Author: Ryan Dunn
Position: DBF Propulsions Subteam Lead 2019-2021
Email: rcdunn@ucsd.edu

Overview:
In the 2020-2021 year, COVID-19 forced the team to operate virtually. As a result, a small suite of optimization and visualization scripts were written to ease the workflow of the Propulsion team. This document will describe the workflow, assumptions, and advice for these MATLAB scripts.

Import Data Script

Generate_Mat_File.m
imports various propeller datasheets and saves it to a .mat file
(Possible task: Finding new propeller datasheets and importing it into the script)

```
%% Input Sheets
MotorFile = 'Motor_data.xlsx';
PropFiles  = {'6x4E.xlsx'
              '7x4E.xlsx'
              '8x6E.xlsx'
              '8x8E.xlsx'
              '9x6.xlsx'
              '9x6E.xlsx'
              '10x5E.xlsx'
              '10x6.xlsx'
              '10x6E.xlsx'
              '10x7.xlsx'
              '10x7E.xlsx'
              '10x8E.xlsx'
              '10x9.xlsx'
              '10x10.xlsx'
              '10x10E.xlsx'
              '11x5.xlsx'
              '11x55E.xlsx'
              '11x6.xlsx'
              '11x7.xlsx'
              '11x7E.xlsx'
              '11x8E.xlsx'
              '11x9.xlsx'
              '11x10E.xlsx'
              '11x11.xlsx'
              '11x12E.xlsx'
              '12x8.xlsx'
              '12x8E.xlsx'
              '12x10E.xlsx'
              '13x4.xlsx'
              '13x65E.xlsx'
              '13x7.xlsx'
              '13x8.xlsx'
              '13x9.xlsx'
              '13x8E.xlsx'
```

Optimization Script

```
%% Initialize
clear all; close all; format longg; clc;
%% Input Parameters

outfile = 'Results.xlsx';

numProps = 2;   % Number of Propellers
Voltage = 22.2; % Voltage of Battery *NOTE* Voltage does change throughout
                % the flight, but the labeled volate is a fair average.

% MISSION REQUIREMENTS
% Based on values acquired from Aero Team: Drag (D) & Cruising speed (v)
% Assuming steady level flight: Thrust = Drag
% [Thrust CruiseAirspeed]
% [lbsf MPH]
Mreq{1} = [1.1/numProps 48.32];
Mreq{2} = [1.9/numProps 70.69];
Mreq{3} = [1.3/numProps 54.80];

% Import Propeller & Motor Datasheets
load('DataImport.mat')

%% Propeller Analysis & Mission Performance Criteria

fprintf('beginning propeller analysis\n');
% Mission-Propeller loop for calculating
max_eff=ones(3,1);
for MISSION=1:3
    for FILE=1:length(Propnames)
        Vmission{FILE} = zeros(1,3);
        for n=maxRPM(FILE):-1:1 % Find the minimum RPM where mission criteria is met
            req1 = T{FILE}{n} > Mreq{MISSION}(1);
            req2 = V{FILE}{n} > Mreq{MISSION}(2);
            if nnz(req1) && nnz(req2) % There exists a V where it provides enough thrust
                for x=30:-1:1
                    if req1(x) && req2(x)
```

Visualization Script

```
%% Imports

% Import Propeller & Motor Datasheets
load('DataImport.mat')

% Quick Motor Efficiency Analysis (SEE STANFORD PAPER)
for Motor = 1:length(Motornames)
    Kt(Motor) = 1355/Kv(Motor);
    RPMmax(Motor) = Kv(Motor) * (Voltage - Rm(Motor)*I0(Motor));

    Imax{Motor} = @(RPM)(Voltage - RPM/Kv(Motor)) / Rm(Motor);
    Qmotor{Motor} = @(A) Kt(Motor)*(A-I0(Motor))*0.007061552; % in-oz to N-m [i think]
    eta_motor{Motor} = @(RPM,A) (Kt(Motor)*(A-I0(Motor))*0.007061552*RPM*2*pi/60) / (Voltage*A);
end
% Find location where cruising speed is reached
for INDEX=1:30
    if V{cp}{RPMcruise/1000}(INDEX) >= speed
        break
    end
end
eta_Prop = Pe{cp}{RPMcruise/1000}(INDEX);

%% Motor Efficiency Contours
x = @(t) t;
y = @(t) (Voltage - t/Kv(cm)) / Rm(cm);
z = @(t) (Kt(cm)*(y(t)-I0(cm))*0.007061552*t*2*pi/60) / (Voltage*y(t));

%% Static Thrust [Thrust v. Amp draw]

figure; hold on

for i = 1:maxRPM(cp)
    Qcrit = Qprop{cp}{i}(1);
    I_temp = Qcrit/(Kt(cm)*0.007061552) + I0(cm);
    if I_temp > Imax{cm}(i*1000)
```

Flight Performance Script:

For the report, it was important to know how long it would take to go along the track three times, which is where the flight performance calculations covered in the "Guide to: Flight Performance" were used. Using these equations for velocities and angular velocity we were able to calculate how long the plane would take to go around the track because it had a common distance. Given the values described in the Guide from the other teams in DBF, the velocities were calculated and we divided distances by velocities to find the time it would take to go through each part of the lap. Then all those times were added up to get the expected time for one lap, then multiplied by three to get the total expected time and expected drag values. Thankfully, the MATLAB script does not require the derivations for the formulas, so only the final formulas were plugged in and run through.

The full scripts for Steady Climb, Steady Flight, Banked Turn, and Total Time can be found in the DBF propulsions folder. Here are pictures to give an idea of what they do:

Steady Climb:

```
function [v] = DBFSteadyClimb(mass,k,Cl,Cd0,ClimbAngle,MotorAngle,WingSurfaceArea)
%Inputs:
%mass(kg),k,Cl,Cd0,ClimbAngle(deg),MotorAngle(deg);WingSurfaceArea(m^3)
%Outputs: velocity(m/s)
%All calculations based off "Simulation Math" pdf

%Calculate drag coefficient
Cd = Cd0 + k*Cl^2;

%Re-label formulas to make formulas easier
M = mass;
L = Cl;
D = Cd;
S = WingSurfaceArea;

%Convert from degrees to radians
T = ClimbAngle*0.0174533;
A = MotorAngle*0.0174533;

%Assign constants
g = 9.81;
R = 1.225; %Air density (kg/m^3)

%Pre-calculations to make formula nicer
Q = cos(T);
W = sin(T);
E = tan(T+A);

%Calculate velocity
v = sqrt(2*M*g/(R*S*(L*Q+L*W*E+D*Q*E-D*W)));
```

Steady Flight:

```
function [v]=DBFSteadyFlight(mass,Cl,k,Cd0,MotorAngle,WingSurfaceArea)
%Inputs: mass(kg),Cl,k,Cd0,MotorAngle(deg),WingSurfaceArea(m^3)
%Outputs: velocity(m/s)
%All calculations based off "Simulation Math" pdf

%Calculate the drag coefficient
Cd = Cd0 + k*Cl^2;

%Re-label variables to make formulas easier
M = mass;
L = Cl;
D = Cd;
S = WingSurfaceArea;

%Convert from degrees to radians
a = MotorAngle*0.0174533;

%Assign constants
g = 9.81;
R = 1.225; %Density of air (kg/m^3)

%Calculate velocity
v = sqrt(2*M*g/(R*S*(L+D*tan(a))));
```

Banked Turn:

```
function [v,TurnRate,TurnRadius] = DBFBankTurn(mass,Cl,n,WingSurfaceArea)
%Inputs: mass(kg),Cl,n,WingSurfaceArea(m^3)
%Outputs: velocity(m/s),TurnRate(rad/s),TurnRadius(m)
%All calculations based off "Simulation Math" pdf

%Re-label formulas to make formulas easier
M = mass;
L = Cl;
Theta = acos(1/n);
S = WingSurfaceArea;

%Assign constants
g = 9.81;
R = 1.225; %Air density (kg/m^3)

%Calculate velocity
v = sqrt(2*M*g/(R*S*L*cos(Theta)));

%Calculate TurnRadius
TurnRadius = 2*M/(R*S*L*sin(Theta));

%Calculate TurnRate
TurnRate = g*tan(Theta);

[v,TurnRate,TurnRadius]
```

Total Time/Drag:

```
%Inputs
mass = 3.175; %kilograms
Cl = 0.277;
k = 0.067;
Cd0 = 0.032;
MotorAngle = 0; %degrees
WingSurfaceArea = .41032176; %m^3
n = 1.5; %g's
dist = 1000; %feet

%Straight
d_straight = dist*0.3048; %feet to meters
v_straight = DBFSteadyFlight(mass,Cl,k,Cd0,MotorAngle,WingSurfaceArea);
t_straight = d_straight/v_straight;

%180 Turn
d_180 = pi; %radians
[v_turn,TurnRate,TurnRadius] = DBFBankTurn(mass,Cl,n,WingSurfaceArea);
t_180 = d_180/TurnRate;

%360 Turn
d_360 = 2*pi; %radians
t_360 = d_360/TurnRate;

%Total time
t_total = 3*(2*t_180 + 2*t_straight + t_360); %seconds


%Drag calculation
drag_straight = 0.5 * 1.225 * (Cd0 + k*Cl^2) *WingSurfaceArea * (v_straight)^2;
drag_banked = 0.5 * 1.225 * (Cd0 + k*Cl^2) *WingSurfaceArea * (v_turn)^2;

%Outputs in useful units
t_total
v_straight = v_straight * 2.23694; %m/s to mph
v_banked = v_turn * 2.23694 %m/s to mph
drag_straight = drag_straight * 0.224809; %N to lbf
drag_banked = drag_banked * 0.224809 %N to lbf
```