm(x(i+1));     % [NM]
    AS(i_AS+1, 3) = W(i+1);
    AS(i_AS+1, 4) = y(i+1);
    AS(i_AS+1, 5) = fts2knots(v_cruise_fts);
    AS(i_AS+1, 6) = fts2knots(groundspeed_fts(i));    % [knots]
    AS(i_AS+1, 7) = power_output(i);  %[hp]
    AS(i_AS+1, 8) = sfc(i);
    AS(i_AS+1, 9) = CL;
    AS(i_AS+1,10) = CD;
    AS(i_AS+1,11) = mode_number;

    if length(AS) > 4176
        disp(length(AS))
        disp(AS(i_AS+1))
        disp("The drag calculated is: " + D(i))
        disp("The climb component of thrust required is: " + W(i)*sin(path_angle(i)))
    end
```

```
    % Checkpoints
    if sfc(i) < 0
        disp("Negative SFC")
    elseif dx(i) < 0
        disp("Plane is moving backwards")
    elseif abs(dW(i)) >= 2
        disp("Plane is changing weight too fast")
%     elseif (v(i)-20) >= v_cruise_fts
%         disp("Plane is going too fast")
%     elseif y(i) >= maxCeiling
%         disp("Plane is too high")
    end
        %   Altitdue
    if dy(i) > 1000
        disp("A jump in altitude of " +  dy(i) + "ft was made")
    elseif dy(i) < -1000
        disp("A dive in altitude of " +  dy(i) + "ft was made")
    end
    %   Power
%     disp("bruh")
    i = i + 1;
    i_AS = i_AS + 1;

end
```