Aero Project Final Report

Eric Larsen*

The purpose of this document is to be an overview of the Aerodynamic Project for USU's course MAE5510. This project is to design a glider for the 2024 Dynamics of Atmospheric Flight Glider Competition, adhering to the design requirements. This project goes through the design and optimization of this unpowered glider for both static and dynamic stability of the aircraft. The static and dynamic stability of an aircraft ensures the aircraft is able to maintain a desired flight plan without being uncontrollable due to disturbances. This project highlighted the influence aircraft components have on the stability of the aircraft, components being the main wing, horizontal stabilizer, and vertical stabilizers. The stability of the aircraft can be determined by analyzing five distinct modes on the aircraft, two longitudinal and three lateral modes. The longitudinal modes are known as the short period and phugiod mode, with lateral modes being the roll, dutch roll, and spiral modes.

I. Nomenclature

b	=	Wing span
$\overline{c}$	=	mean chord
CAP	=	Control Anticipation Parameter
CD	=	Coefficient of drag
CD_0	=	Coefficient of drag at zero lift
CL	=	Coefficient of lift
CL_max	=	Maximum coefficient of lift
CL_α	=	Lift slope
CW	=	Coefficient of weight
e	=	Oswald efficiency number
RA	=	Aspect ratio
SW	=	Planform area
t_{max}/c	=	Maximum thickness of airfoil ratio to chord length
V_{min}	=	Velocity to stall
V_{MDV}	=	Minimum power airspeed
V_{MD}	=	Minimum drag airspeed
W	=	Weight
W_a	=	Weight of aircraft
W_f	=	Weight of fuselage
W_w	=	Weight of weight
γ	=	stress
σ	=	specific weight
ρ	=	density

II. Introduction

The purpose of this project report is to demonstrate the process and requirements of creating a statically and dynamically stable aircraft. The ability to design and construct aircraft that are both statically and dynamically stable is essential when developing new aircraft to ensure proper performance. This project report will go through the personal development of a glider aircraft developed for the 2024 Dynamics of Atmospheric Flight Glider Competition. The glider must be constructed of carbon fiber, EPS foam, and use a tungsten ballast if desired. With these design requirements developed the aircraft can be designed to any desired configuration, with comparisons drawn to a baseline design

*Graduate Researcher, Mechanical and Aerospace Engineering, Utah State University

glider. The baseline glider stands as the project goal when designing our aircraft, the developed glider must be able to outperform the baseline glider in distance traveled. This requirement allows for individuals to implement corrections and optimization methods to create a higher performing glider. For this project the selected aircraft design was a conventional aircraft configuration using a standard horizontal and vertical stabilizer compared to a more complicated design such as a V tail configuration. The design of this aircraft will be reviewed throughout this document and the design process will be outlined. The updated glider configuration will then be compared to the baseline glider for expected performance from the engineering analysis performed. Finally, key points and design considerations for future work will be considered near the end of this report.

III. Aircraft Configuration

As stated before, the configuration for the updated glider remained the same conventional configuration as the baseline glider using a main wing, horizontal stabilizer, and vertical stabilizer (see fig1). This configuration was chosen for its ease of adaptation and to allow for increased understanding of how each component of the aircraft effects the overall stability and handling qualities of the aircraft. For a holistic view of the aircraft see fig. 1.

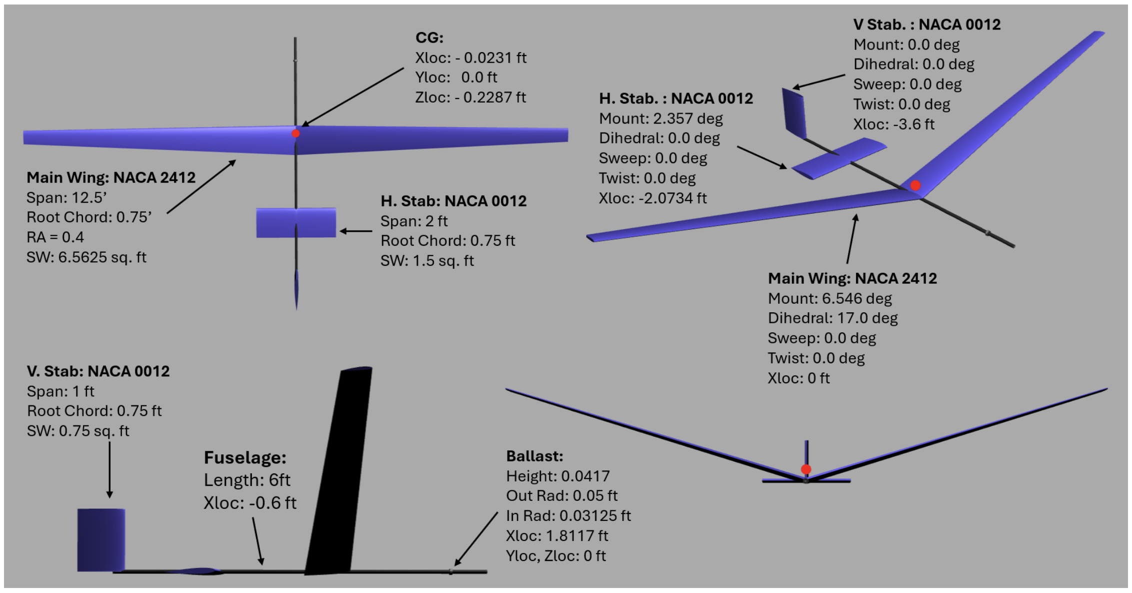


Fig. 1 Dimensional Specs of 6917 Glider

The main design differences between the updated aircraft and that of the baseline is in shifting the carbon fiber fuselage center location, location of the aircraft stabilizers, increasing the dihedral of the main wing, reducing the weight of the tungsten ballast and shifting the ballast location. For total mass properties of this glider see Table 1, with specific dimensions of the tungsten ballast are shown in Table 2

Table 1 Mass Properties of updated glider

Component	Weight [lbf]	X loc [ft]	Y loc [ft]	Z loc[ft]
Maing Wing	0.2421	0.0000	0.0000	0.0000
Horizontal Stabilizer	0.0740	-2.0734	0.0000	0.0000
Vertical Stabilizer	0.0370	-3.6000	0.0000	0.0000
Fuselage	0.1943	-0.600	0.0000	0.0000
Tungsten Ballast	0.2393	1.8117	0.0000	0.0000
CG	-	-0.0231	0.0000	-0.2287
Total Aircraft	0.7867	0.0000	0.0000	0.000

Table 2 Tungsten Ballast Dimensions

Component	Weight [lbf]	Inner Radius [ft]	Outer Radius [ft]	Height [ft]
Tungsten Ballast	0.2393	0.0313	0.0500	0.0416

With the weight of each component now know, focus is turned to understanding the airfoils used in the glider design. The design characteristics of each airfoil are showed below in Table 3

Table 3 Properties of Glider Airfoil Components

Property	Main Wing	Horizontal Stabilizer	Vertical Stabilizer	Total
Airfoil	NACA 2412	NACA 0012	NACA 0012	-
Semispan [ft]	6.25	1.0	1.0	-
Mean Chord [ft]	0.525	0.75	0.75	-
Taper Ratio [-]	0.4	1.00	1.00	-
Planform Area [ft^2]	6.563	1.5	0.75	**8.813**
Dihedral [deg]	18.00	0.00	0.00	-
Sweep [deg]	0.00	0.00	0.00	-
Twist [deg]	0.00	0.00	0.00	-
Mounting Angle [deg]	6.452	5.93	0.00	-

From the results shown in Table 3 it is shown that the updated glider is withing the design constraint that the total lifting-surface area must be under 9 square feet, where this glider has 8.813 square feet total.

IV. Aircraft Analysis

Creating the airfoils and placing them on the aircraft is one thing, the next thing step is to ensure they are structurally sound. To do so we will focus on the main wing and solve for the maximum allowed aspect ratio (RA) the aircraft can handle on the main wing using the known material properties of EPS foam and the weight of aircraft. These properties are then used in eq. 1

$$RA \leq \frac{16(t_{max}/c)\sigma_{max}W_w}{21W_f b\gamma} \tag{1}$$

where W_w is the wing weight, W_f is the fuselage weight, and parameters σ_{max} and γ are material properties of EPS foam; 7,200 lbf/ft^2 and 0.804 lbf/ft^3 respectively. Using this equation and known properties we get the maximum RA for this aircraft is

$$RA \leq \frac{16(t_{max}/c)\sigma_{max}W_w}{21W_f b\gamma} = \frac{16*(0.12)(7,200)(0.2421)}{21(0.43365)(0.804)} = 36.569 \tag{2}$$

From the properties of the main wing we can find its taper ratio as

$$RA = b/\overline{c} = \frac{12.5}{0.525} = 23.80952 \tag{3}$$

Therefore, we know the wing is structurally should because it is below the maximum aspect ratio allowed. Now the developed glider must be assest at its trim state to see the expected operating characteristic of the aircraft. For this project the glider was designed to be trim at an angle of attach of zero degrees. The design lift coefficient (CL) of the aircraft at this trim state is taken from Mach Up 6 as is shown below.

$$CL_{\text{design}} = 0.8325$$

With this lift coefficient and the known weight of the aircraft we can then solve for the trim velocity of the aircraft as follows in eq. 4 using the knowledge that at trim the weight of the aircraft is equal to the lift produced

$$V_{\text{trim}} = \sqrt{\frac{W_a}{1/2\rho C_L S_w}} = \sqrt{\frac{0.7867}{0.5 * (0.002048) * (0.8745) * 6.563}} = 11.522\ ft/s \tag{4}$$

V. Aircraft Design

The design process of this aircraft was a competitive process, not just trying to out perform the baseline glide, but competition between design variables. Throughout this semester emphasis has been placed on looking at each independent component of aircraft stability. Changing one attribute of the aircraft has implications on the rest of the aircraft as well. The first phase of this design was focused on the static stability of the aircraft alone. This was broken into two main sections, longitudinal and lateral stability. In longitudinal stability focus was first placed on ensuring the static margin of the aircraft was over the rule of thumb measurement of 0.05. After this was accomplished focus was placed on lateral stability focusing on the control derivative Cn_β to allow the aircraft to correct for deviations in the flight path. All these conditions were the same between the baseline design and the upgraded glider design. The focus on the upgraded glider was placed on improving the lift to drag ratio of the aircraft. This ratio has direct implications on the expected total flight distance of the aircraft. To increase this factor the upgraded design reduced the total area of the tungsten ballast and the total weight of the ballast. These changes to the aircraft effects the expected drag contributions from the fuselage (CD_0). Reducing this weight though influences the overall CG location of the aircraft and thus requires the adaptation of the ballast location and repositioning of other aircraft components to ensure a good static margin is maintained above the rule of thumb. With this constant back and forth of the aircraft there is several iterations on the aircraft and the final placement of each component.
This iterative design process was not exclusive to the static design of the aircraft. No, instead this was also seen when designing the aircraft for dynamic stability. During the analysis of the first design it was discovered there was a divergent spiral mode on the aircraft. In order to correct this the aircraft dihedral of the main wing was increased to compensate. Doing so raised additional design concerns and required the continual iteration of the aircraft to ensure all dynamic modes were convergent allowing for a statically and dynamically stable aircraft.
During the end stages focus was placed on ensuring all dynamic modes were convergent to create confidence in the aircraft overall. The resulting aircraft was then evaluated against the baseline model using the first analysis used to evaluate performance of the aircraft knowing the static stability conditions. The expected lift to drag ratio was checked to show the final aircraft is expected to fly further than the baseline glider. If this value was not met adaptations to the design were performed focusing on reducing drag via the main wing and fuselage and the design process was repeated until the glider met desired characteristics.

VI. Aircraft Comparison

This section of the report will focus on comparing the performance of the baseline glider to the upgraded glider. This comparison will be done through various angles of attack, but will be shown as a function of velocity of the aircraft. The compared features of the aircraft are the lift coefficient (fig. 2), drag coefficient (fig. 3), pitching moment coefficient (fig. 4), lift to drag ratio (fig. 5, drag (fig. 6), required power (fig.7), static margin (fig.8), no-wind glide ratio (fig.9), sink rate (fig.**??**), and the pitch roll and yaw stability derivatives (figs.11,12,13).
Looking at the lift coefficient between the two aircraft we can see there is only slight differences shown in fig. 2

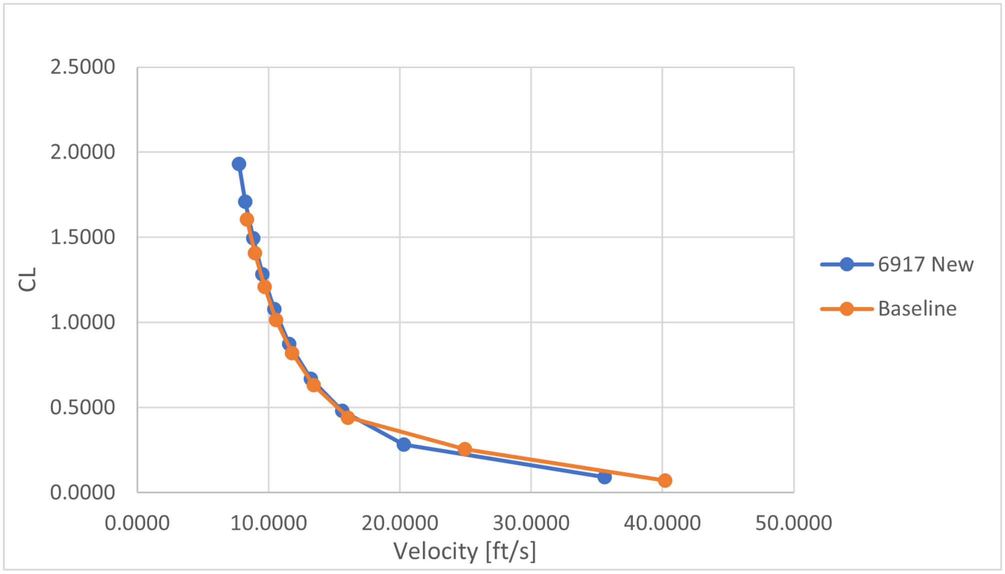


Fig. 2 Lift coefficient compared between two glider designs

The list distribution between these two aircraft are nearly identical as a function of velocity. They both have higher lift coefficients at lower velocities while that lift coefficient experiences and non-linear decay as you increase velocity. Next attention is put onto how these aircraft compare via their drag coefficient shown in fig. 3.

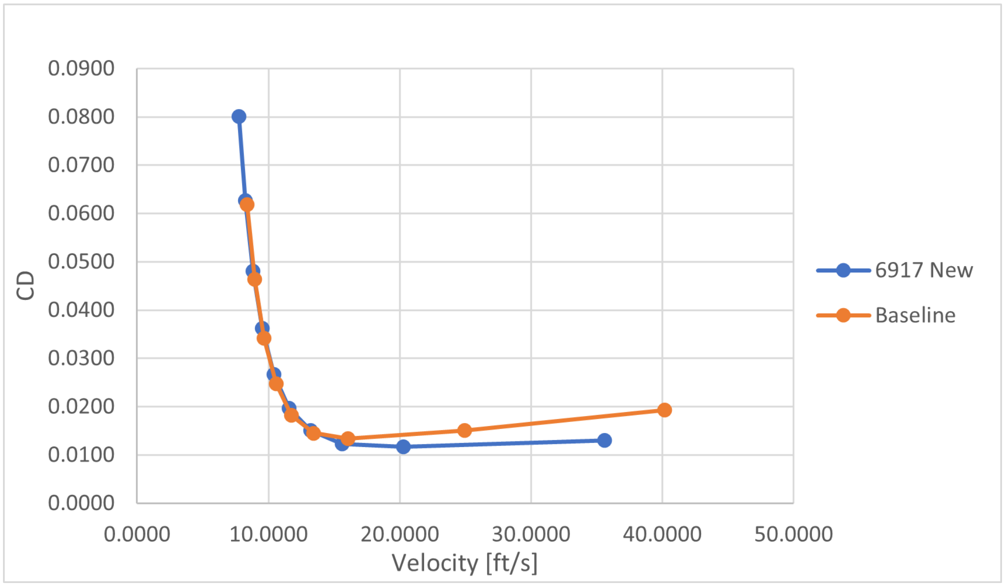


Fig. 3 Drag coefficient compared between two glider designs

Again these two aircraft show nearly identical drag profiles as a function of velocity. The only difference seen is at higher velocities as the baseline glider has a higher drag coefficient. If that trend were to continue you would see the new glider would produce less drag at higher velocities. One contributing factor of this is the decrease in drag contribution from the fuselage. Between the gliders the new glider has a lower area used for the ballast, this contributes to a lower CD_0 contribution from the fuselage overall, allowing for a reduction in drag. Comparing the aircraft on the basis of pitching moment coefficient is shown below in fig. 4

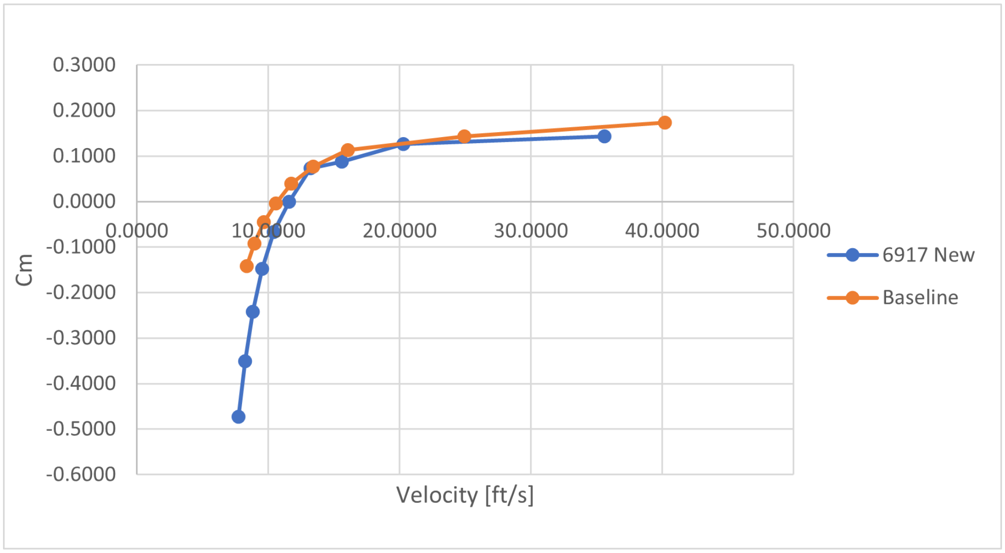


Fig. 4 Pitching moment coefficient compared between two glider designs

The results from fig. 4 it is clear to see one of the major differences between the aircraft. The pitching moment experienced by the new glider design has a higher magnitude than the baseline glider. This will result in a higher driving force to correct the aircraft when it deviates either up for down in angle of attack. This force will help the new aircraft be more stable in pitch than the baseline glider. That being said, pitch stability is not the only characteristic important to an aircraft. To continue the evaluation of these aircraft their lift to drag ratios were compared as a function of velocity.

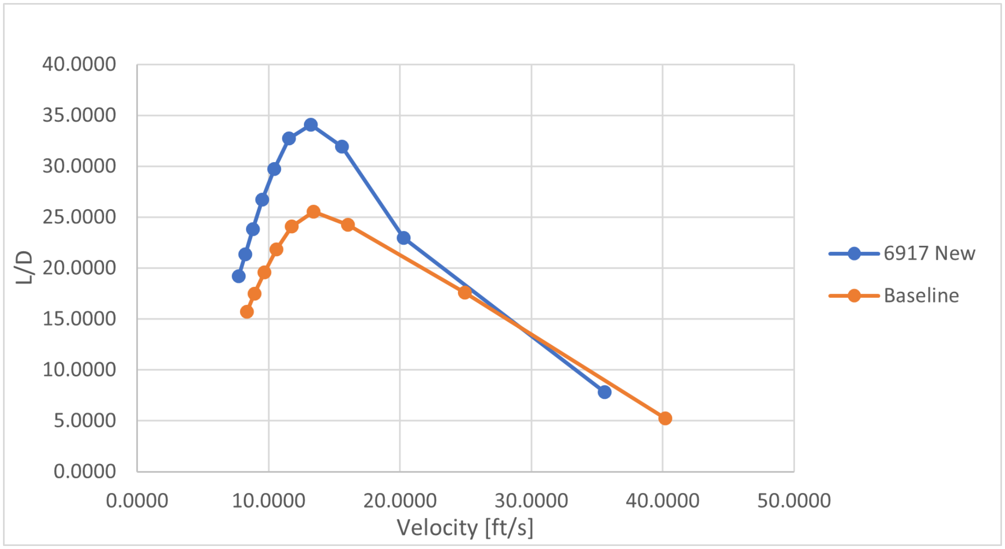


Fig. 5 Lift to drag ratio compared between two glider designs

It is clear from the results of fig. 5 the new glider has a higher lift to drag ratio than the baseline. This performance characteristic is key to improving the overall performance of a glider aircraft. With no means to have powered flight the glider relies on lift to drag ratio to extend the expected flight time of the aircraft. This ratio is not all though, the drag on the actual aircraft must be considered. The comparison of drag force on the aircraft is shown in fig. 6

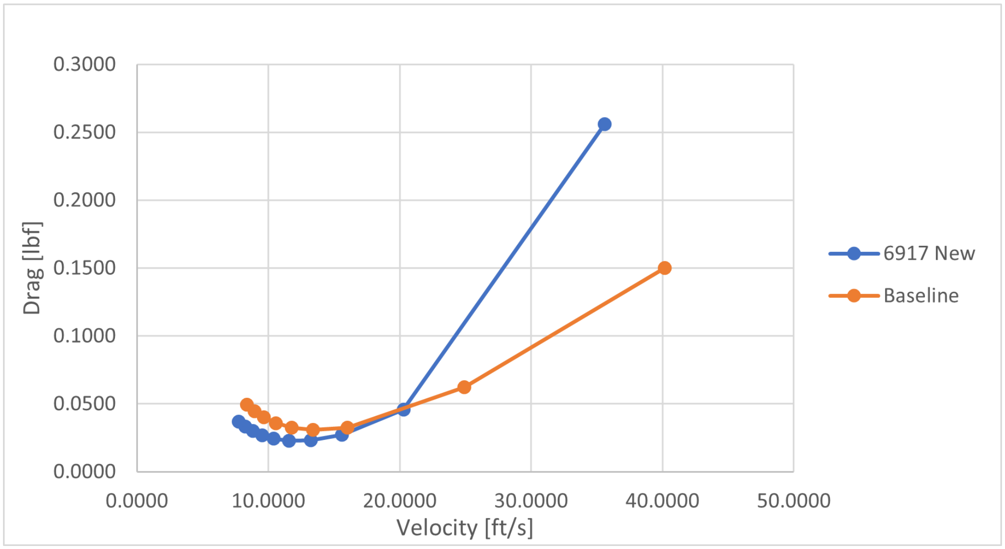


Fig. 6 Drag force compared between two glider designs

Figure 6 shows there is comparable drag forces experienced between the two aircraft. There is a reduction in the new aircraft due to the reduced drag contribution due to the fuselage. That being said, the new aircraft design does out perform the baseline glider design in terms of drag force experienced. This will reduce the expected sink rate of the aircraft and allow for a longer flight time. Any drag force on the aircraft creates the requirement of power to maintain flight. The required power of each glider is shown in fig. 7

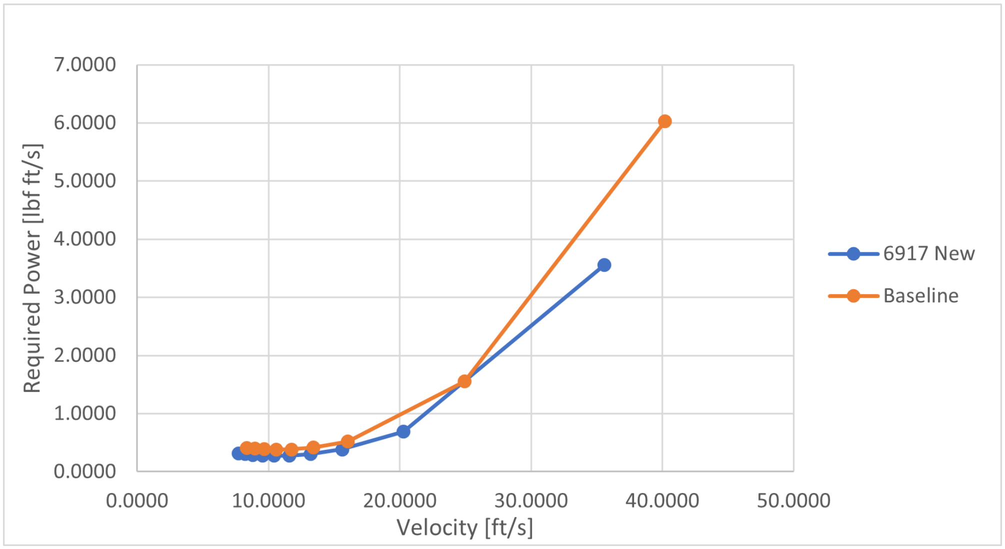


Fig. 7 Required power compared between two glider designs

This figure goes to show the same relationship seen before in the drag force plot. Each aircraft has comparable required power to maintain flight because of comparable drag forces experienced. The new glider design does have a reduced requirement due to the reduced drag on the glider design. Next the aircraft static margin was compared between the glider designs (see fig. 8).

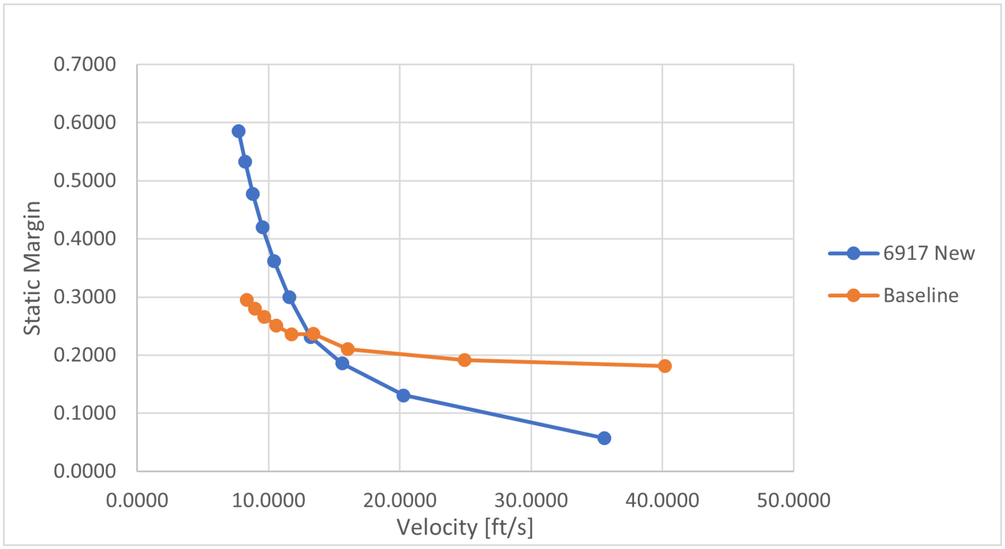


Fig. 8 Static margin compared between two glider designs

The static margin of the new aircraft is considerably higher than the that of the baseline glider, though during higher velocities becomes less. This shows the aircraft is more stable in pitch with an increase in the pitching moment coefficient of the aircraft (see fig. 4). With this increase in static margin, it is expected the aircraft will have a more dominant restoring force when changing angle of attack at low velocities. This restoring force can have a more detrimental affect on the maneuverability of the aircraft, but this will be evaluated later in the report. Focusing on the main objective of these glider designs the step is to evaluate the No wind glider ratio and the sink rate of the aircraft (figs. 9, 10)

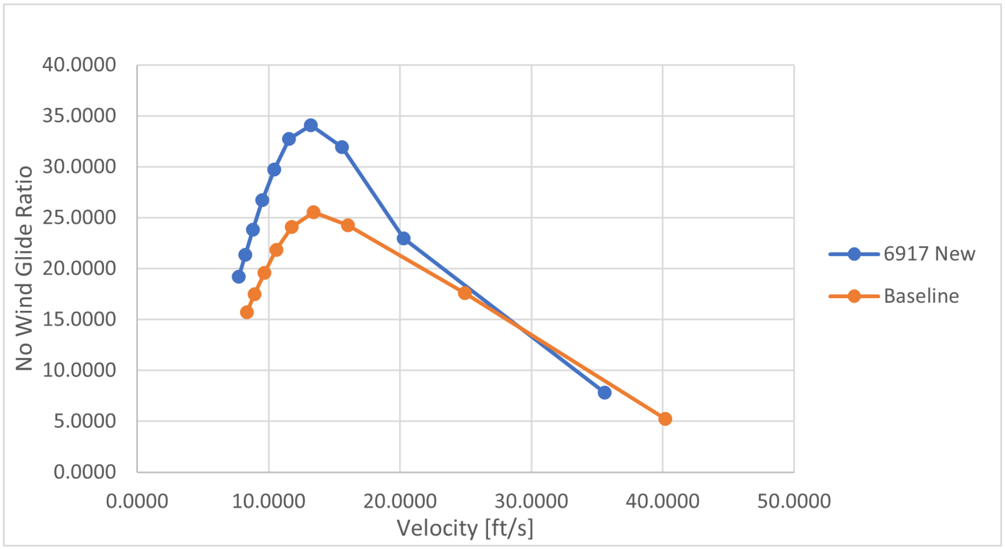


Fig. 9 No wind glide ratio compared between two glider designs

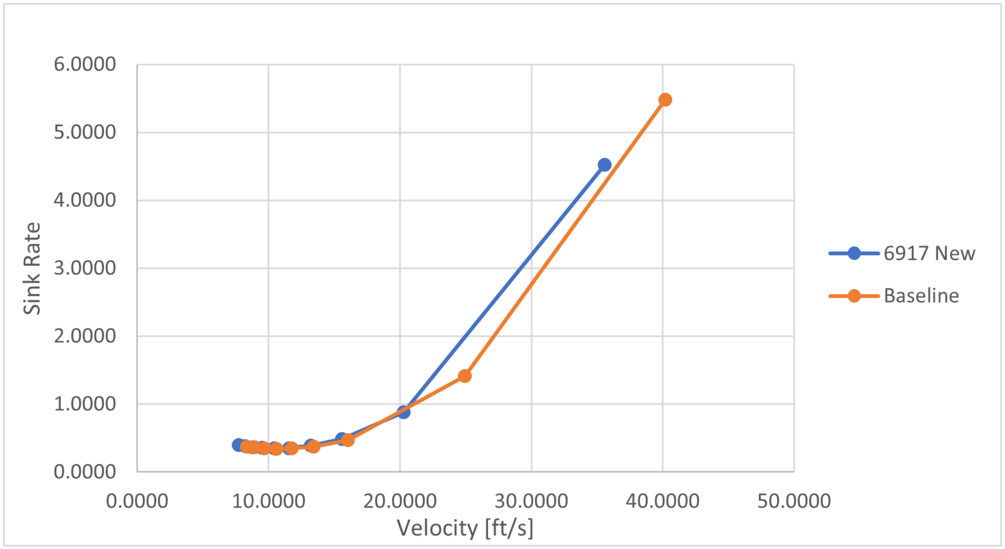


Fig. 10 Sink rate compared between two glider designs

From these results it can be seen the sink rate is comparable between the aircraft with no major performance change between the new and baseline design. The no wind glide ratio does show a higher performance with the new glider configuration instead of the baseline. This is expected due to the increase in the lift to drag ratio of the aircraft. Since the drag between the aircraft designs is comparable it is known the expected lift force is greater in the new glider configuration. Now how do the stability derivatives compare between the two glider configurations. The pitch, roll, and yaw derivatives are compared in figs. 11,12,and 13.

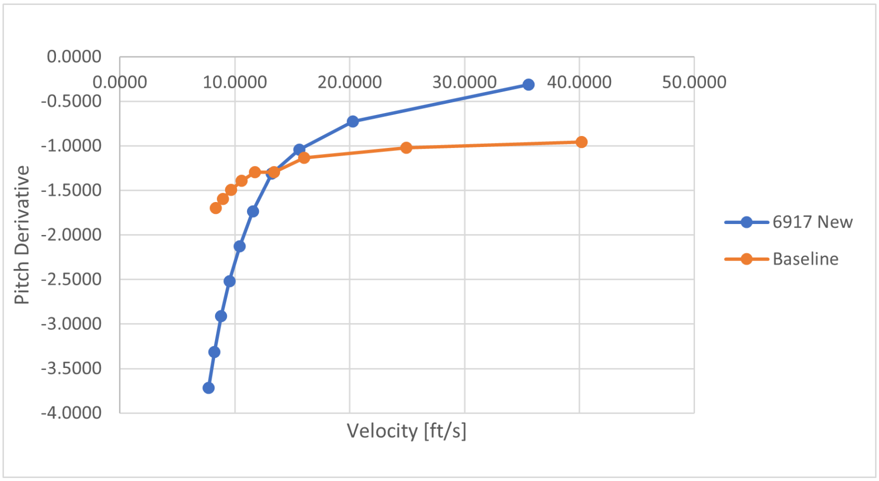


Fig. 11 Pitch stability derivative compared between two glider designs

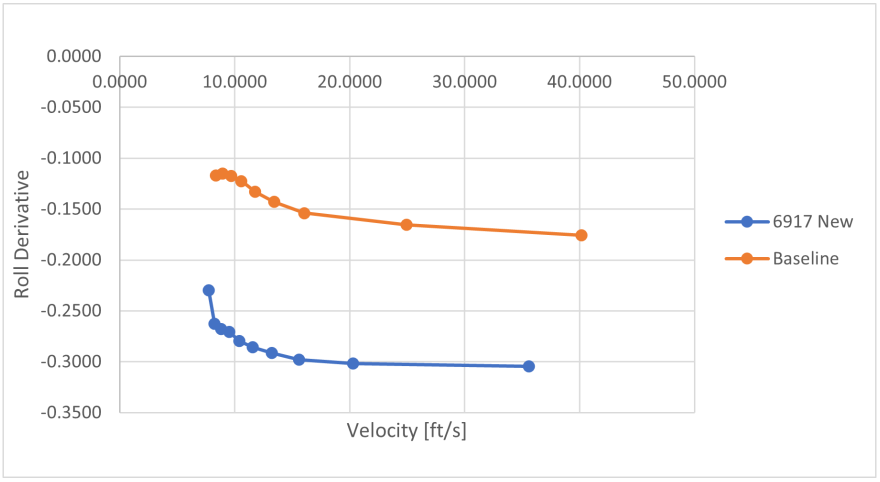


Fig. 12 Roll stability derivative compared between two glider designs

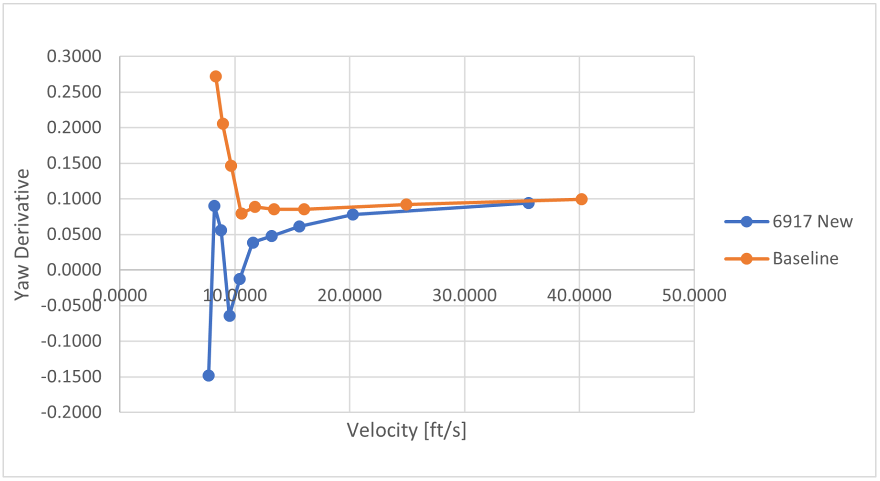


Fig. 13 Yaw stability derivative compared between two glider designs

For the pitch and roll stability it is clear to see both of the glider configurations are stable throughout the range of expected velocities. While the new glider design does have a region of instabilities at low velocities, so we will have a fixed minimum velocity to maintain the stability of the new glider in yaw. Overall the performance of the new glider configuration shown throughout all figures shows an improvement of the glider design compared to the baseline glider. The final comparison for velocity is the locus of aerodynamic centers shown in fig. 14

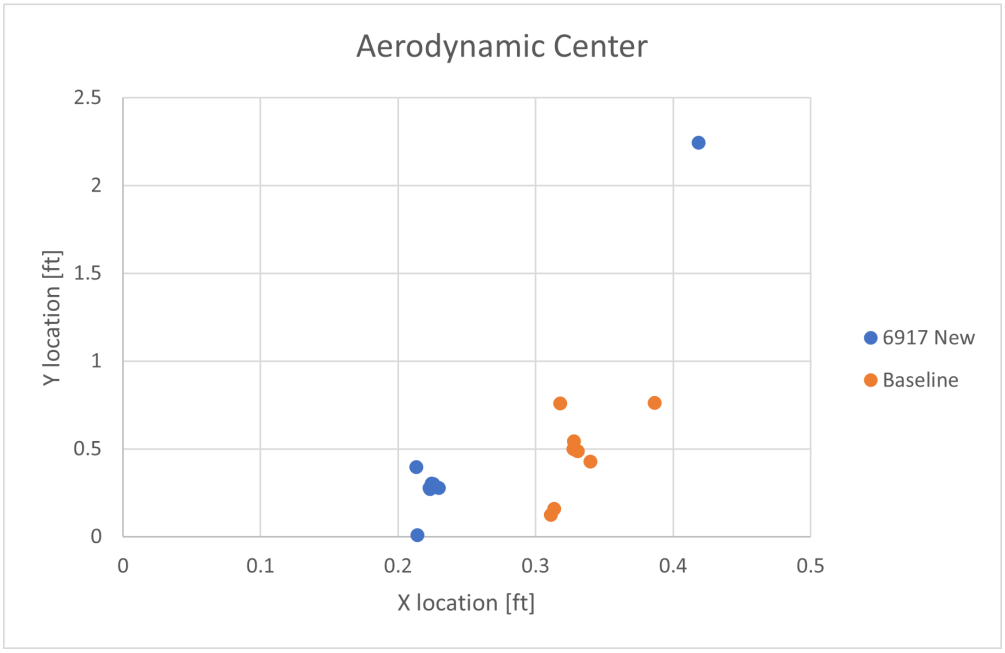


Fig. 14 Locus of aerodynamic centers compared between two glider designs

From these results it can be seen the aerodynamic center shifts closer to the origin (the quarter chord of the main wing) with the new configuration. Continuing the evaluation of the glider configurations this study evaluated the stall speed (V_{min}), minimum drag airspeed (V_{MD}), and the minimum power airspeed (V_{MDV}). The results and the required coefficients to solve for these are shown below in Table 4

Table 4 Velocity evaluations and the required variables

Variable	Baseline	6917 New
ρ_{air} [slug/ft^3]	0.002048	0.002048
W_a [lbf]	1.1017	0.7867
SW [ft^2]	6.75	6.5625
RA [-]	12.00	23.809
CD_0 [-]	0.0138	0.0124
CD_1 [-]	-0.0071	-0.0078
e [-]	0.6854	0.4826
CL_{max} [-]	1.3767	1.4232
V_{min} [ft/s]	10.75	8.738
V_{MDV} [ft/s]	17.52	13.228
V_{MD}[ft/s]	12.64	8.9104

From these results it is clear to see the new glider has a lower stall speed allowing for a slower aircraft without experiencing stall. This result is common among all the velocities calculated and shows the updated aircraft is expected to perform better at lower velocities compared to the baseline. The next stage of the comparison is set between the dimensional eigenvalues of the aircraft. These values determine the dynamic characteristics of the aircraft, and positive real component is an divergent mode, while negative values are convergent modes. The results of this comparison are shown in fig. 15

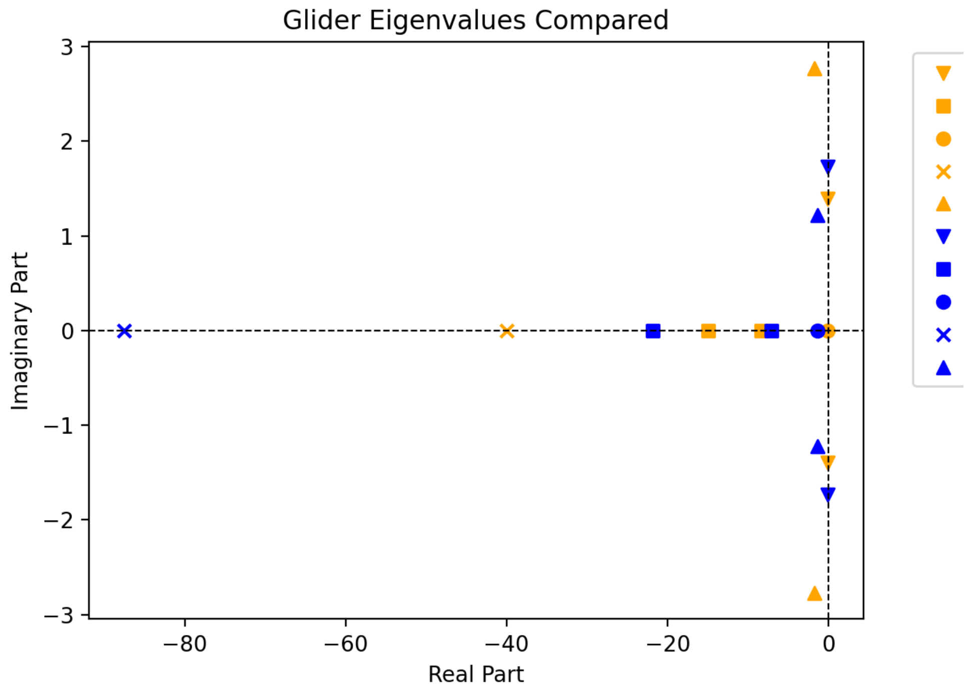


Fig. 15 Dimensional eigenvalues compared between two glider designs

From this comparison it can be seen the new glider has an increase in stability in almost every mode with an increase in the negativity of the values shown. The aircraft demonstrate similar characteristics between the lateral modes, though the new glider has a convergent spiral mode compared to the baseline glider's divergent spiral mode. It is interesting to see the large increase in the new gliders short period value. This shows the aircraft is heavily damped in the short period and this period is not expected to be experienced for long. Using the results of these Eigen values we are able to compute the CAP of the new glider design. This calculation is shown below.

$$CAP = \frac{(\omega_n^2)_{SP}}{CL_\alpha / C_W} = \frac{(9.3785)^2}{6.6085} = 13.3095 \tag{5}$$

With the Eigenvalues known it is easy to calculate the handling qualities of the aircraft for Category B flight phases. The compared performance of these aircraft are shown below in Table 5

Table 5 Category B flight phase level

Mode	Baseline Glider at Trim (V = 15.8573 ft/s)	New Glider at Trim (V = 11.522 ft/s)
Short-Period	3	3
Phugoid	2	2
Roll	1	1
Spiral	3	1
Dutch Roll	1	1

From this result it is shown that the new glider improves the overall spiral effectiveness of the aircraft by creating a converging spiral mode for the aircraft.

VII. Conclusion

Throughout this project much has been learned about the complex nature of aircraft design. It was clear to see from the first design phase it was relatively easy to be able to create a statically stable aircraft, but creating a dynamically stable aircraft is much harder. Attempting to create a stable spiral mode required deeper thought into what components drive spiral stability. Shifting these components then effect the other modes of the aircraft. During the development it was found that there is a competing influence between the phugoid and spiral modes as the dihedral of the aircraft was increased. Instead of being able to fix one mode independent of other modes there needs to be a happy middle ground found. During development of the new glider it was seen that increasing the aspect ratio increased the total expected lift and allowed for longer wings. Decreasing the ballast size allowed for a decrease in the fuselage drag. During the development of this glider there is always ways one could continue to improve the design. The future work would be done to consider the use of a canard configuration to remove the need to carry negative lift from the horizontal stabilizer. In addition, this aircraft would use a V tail configuration to try and reduce the overall needed stabilizer area to allow for more area for the main wing. The v tail configuration is able to control both the lateral and longitudinal dynamics of the aircraft in one stabilizer configuration instead of having two separate pieces. To further improve this aircraft one could remove the ballast altogether

Appendix A: Glider Data File

```
{
      "tag": {},
      "simulation": {
            "constant_density": false,
            "time_step[sec]": 0.05,
            "total_time[sec]": 250,
            "ground_altitude[ft]": 4711
      },
      "aircraft": {
            "wing_area[ft^2]": 6.5625,
            "wing_span[ft]": 12.5,
            "weight[lbf]": 0.7871061038574173,
            "Ixx[slug-ft^2]": 0.0592234767973423,
            "Iyy[slug-ft^2]": 0.06740730255842209,
            "Izz[slug-ft^2]": 0.11730800569057465,
            "Ixy[slug-ft^2]": -5.378096318657965e-18,
            "Ixz[slug-ft^2]": 0.001854020170867443,
            "Iyz[slug-ft^2]": -3.024066013860459e-20,
            "hx[slug-ft^2/s]": 0,
            "hy[slug-ft^2/s]": 0,
            "hz[slug-ft^2/s]": 0,
            "name": "Glidezilla"
      },
      "analysis" : {
            "density[slugs/ft^3]" : 0.002048
      },
      "initial": {
            "airspeed[ft/s]": 13.228,
            "altitude[ft]": 4761,
            "heading[deg]": 0,
            "type": "state",
            "state": {
                  "elevation_angle[deg]": 0,
                  "bank_angle[deg]": 0,
                  "alpha[deg]": 0,
                  "beta[deg]": 0,
                  "p[deg/s]": 0,
                  "q[deg/s]": 0,
                  "r[deg/s]": 0,
                  "aileron[deg]": 0,
                  "elevator[deg]": 0,
                  "rudder[deg]": 0,
                  "throttle": 0
            }
      },
      "aerodynamics": {
            "ground_effect": {
                  "use_ground_effect": true,
                  "taper_ratio": 1
            },
            "gust_magnitude[ft/s]": 2,
            "stall": {
                  "use_stall_model": true,
                  "alpha_blend[deg]": 10.8,
                  "blending_factor": 40
            },
            "CL": {
                  "0": 0.874468159176983,
                  "alpha": 5.77895550222956,
                  "alpha_hat": 0.7557,
                  "qbar": -1.24490516313209,
                  "de": 0
            },
            "CS": {
                "beta": -0.685694937509146,
                "pbar": -0.5694044991615445,
                "Lpbar": -0.06543664639406889,
                "rbar": 0.462982765621289,
                "da": 0,
                "dr": 0
            },
            "CD": {
                "L0": 0.012855925245496318,
                "L": -0.007546501315798773,
                "L2": 0.027579214254294424,
                "S2": 0.5419585059892205,
                "qbar": 0.10231144170040865,
                "Lqbar": 0.5871488192071522,
                "L2qbar": -0.27721609648644296,
                "de": 0,
                "Lde": 0,
                "de2": 0
            },
            "Cl": {
                "beta": -0.285615550230067,
                "pbar": -0.611103727723306,
                "rbar": 0.10251352267177007,
                "Lrbar": 0.222959544927281,
                "da": 0,
                "dr": 0
            },
            "Cm": {
                "0": 0.0000363451326770887,
                "alpha": -1.73455105838682,
                "alpha_hat": -1.9067,
                "qbar": -23.7446343184934,
                "de": 0
            },
            "Cn": {
                "beta": 0.0386849290954465,
                "pbar": 0.06446871725468487,
                "Lpbar": -0.16860800607194723,
                "rbar": -0.0457947904874399,
                "da": 0,
                "Lda": 0,
                "dr": 0
            }
        }
}
```

Appendix B: Dynamic Stability Analysis Code

EigenValue_Project_PartIV

April 12, 2024

[ ]:
```
#import packages
import scipy.linalg as scilin
import numpy as np
import json
import pandas as pd
import os
import matplotlib.pyplot as plt
```

[ ]:
```
#Functions Section

def EigenPrint_TwoMat(A,B):
    C = np.matmul(np.linalg.inv(B),A)
    eigen_vals,eigen_vects = scilin.eig(C)
    id_list = eigen_vals.argsort()[::-1]
    eigen_vals = eigen_vals[id_list]
    eigen_vects = eigen_vects[:,id_list]
    for i in range(len(eigen_vals)):
        print(f'Eigen Value: {eigen_vals[i]}')
    for j in range(len(eigen_vects)):
        print(f'Eigenvector: {eigen_vects[j]}')

def directory_check(directory_name):
    if not os.path.exists(directory_name):
        os.makedirs(directory_name)
        print(f"Directory '{directory_name}' created.")
    else:
        print(f"Directory '{directory_name}' already exists.")
```

[ ]:
```
#Read in the data from the JSON file
filename_load = "6917_new.json"
base_name = os.path.basename(filename_load).split('.')[0]
directory_check(base_name)
Output_File_Long = base_name +'/'+base_name +"_Eigen_Long.txt"
Output_File_Lat = base_name +'/'+base_name +"_Eigen_Lat.txt"
Output_File_Long_mat = base_name +'/'+base_name +'_Matrices_Long.txt'
Output_File_Lat_mat = base_name +'/'+base_name +'_Matrices_Lat.txt'
J_string = open(filename_load).read()
J_vals = json.loads(J_string)
```

```python
#################  Aircraft Properties  ##################################

#Get the moment information of aircraft
I_xx = J_vals['aircraft']['Ixx[slug-ft^2]']
I_yy = J_vals['aircraft']['Iyy[slug-ft^2]']
I_zz = J_vals['aircraft']['Izz[slug-ft^2]']
I_xy = J_vals['aircraft']['Ixy[slug-ft^2]']
I_xz = J_vals['aircraft']['Ixz[slug-ft^2]']

#Name
#Plane_name = J_vals['aircraft']['name']

#Wing Information
Wing_Area = J_vals['aircraft']['wing_area[ft^2]']
Wing_Span = J_vals['aircraft']['wing_span[ft]']
Wing_MeanChord = Wing_Area/Wing_Span

#aircraft weight
Weight = J_vals['aircraft']['weight[lbf]']

#Launch Energy
#Launch_Energy = J_vals['aircraft']['launch_kinetic_energy[ft-lbf]']

#################  Air Properties  ########################################

#density
Air_Density = J_vals['analysis']['density[slugs/ft^3]']

#################  Aerodynamic Properties  #################################

#CL values
CL_0 = J_vals['aerodynamics']['CL']['0']
CL_alpha = J_vals['aerodynamics']['CL']['alpha']
CL_qbar = J_vals['aerodynamics']['CL']['qbar']
CL_alpha_hat = J_vals['aerodynamics']['CL']['alpha_hat']

#CY values
CY_beta = J_vals['aerodynamics']['CS']['beta']
CY_pbar = J_vals['aerodynamics']['CS']['pbar']
CY_rbar = J_vals['aerodynamics']['CS']['rbar']

#CD values
CD_L0 = J_vals['aerodynamics']['CD']['L0']
CD_L1 = J_vals['aerodynamics']['CD']['L']
CD_L2 = J_vals['aerodynamics']['CD']['L2']
CD_0 = CD_L0 + CD_L1*CL_0+CD_L2*CL_0**2
```

```
CD_alpha = CD_L1*CL_alpha + 2*CD_L2*CL_0*CL_alpha
CD_qbar = J_vals['aerodynamics']['CD']['qbar']

#Cl Values
Cl_beta = J_vals['aerodynamics']['Cl']['beta']
Cl_pbar = J_vals['aerodynamics']['Cl']['pbar']
Cl_rbar = J_vals['aerodynamics']['Cl']['rbar']

#Cm Values
Cm_0 = J_vals['aerodynamics']['Cm']['0']
Cm_alpha = J_vals['aerodynamics']['Cm']['alpha']
Cm_qbar = J_vals['aerodynamics']['Cm']['qbar']
Cm_alpha_hat = J_vals['aerodynamics']['Cm']['alpha_hat']

#Cn values
Cn_beta = J_vals['aerodynamics']['Cn']['beta']
Cn_pbar = J_vals['aerodynamics']['Cn']['pbar']
Cn_rbar = J_vals['aerodynamics']['Cn']['rbar']

#################  Solved Variables ###########################################

#Gravity at location
g = 32.17 #gravity at logan elevation used for design

#Solve for the inital velocity given the launch conditions
#V_0 = np.sqrt(2*Launch_Energy*g/Weight) #Need to check if this is correct,␣
↪don't do this, but we can do this later

#Get V_0 from the CL
V_0 = np.sqrt((Weight)/(0.5*Wing_Area*Air_Density*CL_0))
#V_0 = J_vals['initial']['airspeed[ft/s]']
#CW of aircraft
CW = Weight/(0.5*Wing_Area*Air_Density*V_0**2)

print(f'Velocity: {V_0}')
```

```
Directory '6917_new' already exists.
Velocity: 11.573374411266089
```

[ ]:
```
#Solving for all the Base Components found in eqs. 10.70-10.76 to then put into␣
↪the matrices in 10.77 and 10.78, all variable must be unitless

#10.70 -10.72 not required because we don't know these variables and this is a␣
↪glider

#10.73
R_gx = (g*Wing_MeanChord)/(2*V_0**2)
```

```
R_gy = (g*Wing_Span)/(2*V_0**2)
#10.74
R_rhox = (4*Weight/g)/(Air_Density*Wing_Area*Wing_MeanChord)
R_rhoy = (4*Weight/g)/(Air_Density*Wing_Area*Wing_Span)
#10.75
R_xx = (8*I_xx)/(Air_Density*Wing_Area*Wing_Span**3)
R_yy = (8*I_yy)/(Air_Density*Wing_Area*Wing_MeanChord**3)
R_zz = (8*I_zz)/(Air_Density*Wing_Area*Wing_Span**3)
R_xz = (8*I_xz)/(Air_Density*Wing_Area*Wing_Span**3)
#10.76
#Not required since there is no thrust given this aircraft is a glider
```

```
[ ]: #Generating the Matrices, going for the nondimensional form, Longitudinal␣
      ↪Equations
     #10.77 B mat is the LHS matrix, A mat is the RHS matrix
     A_mat_long = np.zeros([6,6])
     B_mat_long = np.identity(6)

     #Variables that could be changed, but assumed to be zero
     CD_mu = 0
     CL_mu_hat = 0
     Cm_mu_hat = 0
     CD_alpha_hat = 0
     alpha_deg = 0
     alpha_rad = alpha_deg*np.pi/180
     #Fill in the A Matrix given what is known
         #Single Row
     A_mat_long[0,0] = -2*CD_0 #+ CT_V*np.cos(alpha_rad)
     A_mat_long[0,1] = CL_0-CD_alpha
     A_mat_long[0,2] = -CD_qbar
     A_mat_long[0,5] = -R_rhox*R_gx*np.cos(alpha_rad)
         #Second Row
     A_mat_long[1,0] = -2*CL_0 #+ CT_V*np.sin(alpha_rad)
     A_mat_long[1,1] = -CL_alpha-CD_0  #Variable is not correct currently, CD_0 is␣
      ↪not correct
     A_mat_long[1,2] = -CL_qbar+R_rhox
     A_mat_long[1,5] = -R_rhox*R_gx*np.sin(alpha_rad)
         #Third Row
     A_mat_long[2,0] = 2*Cm_0
     A_mat_long[2,1] = Cm_alpha
     A_mat_long[2,2] = Cm_qbar
         #Fourth Row
     A_mat_long[3,0] = np.cos(alpha_rad)
     A_mat_long[3,1] = np.sin(alpha_rad)
     A_mat_long[3,5] = -np.sin(alpha_rad)
         #Fifth Row
     A_mat_long[4,0] = -np.sin(alpha_rad)
```

```
A_mat_long[4,1] = np.cos(alpha_rad)
A_mat_long[4,5] = -np.cos(alpha_rad)
    #Sixth Row
A_mat_long[5,2] = 1

#Fill in the B Matrix given what is known
B_mat_long[0,0] = R_rhox+CD_mu
B_mat_long[0,1] = CD_alpha_hat
B_mat_long[1,0] = CL_mu_hat
B_mat_long[1,1] = R_rhox+CL_alpha_hat
B_mat_long[2,0] = -Cm_mu_hat
B_mat_long[2,1] = -Cm_alpha_hat
B_mat_long[2,2] = R_yy

#Create a New matrix using the A and B matrix to do the Eigen value solve

C_mat_long = np.matmul(np.linalg.inv(B_mat_long),A_mat_long)

#Generating the Matrices, going for the nondimensional form, Lateral Equations
#10.77 B mat is the LHS matrix, A mat is the RHS matrix
A_mat_lat = np.zeros([6,6])
B_mat_lat = np.identity(6)

#Variables that could be changed, but assumed to be zero
CD_mu = 0
CL_mu_hat = 0
Cm_mu_hat = 0
CD_alpha_hat = 0
alpha_deg = 0
alpha_rad = alpha_deg*np.pi/180
#Fill in the A Matrix given what is known
    #Single Row
A_mat_lat[0,0] = CY_beta
A_mat_lat[0,1] = CY_pbar
A_mat_lat[0,2] = (CY_rbar-R_rhoy)
A_mat_lat[0,4] = R_rhoy*R_gy*np.cos(alpha_rad)
    #Second Row
A_mat_lat[1,0] = Cl_beta
A_mat_lat[1,1] = Cl_pbar
A_mat_lat[1,2] = Cl_rbar
    #Third Row
A_mat_lat[2,0] = Cn_beta
A_mat_lat[2,1] = Cn_pbar
A_mat_lat[2,2] = Cn_rbar
    #Fourth Row
A_mat_lat[3,0] = 1
A_mat_lat[3,5] = np.cos(alpha_rad)
```

```
    #Fifth Row
A_mat_lat[4,1] = 1
A_mat_lat[4,2] = np.tan(alpha_rad)
    #Sixth Row
A_mat_lat[5,2] = 1/np.cos(alpha_rad)

#Fill in the B Matrix given what is known
B_mat_lat[0,0] = R_rhoy
B_mat_lat[1,1] = R_xx
B_mat_lat[1,2] = -R_xz
B_mat_lat[2,1] = -R_xz
B_mat_lat[2,2] = R_zz

#Create a New matrix using the A and B matrix to do the Eigen value solve

C_mat_lat = np.matmul(np.linalg.inv(B_mat_lat),A_mat_lat)

#pd.DataFrame(C_mat_lat).head(6)
```

```
[ ]: #Getting the actual Eigenvalues and Eigenvectors

eigen_vals_long,eigen_vects_long = scilin.eig(C_mat_long)


# Print eigenvalues and eigenvectors
print("Eigenvalues:")
for i, val in enumerate(eigen_vals_long):
    print(f"Eigenvalue {i + 1}: {val.real:.4f} + {val.imag:.4f}j")

print("\nEigenvectors:")
for i in range(2,len(eigen_vects_long)):
    print(f"Eigenvector {i + 1}:")
    for j in range(len(eigen_vects_long)):
        print(f"   {eigen_vects_long[j, i].real:.4f} + {eigen_vects_long[j, i].
 ↪imag:.4f}j")

        #Getting the actual Eigenvalues and Eigenvectors

eigen_vals_lat,eigen_vects_lat = scilin.eig(C_mat_lat)


# Print eigenvalues and eigenvectors
print("Eigenvalues:")
for i, val in enumerate(eigen_vals_lat):
    print(f"Eigenvalue {i + 1}: {val.real:.4f} + {val.imag:.4f}j")

print("\nEigenvectors:")
```

```
for i in range(2,len(eigen_vects_lat)):
    print(f"Eigenvector {i + 1}:")
    for j in range(len(eigen_vects_lat)):
        print(f"   {eigen_vects_lat[j, i].real:.4f} + {eigen_vects_lat[j, i].
 ↪imag:.4f}j")
```

```
Eigenvalues:
Eigenvalue 1: 0.0000 + 0.0000j
Eigenvalue 2: 0.0000 + 0.0000j
Eigenvalue 3: -0.3674 + 0.0000j
Eigenvalue 4: -0.1223 + 0.0000j
Eigenvalue 5: -0.0019 + 0.0323j
Eigenvalue 6: -0.0019 + -0.0323j

Eigenvectors:
Eigenvector 3:
   -0.0440 + 0.0000j
   0.3315 + 0.0000j
   0.0044 + 0.0000j
   0.1198 + 0.0000j
   -0.9347 + 0.0000j
   -0.0119 + 0.0000j
Eigenvector 4:
   0.0512 + 0.0000j
   -0.0501 + 0.0000j
   -0.0074 + 0.0000j
   -0.4186 + 0.0000j
   0.9033 + 0.0000j
   0.0604 + 0.0000j
Eigenvector 5:
   0.0017 + -0.0287j
   -0.0003 + 0.0074j
   -0.0001 + -0.0005j
   -0.8886 + 0.0000j
   0.0318 + -0.4563j
   -0.0150 + 0.0055j
Eigenvector 6:
   0.0017 + 0.0287j
   -0.0003 + -0.0074j
   -0.0001 + 0.0005j
   -0.8886 + -0.0000j
   0.0318 + 0.4563j
   -0.0150 + -0.0055j
Eigenvalues:
Eigenvalue 1: 0.0000 + 0.0000j
Eigenvalue 2: 0.0000 + 0.0000j
Eigenvalue 3: -34.5173 + 0.0000j
Eigenvalue 4: -0.8185 + 0.0000j
Eigenvalue 5: -0.4256 + 0.7548j
Eigenvalue 6: -0.4256 + -0.7548j

Eigenvectors:
Eigenvector 3:
   -0.0303 + 0.0000j
   -0.9984 + 0.0000j
   0.0388 + 0.0000j
   0.0009 + 0.0000j
   0.0289 + 0.0000j
   -0.0011 + 0.0000j
Eigenvector 4:
   -0.3738 + 0.0000j
   0.0869 + 0.0000j
   -0.5380 + 0.0000j
   -0.3464 + 0.0000j
   -0.1062 + 0.0000j
   0.6573 + 0.0000j
Eigenvector 5:
   0.6593 + 0.0000j
   -0.2947 + -0.0147j
   0.1036 + -0.1264j
   -0.3011 + -0.4575j
   0.1523 + 0.3046j
   -0.1858 + -0.0325j
Eigenvector 6:
   0.6593 + -0.0000j
   -0.2947 + 0.0147j
   0.1036 + 0.1264j
   -0.3011 + 0.4575j
   0.1523 + -0.3046j
   -0.1858 + 0.0325j
```

[ ]:
```
#Get the amplitude and phase of each component as a numpy array from the␣
 ↪eigenvectors

#long amp and phase matrices
amplitude_long = np.zeros_like(eigen_vects_long)
phase_long = np.zeros_like(eigen_vects_long)
#loop through columns
for j in range(eigen_vects_long.shape[0]):
     #loop through rows
     for i in range(eigen_vects_long.shape[1]):
          amplitude_long[i,j] = np.sqrt(eigen_vects_long[i,j].real**2 +␣
 ↪eigen_vects_long[i,j].imag**2)
          phase_long[i,j] = np.arctan2(eigen_vects_long[i,j].
 ↪imag,eigen_vects_long[i,j].real)*(180/np.pi)
```

```
#Lateral amp and phase matrices
amplitude_lat = np.zeros_like(eigen_vects_lat)
phase_lat = np.zeros_like(eigen_vects_lat)
#loop through columns
for j in range(eigen_vects_lat.shape[0]):
     #loop through rows
     for i in range(eigen_vects_lat.shape[1]):
          amplitude_lat[i,j] = np.sqrt(eigen_vects_lat[i,j].real**2 +␣
 ↪eigen_vects_lat[i,j].imag**2)
          phase_lat[i,j] = np.arctan2(eigen_vects_lat[i,j].
 ↪imag,eigen_vects_lat[i,j].real)*(180/np.pi)
```

[ ]:

```
#Going through the damping rate

variable_symbols_lat = ['Δ ', 'Δp_bar', 'Δr_bar', 'Δ _y', 'ΔΦ', 'Δ ']
variable_symbols_long = ['Δ ', 'Δ ', 'Δq_bar', 'Δ _x', 'Δ _z', 'ΔΘ']


for z in range(len(eigen_vals_long)):
    i = eigen_vals_long[z]
   ␣
 ↪print(f'--------------------------------------------------------------------------------')
    print(f'Dimensionless Eigen Value: {i.real:8.6f}+{i.imag:12.8f}j')
    sigma = -i.real
    #Rigid Body mode
    if i.real == 0:
        print('\t Rigid Body Mode: Eigen Value: 0 ')
        print('\t No analysis required currently\n')
    #Convergent Modes Sigma > 0
    if sigma > 0:
        #Damping rate
        Damp_rate = sigma*2*V_0/Wing_MeanChord
        #99 Damping Time
        Damp_time_99 = np.log(0.01)/-Damp_rate
        #Damped natural frequency and Period
        if i.imag != 0:
            W_d = abs(i.imag)*2*V_0/Wing_MeanChord
            Period = (2*np.pi)/W_d
            #Damping Ratio
            Damp_Ratio =   -i.real/(np.sqrt(i.real**2 + i.imag**2))

        print(f'\t Damping Rate [1/s]: {Damp_rate:12.6f}')
        print(f'\t 99% Damping Time [s]: {Damp_time_99:12.6f}')
        if i.imag != 0:
            print(f'\t Damped Nat Freq: {W_d:12.6f}')
            print(f'\t Period: {Period:12.6f}')
            print(f'\t Damping Ratio: {Damp_Ratio:12.6f}')
        print('')
    #Divergent Modes Sigma < 0
    if sigma < 0:
    #Damping rate
        Damp_rate = sigma*2*V_0/Wing_MeanChord
        #99 Damping Time
        Double_time = np.log(2.00)/-Damp_rate
        #Damped natural frequency and Period
        if i.imag != 0:
            W_d = abs(i.imag)*2*V_0/Wing_MeanChord
            Period = (2*np.pi)/W_d
            #Damping Ratio
            Damp_Ratio =   -i.real/(np.sqrt(i.real**2 + i.imag**2))

        print(f'\t Damping Rate [1/s]: {Damp_rate:12.6f}')
        print(f'\t Doubling Time [s]: {Double_time:12.6f}')
        if i.imag != 0:
            print(f'\t Damped Nat Freq: {W_d:12.6f}')
            print(f'\t Period: {Period:12.6f}')
            print(f'\t Damping Ratio: {Damp_Ratio:12.6f}')
        print('')

    #Print the Eigen Vector
    ␣
↪print(f'--------------------------------------------------------------------------------')
    print(f'{"Variable":<15} {"Real Part":<15} {"Imaginary Part":<20}␣
↪{"Amplitude":<15} {"Phase":<15}')
    eigen_vects_long = np.asarray(eigen_vects_long)
    for j in range(6):
        real = eigen_vects_long[j,z].real
        imag = eigen_vects_long[j,z].imag
        amp = amplitude_long[j,z].real
        phase_deg = phase_long[j,z].real
        symbol = variable_symbols_long[j]
        print(f'{symbol:<15} {real:<15.6f} {imag:<20.6f} {amp:<15.6f}␣
↪{phase_deg:<15.6f}')
    ␣
↪print(f'--------------------------------------------------------------------------------')
```

```
--------------------------------------------------------------------------------
Dimensionless Eigen Value: 0.000000+  0.00000000j
         Rigid Body Mode: Eigen Value: 0
         No analysis required currently

--------------------------------------------------------------------------------
Variable        Real Part       Imaginary Part       Amplitude       Phase
```

```
Δ               0.000000        0.000000            0.000000        0.000000
Δ               0.000000        0.000000            0.000000        0.000000
Δq_bar           0.000000        0.000000            0.000000        0.000000
Δ _x            1.000000        0.000000            1.000000        0.000000
Δ _z            0.000000        0.000000            0.000000        0.000000
ΔΘ               0.000000        0.000000            0.000000        0.000000
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
Dimensionless Eigen Value: 0.000000+  0.00000000j
         Rigid Body Mode: Eigen Value: 0
         No analysis required currently

-------------------------------------------------------------------------------
Variable        Real Part       Imaginary Part      Amplitude       Phase
Δ               0.000000        0.000000            0.000000        0.000000
Δ               0.000000        0.000000            0.000000        0.000000
Δq_bar           0.000000        0.000000            0.000000        0.000000
Δ _x            0.000000        0.000000            0.000000        0.000000
Δ _z            1.000000        0.000000            1.000000        0.000000
ΔΘ               0.000000        0.000000            0.000000        0.000000
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
Dimensionless Eigen Value: -0.367433+  0.00000000j
         Damping Rate [1/s]:    16.199771
         99% Damping Time [s]:     0.284274

-------------------------------------------------------------------------------
Variable        Real Part       Imaginary Part      Amplitude       Phase
Δ               -0.044019       0.000000            0.044019        180.000000
Δ               0.331499        0.000000            0.331499        0.000000
Δq_bar           0.004387        0.000000            0.004387        0.000000
Δ _x            0.119802        0.000000            0.119802        0.000000
Δ _z            -0.934696       0.000000            0.934696        180.000000
ΔΘ               -0.011939       0.000000            0.011939        180.000000
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
Dimensionless Eigen Value: -0.122344+  0.00000000j
         Damping Rate [1/s]:     5.394041
         99% Damping Time [s]:     0.853751

-------------------------------------------------------------------------------
Variable        Real Part       Imaginary Part      Amplitude       Phase
Δ               0.051212        0.000000            0.051212        0.000000
Δ               -0.050107       0.000000            0.050107        180.000000
Δq_bar           -0.007390       0.000000            0.007390        180.000000
Δ _x            -0.418588       0.000000            0.418588        180.000000
Δ _z            0.903298        0.000000            0.903298        0.000000
ΔΘ               0.060407        0.000000            0.060407        0.000000
```

```
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
Dimensionless Eigen Value: -0.001947+  0.03233351j
         Damping Rate [1/s]:     0.085822
         99% Damping Time [s]:    53.659487
         Damped Nat Freq:     1.425554
         Period:     4.407540
         Damping Ratio:     0.060094

--------------------------------------------------------------------------------
Variable        Real Part       Imaginary Part      Amplitude       Phase
Δ               0.001730        -0.028733           0.028785        -86.554800
Δ               -0.000306       0.007426            0.007432        92.355894
Δq_bar           -0.000149       -0.000496           0.000518        -106.728901
Δ _x            -0.888634       0.000000            0.888634        180.000000
Δ _z            0.031786        -0.456267           0.457372        -86.014944
ΔΘ               -0.014996       0.005510            0.015977        159.825900
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
Dimensionless Eigen Value: -0.001947+ -0.03233351j
         Damping Rate [1/s]:     0.085822
         99% Damping Time [s]:    53.659487
         Damped Nat Freq:     1.425554
         Period:     4.407540
         Damping Ratio:     0.060094

--------------------------------------------------------------------------------
Variable        Real Part       Imaginary Part      Amplitude       Phase
Δ               0.001730        0.028733            0.028785        86.554800
Δ               -0.000306       -0.007426           0.007432        -92.355894
Δq_bar           -0.000149       0.000496            0.000518        106.728901
Δ _x            -0.888634       -0.000000           0.888634        -180.000000
Δ _z            0.031786        0.456267            0.457372        86.014944
ΔΘ               -0.014996       -0.005510           0.015977        -159.825900
--------------------------------------------------------------------------------
```

[ ]:
```python
#Going through the damping rate for lateral mode

for z in range(len(eigen_vals_lat)):
    i = eigen_vals_lat[z]
    ␣
 ↪print(f'--------------------------------------------------------------------------------')
    print(f'Dimensionless Eigen Value: {i.real:8.6f}+{i.imag:12.8f}j')
    sigma = -i.real
    #Rigid Body mode
    if i.real == 0:
        print('\t Rigid Body Mode: Eigen Value: 0 ')
        print('\t No analysis required currently\n')
    #Convergent Modes Sigma > 0
    if sigma > 0:
        #Damping rate
        Damp_rate = sigma*2*V_0/Wing_Span
        #99 Damping Time
        Damp_time_99 = np.log(0.01)/-Damp_rate
        #Damped natural frequency and Period
        if i.imag != 0:
            W_d = abs(i.imag)*2*V_0/Wing_Span
            Period = (2*np.pi)/W_d
            #Damping Ratio
            Damp_Ratio =   -i.real/(np.sqrt(i.real**2 + i.imag**2))

        print(f'\t Damping Rate [1/s]: {Damp_rate:12.6f}')
        print(f'\t 99% Damping Time [s]: {Damp_time_99:12.6f}')
        if i.imag != 0:
            print(f'\t Damped Nat Freq: {W_d:12.6f}')
            print(f'\t Period: {Period:12.6f}')
            print(f'\t Damping Ratio: {Damp_Ratio:12.6f}')
        print('')
    #Divergent Modes Sigma < 0
    if sigma < 0:
    #Damping rate
        Damp_rate = sigma*2*V_0/Wing_Span
        #99 Damping Time
        Double_time = np.log(2.00)/-Damp_rate
        #Damped natural frequency and Period
        if i.imag != 0:
            W_d = abs(i.imag)*2*V_0/Wing_Span
            Period = (2*np.pi)/W_d
            #Damping Ratio
            Damp_Ratio =   -i.real/(np.sqrt(i.real**2 + i.imag**2))

        print(f'\t Damping Rate [1/s]: {Damp_rate:12.6f}')
        print(f'\t Doubling Time [s]: {Double_time:12.6f}')
        if i.imag != 0:
            print(f'\t Damped Nat Freq: {W_d:12.6f}')
            print(f'\t Period: {Period:12.6f}')
            print(f'\t Damping Ratio: {Damp_Ratio:12.6f}')
        print('')

    #Print the Eigen Vector
  ␣
↪print(f'--------------------------------------------------------------------------------')
    print(f'{"Variable":<15} {"Real Part":<15} {"Imaginary Part":<20}␣
↪{"Amplitude":<15} {"Phase":<15}')
```

```
    eigen_vects_lat = np.asarray(eigen_vects_lat)
    for j in range(6):
        real = eigen_vects_lat[j,z].real
        imag = eigen_vects_lat[j,z].imag
        amp = amplitude_lat[j,z].real
        phase_deg = phase_lat[j,z].real
        symbol = variable_symbols_lat[j]
        print(f'{symbol:<15} {real:<15.6f} {imag:<20.6f} {amp:<15.6f}␣
↪{phase_deg:<15.6f}')
    ␣
↪print(f'--------------------------------------------------------------------------------')
```

```
--------------------------------------------------------------------------------
Dimensionless Eigen Value: 0.000000+  0.00000000j
         Rigid Body Mode: Eigen Value: 0
         No analysis required currently

--------------------------------------------------------------------------------
Variable        Real Part       Imaginary Part       Amplitude       Phase
Δ              0.000000        0.000000            0.000000        0.000000
Δp_bar          0.000000        0.000000            0.000000        0.000000
Δr_bar          0.000000        0.000000            0.000000        0.000000
Δ _y           1.000000        0.000000            1.000000        0.000000
ΔΦ              0.000000        0.000000            0.000000        0.000000
Δ              0.000000        0.000000            0.000000        0.000000
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
Dimensionless Eigen Value: 0.000000+  0.00000000j
         Rigid Body Mode: Eigen Value: 0
         No analysis required currently

--------------------------------------------------------------------------------
Variable        Real Part       Imaginary Part       Amplitude       Phase
Δ              0.000000        0.000000            0.000000        0.000000
Δp_bar          0.000000        0.000000            0.000000        0.000000
Δr_bar          0.000000        0.000000            0.000000        0.000000
Δ _y           -1.000000       0.000000            1.000000        180.000000
ΔΦ              0.000000        0.000000            0.000000        0.000000
Δ              0.000000        0.000000            0.000000        0.000000
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
Dimensionless Eigen Value: -34.517298+  0.00000000j
         Damping Rate [1/s]:    63.917059
         99% Damping Time [s]:     0.072049
```

```
--------------------------------------------------------------------------------
Variable         Real Part        Imaginary Part       Amplitude        Phase
Δ               -0.030333        0.000000             0.030333         180.000000
Δp_bar           -0.998368        0.000000             0.998368         180.000000
Δr_bar           0.038768         0.000000             0.038768         0.000000
Δ _y            0.000911         0.000000             0.000911         0.000000
ΔΦ               0.028924         0.000000             0.028924         0.000000
Δ               -0.001123        0.000000             0.001123         180.000000
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
Dimensionless Eigen Value: -0.818540+  0.00000000j
         Damping Rate [1/s]:     1.515724
         99% Damping Time [s]:     3.038265

--------------------------------------------------------------------------------
Variable         Real Part        Imaginary Part       Amplitude        Phase
Δ               -0.373762        0.000000             0.373762         180.000000
Δp_bar           0.086944         0.000000             0.086944         0.000000
Δr_bar           -0.538015        0.000000             0.538015         180.000000
Δ _y            -0.346377        0.000000             0.346377         180.000000
ΔΦ               -0.106218        0.000000             0.106218         180.000000
Δ               0.657285         0.000000             0.657285         0.000000
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
Dimensionless Eigen Value: -0.425593+  0.75475346j
         Damping Rate [1/s]:     0.788088
         99% Damping Time [s]:     5.843473
         Damped Nat Freq:     1.397607
         Period:     4.495674
         Damping Ratio:     0.491177

--------------------------------------------------------------------------------
Variable         Real Part        Imaginary Part       Amplitude        Phase
Δ               0.659280         0.000000             0.659280         0.000000
Δp_bar           -0.294734        -0.014701            0.295100         -177.144517
Δr_bar           0.103626         -0.126428            0.163470         -50.660266
Δ _y            -0.301057        -0.457519            0.547685         -123.345749
ΔΦ               0.152296         0.304626             0.340575         63.437539
Δ               -0.185839        -0.032507            0.188660         -170.078211
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
Dimensionless Eigen Value: -0.425593+ -0.75475346j
         Damping Rate [1/s]:     0.788088
         99% Damping Time [s]:     5.843473
         Damped Nat Freq:     1.397607
         Period:     4.495674
         Damping Ratio:     0.491177
```

```
--------------------------------------------------------------------------------
Variable          Real Part        Imaginary Part       Amplitude        Phase
Δ                0.659280         -0.000000            0.659280         -0.000000
Δp_bar            -0.294734        0.014701             0.295100         177.144517
Δr_bar            0.103626         0.126428             0.163470         50.660266
Δ _y             -0.301057        0.457519             0.547685         123.345749
ΔΦ                0.152296         -0.304626            0.340575         -63.437539
Δ                -0.185839        0.032507             0.188660         170.078211
--------------------------------------------------------------------------------
```

[ ]:
```python
#Longitude write file to txt to use

def write_info_file_long(output_file, eigen_vals, eigen_vects, amplitude,␣
 ↪phase, V_0, Wing_MeanChord):
    # Define the variable symbols list manually
    variable_symbols_long = ['Δ ', 'Δ ', 'Δq_bar', 'Δ _x', 'Δ _z', 'Δθ']

    # Open the output file for writing with UTF-8 encoding
    with open(output_file, 'w', encoding='utf-8') as file:
        file.
 ↪write('--------------------------------------------------------------------------------\n')
        file.write('Linearized Longitudinal Equation Results \n')
        for z in range(len(eigen_vals)):
            i = eigen_vals[z]
            file.
 ↪write('--------------------------------------------------------------------------------\n')
            file.write(f'Dimensionless Eigen Value: {i.real:8.6f}+{i.imag:12.
 ↪8f}j\n')
            sigma = -i.real
            # Rigid Body mode
            if i.real == 0:
                file.write('\t Rigid Body Mode: Eigen Value: 0 \n')
                file.write('\t No analysis required currently\n\n')
            # Convergent Modes Sigma > 0
            if sigma > 0:
                # Damping rate
                Damp_rate = sigma*2*V_0/Wing_MeanChord
                # 99 Damping Time
                Damp_time_99 = np.log(0.01)/-Damp_rate
                # Damped natural frequency and Period
                if i.imag != 0:
                    W_d = abs(i.imag)*2*V_0/Wing_MeanChord
                    Period = (2*np.pi)/W_d
                    #Damping Ratio
                    Damp_Ratio =   -i.real/(np.sqrt(i.real**2 + i.imag**2))
                file.write(f'\t Damping Rate [1/s]: {Damp_rate:12.6f}\n')
                file.write(f'\t 99% Damping Time [s]: {Damp_time_99:12.6f}\n')
                if i.imag != 0:
                    file.write(f'\t Damped Nat Freq: {W_d:12.6f}\n')
                    file.write(f'\t Period: {Period:12.6f}\n')
                    file.write(f'\t Damping Ratio: {Damp_Ratio:12.6f}\n\n')
            # Divergent Modes Sigma < 0
            if sigma < 0:
                # Damping rate
                Damp_rate = sigma*2*V_0/Wing_MeanChord
                # 99 Damping Time
                Double_time = np.log(2.00)/-Damp_rate
                # Damped natural frequency and Period
                if i.imag != 0:
                    W_d = abs(i.imag)*2*V_0/Wing_MeanChord
                    Period = (2*np.pi)/W_d
                    #Damping Ratio
                    Damp_Ratio =   -i.real/(np.sqrt(i.real**2 + i.imag**2))
                file.write(f'\t Damping Rate [1/s]: {Damp_rate:12.6f}\n')
                file.write(f'\t Doubling Time [s]: {Double_time:12.6f}\n')
                if i.imag != 0:
                    file.write(f'\t Damped Nat Freq: {W_d:12.6f}\n')
                    file.write(f'\t Period: {Period:12.6f}\n')
                    file.write(f'\t Damping Ratio: {Damp_Ratio:12.6f}\n\n')
            # Print the Eigen Vector
            file.
 ↪write('--------------------------------------------------------------------------------\n')
            file.write(f'{"Variable":<15} {"Real Part":<15} {"Imaginary Part":
 ↪<20} {"Amplitude":<15} {"Phase":<15}\n')
            eigen_vects = np.asarray(eigen_vects)
            for j in range(6):
                real = eigen_vects[j,z].real
                imag = eigen_vects[j,z].imag
                amp = amplitude[j,z].real
                phase_deg = phase[j,z].real
                symbol = variable_symbols_long[j]
                file.write(f'{symbol:<15} {real:<15.6f} {imag:<20.6f} {amp:<15.
 ↪6f} {phase_deg:<15.6f}\n')
            file.
 ↪write('--------------------------------------------------------------------------------\n')


#Lateral write file to txt to use

def write_info_file_lat(output_file, eigen_vals, eigen_vects, amplitude, phase,␣
 ↪V_0, Wing_Span):
    # Define the variable symbols list manually
    variable_symbols_long = ['Δ ', 'Δ ', 'Δq_bar', 'Δ _x', 'Δ _z', 'Δθ']
    # Open the output file for writing with UTF-8 encoding
    with open(output_file, 'w', encoding='utf-8') as file:
        file.
↪write('--------------------------------------------------------------------------------\n')
        file.write('Linearized Lateral Equation Results \n')
        for z in range(len(eigen_vals)):
            i = eigen_vals[z]
            file.
↪write('--------------------------------------------------------------------------------\n')
            file.write(f'Dimensionless Eigen Value: {i.real:8.6f}+{i.imag:12.
↪8f}j\n')
            sigma = -i.real
            # Rigid Body mode
            if i.real == 0:
                file.write('\t Rigid Body Mode: Eigen Value: 0 \n')
                file.write('\t No analysis required currently\n\n')
            # Convergent Modes Sigma > 0
            if sigma > 0:
                # Damping rate
                Damp_rate = sigma*2*V_0/Wing_Span
                # 99 Damping Time
                Damp_time_99 = np.log(0.01)/-Damp_rate
                # Damped natural frequency and Period
                if i.imag != 0:
                    W_d = abs(i.imag)*2*V_0/Wing_Span
                    Period = (2*np.pi)/W_d
                    #Damping Ratio
                    Damp_Ratio =   -i.real/(np.sqrt(i.real**2 + i.imag**2))
                file.write(f'\t Damping Rate [1/s]: {Damp_rate:12.6f}\n')
                file.write(f'\t 99% Damping Time [s]: {Damp_time_99:12.6f}\n')
                if i.imag != 0:
                    file.write(f'\t Damped Nat Freq: {W_d:12.6f}\n')
                    file.write(f'\t Period: {Period:12.6f}\n')
                    file.write(f'\t Damping Ratio: {Damp_Ratio:12.6f}\n\n')
            # Divergent Modes Sigma < 0
            if sigma < 0:
                # Damping rate
                Damp_rate = sigma*2*V_0/Wing_Span
                # 99 Damping Time
                Double_time = np.log(2.00)/-Damp_rate
                # Damped natural frequency and Period
                if i.imag != 0:
                    W_d = abs(i.imag)*2*V_0/Wing_Span
                    Period = (2*np.pi)/W_d
                    #Damping Ratio
                    Damp_Ratio =   -i.real/(np.sqrt(i.real**2 + i.imag**2))
                file.write(f'\t Damping Rate [1/s]: {Damp_rate:12.6f}\n')
                file.write(f'\t Doubling Time [s]: {Double_time:12.6f}\n')
                if i.imag != 0:
                    file.write(f'\t Damped Nat Freq: {W_d:12.6f}\n')
                    file.write(f'\t Period: {Period:12.6f}\n')
                    file.write(f'\t Damping Ratio: {Damp_Ratio:12.6f}\n\n')

            # Print the Eigen Vector
            file.
↪write('--------------------------------------------------------------------------------\n')
            file.write(f'{"Variable":<15} {"Real Part":<15} {"Imaginary Part":
↪<20} {"Amplitude":<15} {"Phase":<15}\n')
            eigen_vects = np.asarray(eigen_vects)
            for j in range(6):
                real = eigen_vects[j,z].real
                imag = eigen_vects[j,z].imag
                amp = amplitude[j,z].real
                phase_deg = phase[j,z].real
                symbol = variable_symbols_lat[j]
                file.write(f'{symbol:<15} {real:<15.6f} {imag:<20.6f} {amp:<15.
↪6f} {phase_deg:<15.6f}\n')
            file.
↪write('--------------------------------------------------------------------------------\n')
```

[ ]:
```
# Write the results to a file
write_info_file_lat(Output_File_Lat,eigen_vals_lat,eigen_vects_lat,amplitude_lat,phase_lat,V_C
write_info_file_long(Output_File_Long,eigen_vals_long,eigen_vects_long,amplitude_long,phase_lo
```

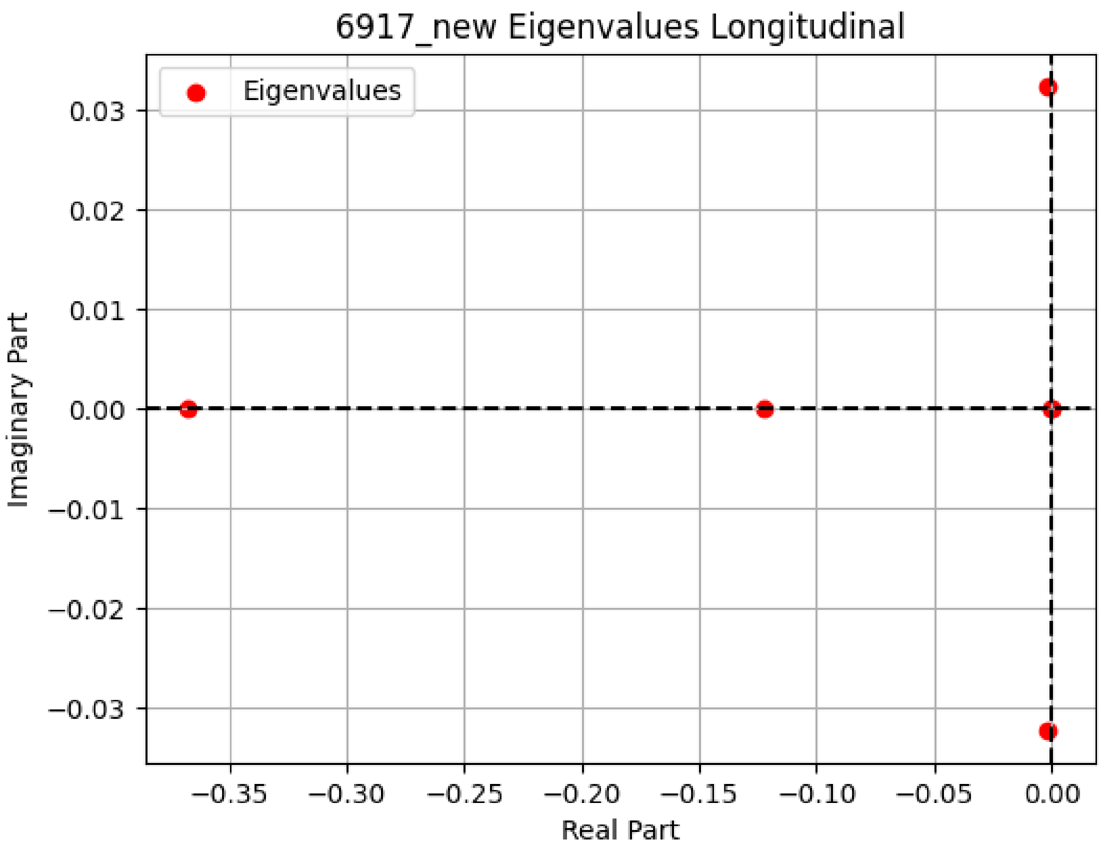
[ ]:
```
def mat_to_file(matrix_A, matrix_B, matrix_C, file_name, Mode_type=str):
    with open(file_name, 'w') as file:
        file.
↪write('----------------------------------------------------------\n')
        file.write(f'{Mode_type} A and B matrix Results:\n')
        file.
↪write('----------------------------------------------------------\n')
        file.write("Matrix A:\n")
        file.
↪write('----------------------------------------------------------\n')
        for row in matrix_A:
            formatted_row = ["{:12.7f}".format(element) for element in row]
            file.write(" ".join(formatted_row) + '\n')

        file.
↪write('----------------------------------------------------------\n')
        file.write("Matrix B:\n")
        file.
↪write('----------------------------------------------------------\n')
```

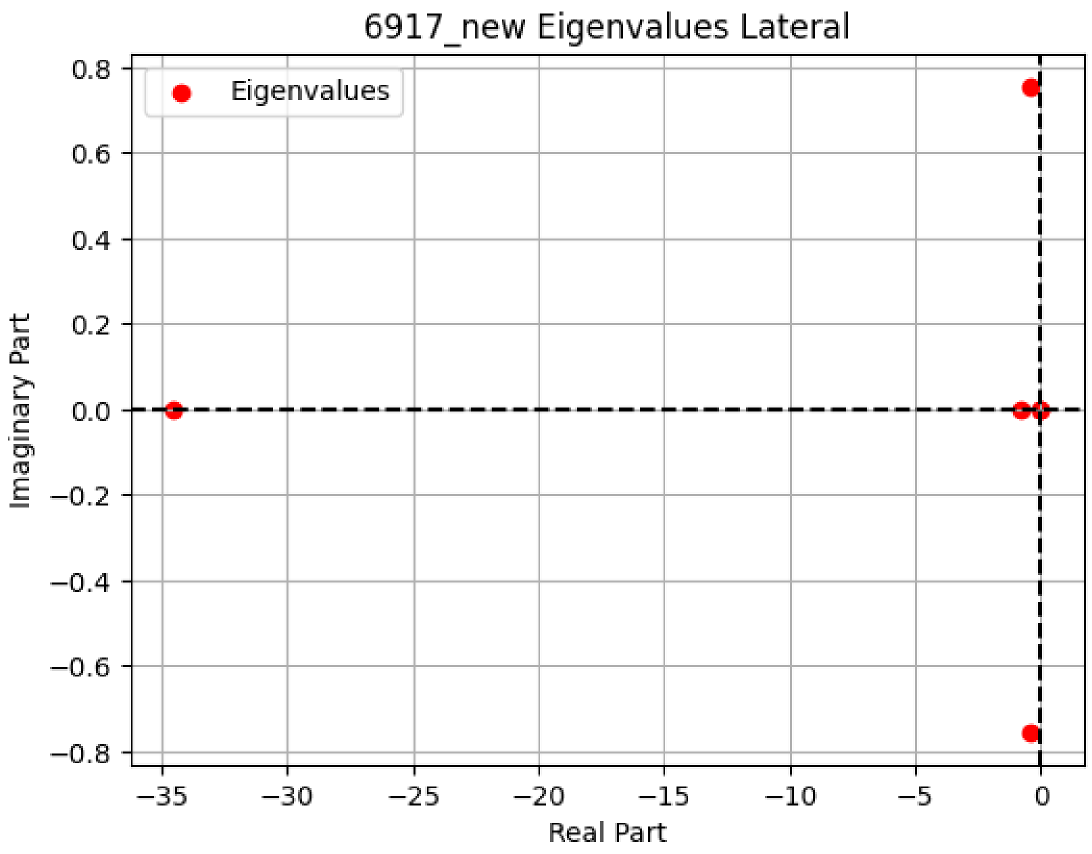
```
        for row in matrix_B:
            formatted_row = ["{:12.7f}".format(element) for element in row]
            file.write(" ".join(formatted_row) + '\n')

        file.
↪write('---------------------------------------------------------\n')
        file.write("Matrix C:\n")
        file.
↪write('---------------------------------------------------------\n')
        for row in matrix_C:
            formatted_row = ["{:12.7f}".format(element) for element in row]
            file.write(" ".join(formatted_row) + '\n')
```

[ ]:
```
#Write the matrices to a file, long is one file lat is another file should have␣
↪the long matrices

mat_to_file(A_mat_long,B_mat_long,C_mat_long,Output_File_Long_mat,"Longitudinal")
mat_to_file(A_mat_lat,B_mat_lat,C_mat_lat,Output_File_Lat_mat,"Lateral")
```

[ ]:
```
#Creating the EigenValuePlots
print(eigen_vals_long)
print(eigen_vals_lat)
#Eigenvals #needs to be dimensional eigenvalues for plot, shows more
long_real = [val.real for val in eigen_vals_long]
long_imag = [val.imag for val in eigen_vals_long]
lat_real =[val.real for val in eigen_vals_lat]
lat_imag = [val.imag for val in eigen_vals_lat]

#Longitudinal

Long_plot = plt.figure()
plt.scatter(long_real,long_imag,color='red',label='Eigenvalues')
plt.axhline(0, color='black', linestyle='--')
plt.axvline(0, color='black', linestyle='--')
plt.xlabel('Real Part')
plt.ylabel('Imaginary Part')
plt.title(f'{base_name} Eigenvalues Longitudinal')
plt.legend()
plt.grid(True)
Long_plot.savefig(f'{base_name}/Long_Eig_plot.png',dpi=200)
plt.show()

#Lateral

Lat_plot = plt.figure()
plt.scatter(lat_real,lat_imag,color='red',label='Eigenvalues')
plt.axhline(0, color='black', linestyle='--')
plt.axvline(0, color='black', linestyle='--')
plt.xlabel('Real Part')
plt.ylabel('Imaginary Part')
plt.title(f'{base_name} Eigenvalues Lateral')
plt.legend()
plt.grid(True)
Lat_plot.savefig(f'{base_name}/Lat_Eig_plot.png',dpi=200)
plt.show()
```

```
[ 0.        +0.j          0.        +0.j          -0.36743301+0.j
 -0.12234424+0.j         -0.00194656+0.03233351j -0.00194656-0.03233351j]
[  0.        +0.j           0.        +0.j
 -34.51729821+0.j          -0.81854038+0.j
  -0.42559321+0.75475346j  -0.42559321-0.75475346j]
```





[ ]:
```
#Approximations of the Short period and Phuguoid mode of the aircraft, compare␣
  ↪to true values

print('Long. Mode Approximation')

#Short Period Mode

A_sp = R_yy*(R_rhox+CL_alpha_hat)
B_sp = R_yy*(CL_alpha+CD_0)-Cm_qbar*(R_rhox +␣
  ↪CL_alpha_hat)-Cm_alpha_hat*(R_rhox - CL_qbar)
C_sp = -Cm_qbar*(CL_alpha+CD_0)-Cm_alpha*(R_rhox-CL_qbar)

eigen_sp = (-B_sp + np.sqrt(B_sp**2-4*A_sp*C_sp))/(2*A_sp)
Damp_rate_sp = (V_0/Wing_MeanChord)*(B_sp/A_sp)
Frequency_sp = (V_0/Wing_MeanChord)*abs(np.sqrt(B_sp**2 -4*A_sp*C_sp)/A_sp)

print(f'\nShort Period Mode:\n')
print(f'Damping Rate: {Damp_rate_sp}')
if Damp_rate_sp < 0:
    print(f'Double Time: {np.log(2.00)/Damp_rate_sp}')
else:
    print(f'Damp Time 99%: {np.log(0.01)/(-Damp_rate_sp)}')
if Frequency_sp != 0:
    print(f'Damped Frequency[rad/s]: {Frequency_sp}')
    print(f'Period [s]: {2*np.pi/Frequency_sp}')

#Phugoid Mode

sigma_D = (g/V_0)*(CD_0/CL_0)
sigma_q = (g/V_0)*((Cm_qbar*(CL_0-CD_alpha))/
 ↪(R_rhox*Cm_alpha+(CD_0+CL_alpha)*Cm_qbar))
R_ps = (R_rhox*Cm_alpha)/(R_rhox*Cm_alpha + Cm_qbar*(CD_0 + CL_alpha))
sigma_phi = -(g/V_0)*R_gx*R_ps*((R_rhox*Cm_qbar-R_yy*(CD_0+CL_alpha))/
 ↪(R_rhox*Cm_alpha + (CD_0+CL_alpha)*Cm_qbar))

Damp_rate_phug = sigma_D + sigma_q + sigma_phi
Frequency_phug = np.sqrt(2*R_ps*(g/V_0)**2 - (sigma_D+sigma_q)**2)

print(f'\nPhugoid Mode:\n')
print(f'Damping Rate: {Damp_rate_phug}')
if Damp_rate_phug < 0:
    print(f'Double Time: {np.log(2.00)/Damp_rate_phug}')
else:
    print(f'Damp Time 99%: {np.log(0.01)/(-Damp_rate_phug)}')
if Frequency_phug != 0:
    print(f'Damped Frequency[rad/s]: {Frequency_phug}')
    print(f'Period [s]: {2*np.pi/Frequency_phug}')
```

```
Long. Mode Approximation

Short Period Mode:

Damping Rate: 10.795802372184895
Damp Time 99%: 0.42657044166102864
Damped Frequency[rad/s]: 6.156358534650823
Period [s]: 1.0206009399574316

Phugoid Mode:

Damping Rate: 0.03567998429740071
Damp Time 99%: 129.06872793449008
Damped Frequency[rad/s]: 1.4748504198606485
Period [s]: 4.260218678836091
```

```
[ ]: # Lateral Mode Approximations
```

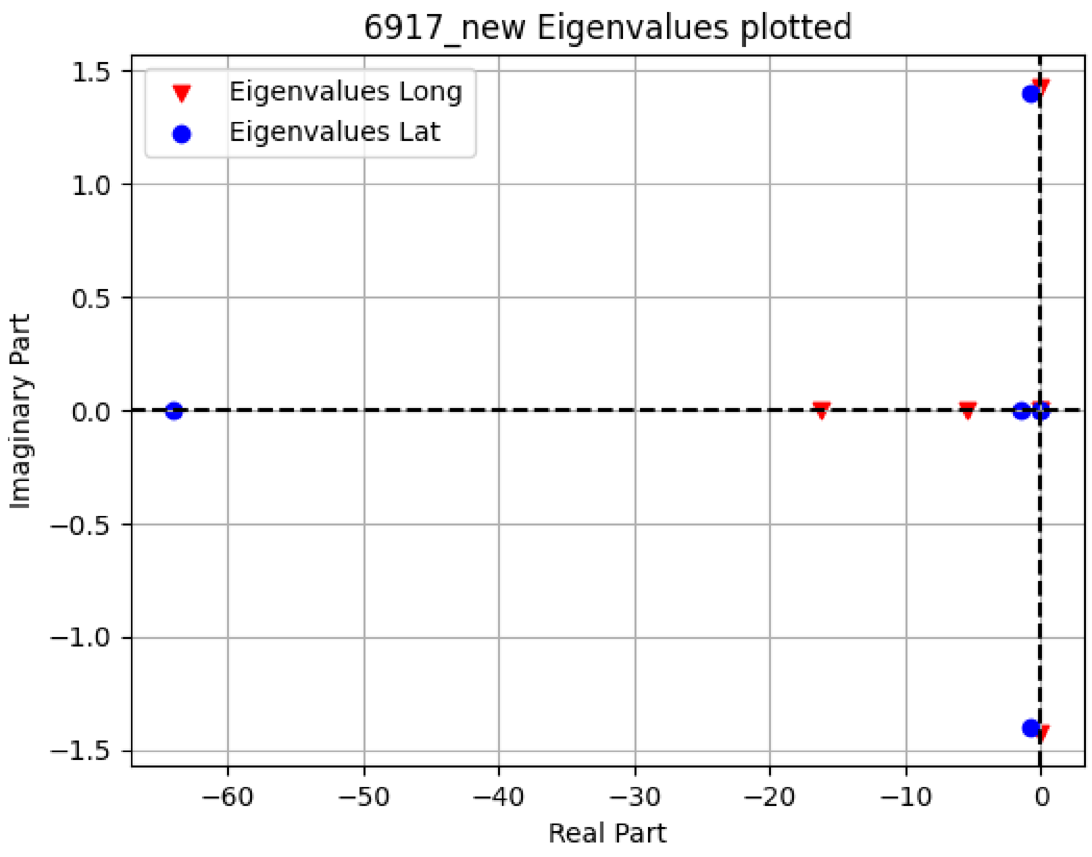
```
print(f'Lateral Mode Approximation \n')

#Roll Mode

sigma_r = ((-V_0*Air_Density*Wing_Area*Wing_Span**2)/(4*I_xx))*Cl_pbar
print(f'\nDamping Rate Roll Mode: {sigma_r}')
if sigma_r > 0:
    print(f'Damping Time 99%: {np.log(0.01)/-sigma_r}')
else:
    print(f'Doubling Time: {np.log(2)/sigma_r}')

#spiral mode

sigma_spiral = (g/V_0)*((Cl_beta*Cn_rbar - Cl_rbar*Cn_beta)/(Cl_beta*Cn_pbar -␣
 ↪Cl_pbar*Cn_beta))
print(f'\nDamping Rate Spiral Mode: {sigma_spiral}')
if sigma_spiral > 0:
    print(f'Damping Time 99%: {np.log(0.01)/-sigma_spiral}')
else:
    print(f'Doubling Time: {np.log(2)/-sigma_spiral}')

#Dutch Roll Approx
R_Ds = (Cl_beta*(R_gy*R_rhoy*R_zz-(R_rhoy-CY_rbar)*Cn_pbar)␣
 ↪-CY_beta*Cl_rbar*Cn_pbar)/(R_rhoy*R_zz*Cl_pbar)
sigma_dr = (-V_0/Wing_Span)*(CY_beta/R_rhoy + Cn_rbar/R_zz - (Cl_rbar*Cn_pbar)/
 ↪(Cl_pbar*R_zz) + (R_gy*(Cl_rbar*Cn_beta - Cl_beta*Cn_rbar))/
 ↪(Cl_pbar*(Cn_beta+CY_beta*Cn_rbar/R_rhoy)) - R_xx*(R_Ds/Cl_pbar) )
Frequency_dr = (2*V_0/Wing_Span)*np.sqrt((1-CY_rbar/R_rhoy)*(Cn_beta/R_zz) +␣
 ↪(CY_beta*Cn_rbar)/(R_rhoy*R_zz) + R_Ds -0.25*(CY_beta/R_rhoy + Cn_rbar/
 ↪R_zz)**2 )


print(f'\nDamping Rate Dutch Roll Mode: {sigma_dr}')
if sigma_dr > 0:
    print(f'Damping Time 99%: {np.log(0.01)/-sigma_dr}')
else:
    print(f'Doubling Time: {np.log(2)/-sigma_dr}')
if Frequency_dr != 0:
    print(f'Damped Frequency[rad/s]: {Frequency_dr}')
    print(f'Period [s]: {2*np.pi/Frequency_dr}')
#print results
```

```
Lateral Mode Approximation
```

```
Damping Rate Roll Mode: 62.696073068514906
Damping Time 99%: 0.07345229071931689

Damping Rate Spiral Mode: 4.846484274436409
Damping Time 99%: 0.950208424337376

Damping Rate Dutch Roll Mode: 1.7671176647259874
Damping Time 99%: 2.6060348316998896
Damped Frequency[rad/s]: 1.1592683527698955
Period [s]: 5.419957589773645
```

[ ]:
```
#Problem A14: Print the dimensional eigenvalues for the short period, phugoid,␣
↪roll, spiral and dutch roll modes
#On a single graph
long_co = 2*V_0/Wing_MeanChord
lat_co = 2*V_0/Wing_Span
print(eigen_vals_long)
print(eigen_vals_lat)
eigen_vals_long_dim = np.asarray(eigen_vals_long)*long_co
eigen_vals_lat_dim = np.asarray(eigen_vals_lat)*lat_co

#Eigenvals #needs to be dimensional eigenvalues for plot, shows more
long_real = [val.real for val in eigen_vals_long_dim]
long_imag = [val.imag for val in eigen_vals_long_dim]
lat_real =[val.real for val in eigen_vals_lat_dim]
lat_imag = [val.imag for val in eigen_vals_lat_dim]



#Longitudinal

Eig_both = plt.figure()
plt.scatter(long_real,long_imag,color='red',label='Eigenvalues Long',marker='v')
plt.scatter(lat_real,lat_imag,color='blue',label='Eigenvalues Lat',marker='o')
plt.axhline(0, color='black', linestyle='--')
plt.axvline(0, color='black', linestyle='--')
plt.xlabel('Real Part')
plt.ylabel('Imaginary Part')
plt.title(f'{base_name} Eigenvalues plotted')
plt.legend()
plt.grid(True)
Eig_both.savefig(f'{base_name}/Eig_both_plot.png',dpi=200)
plt.show()
```

```
[ 0.        +0.j          0.        +0.j         -0.36743301+0.j
 -0.12234424+0.j         -0.00194656+0.03233351j -0.00194656-0.03233351j]
[  0.        +0.j            0.        +0.j
 -34.51729821+0.j           -0.81854038+0.j
 -0.42559321+0.75475346j  -0.42559321-0.75475346j]
```



```
[ ]: #A 15, getting the CAP factor for the aircraft

     #Need the short period values
     W_n_sp = np.sqrt((eigen_vals_long[2].real*eigen_vals_long[3].real))*(2*V_0/
      ↪Wing_MeanChord)

     print(f'Undamped Natural Frequency: {W_n_sp}')

     #solve for the cap parameter

     CAP = W_n_sp**2/(CL_alpha/CW)

     print(f'CAP Value: {CAP}')
```

```
Undamped Natural Frequency: 9.34784611423023
CAP Value: 13.222627365519735
```

```
[ ]: #A 16, compute the handling qualities for the baseline glider Category B flight␣
      ↪phases

     #Get all the needed values for each mode

     #shortperiod
     def shortperiod_B_handleval(eigenvalue1,eigenvalue2,vel,chord,CL_alpha,CW):
         #get the cap value
         wn_sp = np.sqrt((eigenvalue1*eigenvalue2))*(2*vel/chord)
         print(wn_sp)
         print(CL_alpha/CW)
         CAP = wn_sp**2/(CL_alpha/CW)
         print(f'Short Period CAP value: {round(CAP.real,4)}\n')

         #now we need the squiggle of the Short period
         num = -(eigenvalue1+eigenvalue2)
         denom = np.sqrt((eigenvalue1*eigenvalue2))

         squiggle = num/denom
         #give value dependent on CAP and Squiggle
         if 0.30 < squiggle < 2.00:
             print(f'Short Period Handling Level: 1\n')

         elif 0.20 < squiggle < 2.00:
             print(f'Short Period Handling Level: 2\n')
         elif 0.16 < squiggle:
             print(f'Short Period Handling Level: 3\n')
         else:
             print(f'Short Period Handling Level: 4, Iteration of Design Required\n')


     #phugoid
     def phugoid_B_handleval(eigenvalue1,eigenvalue2):
         sigma = -eigenvalue1.real

         #evaluate the squiggle
         num = -(eigenvalue1+eigenvalue2)
         denom = np.sqrt((eigenvalue1.real*eigenvalue2.real))
         squiggle = num/denom

         if sigma > 0:
             if sigma > 0.04:
                 print(f'Phugoid Handling Level: 1\n')
             else:
                 print(f'Phugoid Handling Level: 2\n')
         else:
             #check the doubling time
```

```
        double_time = -np.log(2.00)/sigma
        print(f'Divergent Mode: Doubling Time {round(double_time,2)}[sec]\n')
        if double_time > 55:
            print(f'Phugoid Handling Level: 3\n')
        else:
            print(f'Phugoid Handling Level: 4, Iteration of Design Required\n')

#Roll
def roll_B_handleval(eigenvalue):
    sigma = -eigenvalue.real

    HandlingQual = 1/sigma
    #give rating dependent on this value
    if HandlingQual > 0:
        if HandlingQual < 1.4:
            print(f'Roll Handling Level: 1\n')
        if 1.4 < HandlingQual < 3.0:
            print(f'Roll Handling Level: 2\n')
        if 3.0 < HandlingQual < 10.0:
            print(f'Roll Handling Level: 3\n')
        if HandlingQual > 10.0:
            print(f'Roll Handling Level: 4, Iteration of Design Required\n')
    else:
        print(f'Roll Handling Level: 4 Iteration of Design Required\n')

#dutch roll

def dutch_roll_B_handlevel(eigenvalue1,eigenvalue2,vel,span):
    #get the cap value
    wn_dutch = np.sqrt((eigenvalue1*eigenvalue2))*(2*vel/span)

    #now we need the squiggle of the Short period
    num = -(eigenvalue1+eigenvalue2)
    denom = np.sqrt((eigenvalue1.real*eigenvalue2.real))

    squiggle = num/denom

    cond_1 = squiggle
    cond_2 = squiggle*wn_dutch
    cond_3 = wn_dutch

    #Conditions for handling
    #first is it is divergent it is unacceptable
    if squiggle < 0:
        print(f'Ducth Roll Handling Level: 4, Iteration of Design Required\n')
    else:
        if cond_1 > 0.08 and cond_2 > 0.15 and cond_3 > 0.4:
            print(f'Dutch Roll Handling Level: 1\n')
        elif cond_1 > 0.02 and cond_2 > 0.05 and cond_3 > 0.4:
            print(f'Dutch Roll Handling Level: 2\n')
        elif cond_1 > 0.00 and cond_3 > 0.4:
            print(f'Dutch Roll Handling Level: 3\n')
        else:
            print(f'Dutch Roll Handling Level: 4, Iteration of Design␣
 ↪Required\n')

#spiral

def spiral_B_handlevel(eigenvalue):
    sigma_spiral = -eigenvalue.real

    if sigma_spiral < 0:
        double_time = np.log(2.00)/-sigma_spiral
        if double_time > 20:
            print(f'Spiral Handling Level: 1\n')
        elif 20 > double_time > 12:
            print(f'Spiral Handling Level: 2\n')
        elif 12 > double_time > 4:
            print(f'Spiral Handling Level: 3\n')
        else:
            print(f'Spiral Handling Level: 4, Interation of Design Required\n')
    else:
        print(f'Spiral Handling Level: 1\n')
```

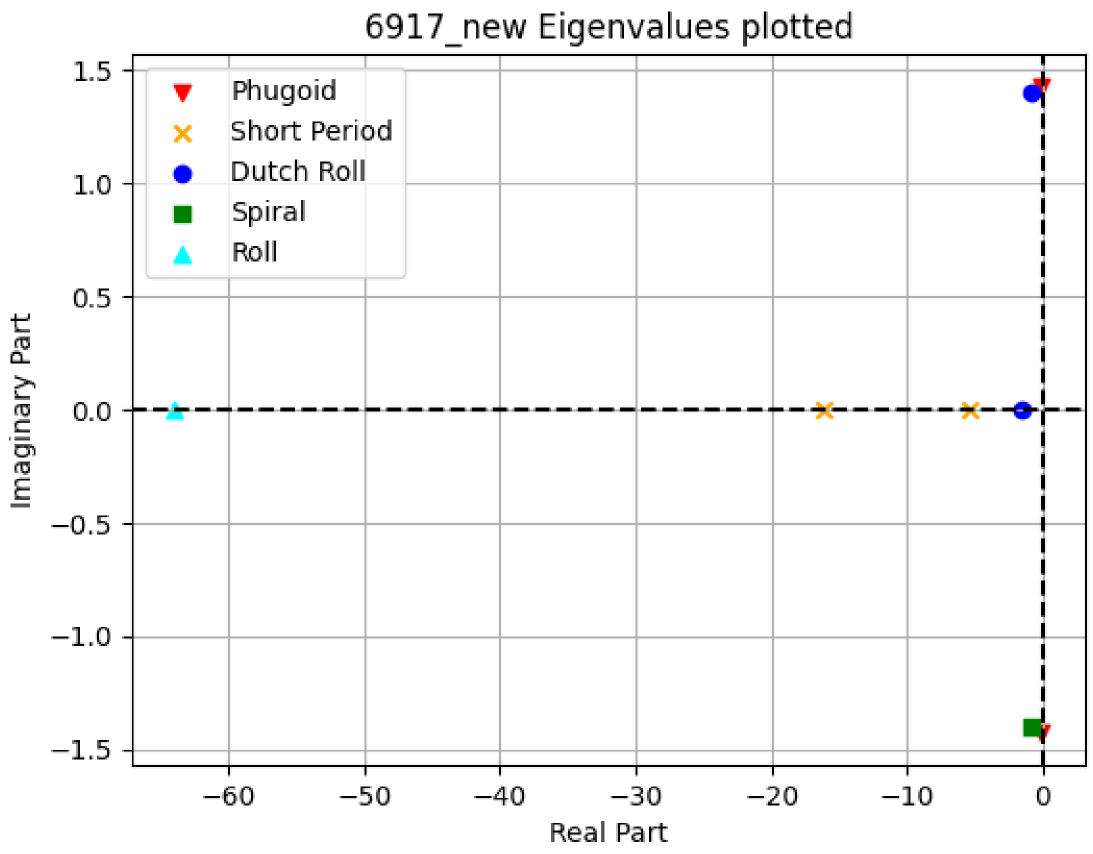
```
[ ]: #Give the Eigenvalues to each mode

     #phugoid mode

     Phugoid_eigenvalue_1 = eigen_vals_long[4]
     Phugoid_eigenvalue_2 = eigen_vals_long[5]
     phugoid_B_handleval(Phugoid_eigenvalue_1,Phugoid_eigenvalue_2)
     #short period mode

     Short_pred_eigenvalue_1 = eigen_vals_long[3]
     Short_pred_eigenvalue_2 = eigen_vals_long[2]

     shortperiod_B_handleval(Short_pred_eigenvalue_1,Short_pred_eigenvalue_2,vel=V_0,chord=Wing_Mea

     #Roll

     Roll_eigenval = eigen_vals_lat[2]
     roll_B_handleval(Roll_eigenval)

     #spiral
```

```
Spiral_eigenval = eigen_vals_lat[5]
spiral_B_handlevel(Spiral_eigenval)

#dutchroll
dutch_eigenval_1 = eigen_vals_lat[3]
dutch_eigenval_2 = eigen_vals_lat[4]

dutch_roll_B_handlevel(dutch_eigenval_1,dutch_eigenval_2,vel=V_0,span=Wing_Span)
```

```
Phugoid Handling Level: 2

(9.34784611423023+0j)
6.608537362490705
Short Period CAP value: 13.2226

Short Period Handling Level: 3

Roll Handling Level: 1

Spiral Handling Level: 1

Dutch Roll Handling Level: 1
```

[ ]:
```
#Plot of all the eigenvalues with the modes labeled

#get the dimensional eigenvalues

#Phugoid

Phugoid_eigenvalue_1 = eigen_vals_long_dim[4]
Phugoid_eigenvalue_2 = eigen_vals_long_dim[5]

#short period mode

Short_pred_eigenvalue_1 = eigen_vals_long_dim[3]
Short_pred_eigenvalue_2 = eigen_vals_long_dim[2]

#Roll

Roll_eigenval = eigen_vals_lat_dim[2]

#spiral
Spiral_eigenval = eigen_vals_lat_dim[5]

#dutchroll
dutch_eigenval_1 = eigen_vals_lat_dim[4]
```

```
dutch_eigenval_2 = eigen_vals_lat_dim[3]

#Put evergything into one figure to see

Eigen_dim_plot = plt.figure()
plt.scatter(Phugoid_eigenvalue_1.real,Phugoid_eigenvalue_1.
 ↪imag,color='red',label='Phugoid',marker='v')
plt.scatter(Phugoid_eigenvalue_2.real,Phugoid_eigenvalue_2.
 ↪imag,color='red',marker='v')
plt.scatter(Short_pred_eigenvalue_2.real,Short_pred_eigenvalue_2.
 ↪imag,color='orange',label='Short Period',marker='x')
plt.scatter(Short_pred_eigenvalue_1.real,Short_pred_eigenvalue_1.
 ↪imag,color='orange',marker='x')
plt.scatter(dutch_eigenval_1.real,dutch_eigenval_1.
 ↪imag,color='blue',label='Dutch Roll',marker='o')
plt.scatter(dutch_eigenval_2.real,dutch_eigenval_2.imag,color='blue',marker='o')
plt.scatter(Spiral_eigenval.real,Spiral_eigenval.
 ↪imag,color='green',label='Spiral',marker='s')
plt.scatter(Roll_eigenval.real,Roll_eigenval.
 ↪imag,color='cyan',label='Roll',marker='^')
plt.axhline(0, color='black', linestyle='--')
plt.axvline(0, color='black', linestyle='--')
plt.xlabel('Real Part')
plt.ylabel('Imaginary Part')
plt.title(f'{base_name} Eigenvalues plotted')
plt.legend()
plt.grid(True)
Eigen_dim_plot.savefig(f'{base_name}/Eig_Model_labeled.png',dpi=200)
plt.show()
```


6917_new Eigenvalues plotted
Phugoid
Short Period
Dutch Roll
Spiral
Roll
Imaginary Part
Real Part
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−60
−50
−40
−30
−20
−10
0