An Introduction into Aeroelasticity

By

Nicholas Cheong

Department of Aerospace Engineering

University of Glasgow

May 2018

Dr George Barakos

Contents

Nomenclature

z	Bending displacement
$\dot{z}$	Rate of curvature
y	Spanwise position
s	Semi-span length
t	Time
θ	Torsional displacement
$\dot{\theta}$	Rate of twist
Ψ_i	Bending assumed shape
$\boldsymbol{\Psi}$	Matrix form of bending assumed shapes
φ_i	Torsional assumed shape
$\boldsymbol{\varphi}$	Matrix form of torsional assumed shapes
$q_i(t)$	Generalized coordinate
N	Number of bending modes
M	Number of torsion modes
T	Kinetic energy
U	Potential energy
τ	Dissipative function
Q_q	Generalized force
m	Mass
c	Damping
k	Stiffness

μ	Mass per length
EI	Flexural rigidity
GJ	Torsional rigidity
χ	Moment of inertia in twist per unit length
x_A	Distance between the elastic and mass axis
b	Semi-chord
m_s	Mass of concentrated point mass
x_s	Distance between store centre of mass and wing flexural axis
M_{wing}	Wing mass matrix
M_{store}	Store mass matrix
M_{total}	Total mass matrix
K_{wing}	Wing stiffness matrix
λ	Eigenvalue
ω	Natural frequency (rad/s)
DEL	Bending contribution to wing mass matrix
C	Bending-torsion contribution to wing mass matrix
D	Torsion contribution to wing mass matrix
$DELS$	Bending contribution to mass store matrix
Cs	Bending-torsion contribution to mass store matrix
Ds	Torsion contribution to mass store matrix
B	Bending contribution to stiffness matrix
T	Torsion contribution to stiffness matrix
ϕ	Potential function
ψ	Stream function

Γ	Circulation
V_θ	Tangential velocity
r	Radius
L	Lift
ρ	Density
V	Velocity
$\underline{r_i}$	Spatial vectors
w	Downwash
$\boldsymbol{\eta}$	Influence coefficient matrix
α	Incidence angle
α_e	Effective angle of attack
Λ	Sweep angle
Δy	Distance between panels
$y_{c,space}$	Cosine spacing
OF	Initial offset
RF	Relaxation factor
L_{over}	Lift over (or below) the trim state weight
α_{over}	Angle of attack over (or below) to produce differential in L_{over}
g	Acceleration due to gravity
$C_{l\alpha}$	Lift slope
k	Reduced frequency in semi-chords
ν	Reduced frequency in chord
c	Chord
$C_{theo}(k)$	Total Theodorsen's function

$F(k)$	Real part of Theodorsen's function
$iG(k)$	Imaginary part of Theodorsen's function
K_i	Modified Bessel functions of a second kind
$\emptyset$	Characteristic polynomial
Q	Unsteady aerodynamic matrix
A^*	Apparent mass matrix
B^*	Apparent damping matrix
C^*	Apparent stiffness matrix
g	Damping
$\hat{A}$	Aerodynamic response matrix
e_w	Distance between elastic axis and centre of gravity
U_F	Flutter velocity

Chapter 1:

Brief introduction into Aeroelasticity

Aeroelasticity is the analysis of the deformation of a wing (or rotor) caused by the interaction of aerodynamic, elastic and inertial forces [1]. The sub-sector can be classified into static and dynamic aeroelasticity. In terms of a wing, the static aeroelasticity is the study of the deformation of the wing affecting the lift distribution and therefore the static stability of the aircraft, which in turn, can also lead to control-loss known as aileron reversal [1]. The dynamic aeroelasticity includes a phenomenon known as flutter in which entails the coupling of vibrational modes, resulting in the wing extracting energy from the freestream. This leads to self-excited, unstable vibrations in the wing, which can potentially lead to divergence, i.e. wing failure [1]. With the above considered, the aeroelasticity subsector is a very important factor which must be considered for the viability of any aircraft. The objective of this report is to explores key areas into static and dynamic aeroelasticity through three assignments. These assignments will, so to speak, scratch the surface of the work that has been achieved over the past decades and are entailed in the three prevailing chapters.

- ***Chapter 1****: Mode shapes and natural frequency*

  This is an overview of the aeroelastic structural considerations of a wing where the effects of changing the mass axis in relation to the elastic axis are considered on the natural frequencies of the normal modes.

- ***Chapter 2:*** *Elastic wing with aerodynamics*

  This is an overview of static aeroelasticity which entails the effects of wing deformation on the lift distribution of a wing. Furthermore, wing sweep is also considered on the above.

- ***Chapter 3:*** *Predicting flutter*

  Here is where the dynamic aeroelasticity is introduced and overviewed, where the effects of changing the mass axis in relation to the elastic axis are considered on the flutter speed, frequency and damping of a wing.

Chapter 2:
Mode shapes and natural frequency

2.1 Introduction and objectives

Determining the mode shapes and natural frequency is essential for aeroelastic analysis; the objective of this assignment was to determine these properties of an un-swept, un-tapered wing and compare the results with paper [2] by Runyan and Sewall of the National Advisory Committee of Aeronautics (NACA). The wing analysed in the assignment was identical to that in the paper in which investigates the aeroelastic properties of the wing with a concentrated weight at variant chord- and spanwise locations. The results of this assignment should correlate with the experimental results from the paper. The method used to establish the structural model of the wing entails the Rayleigh Ritz method for assumed shapes followed by the Lagrange energy equation to obtain the mass and stiffness matrices. The natural frequencies and mode shapes can then be extracted by obtaining the eigenvectors and eigenvalues of the structural model.

2.2 Rayleigh Ritz method

The Rayleigh Ritz method is used to model continuous systems with a finite series of assumed shapes to represent the deformation [1]. This method is an approximation method; in terms of accuracy, the quantity of assumed shapes used in the series of shapes follows similar principles to the accuracy of the Taylor's Series expansion where the higher number of terms included results in a closer approximation to the exact solution. However, with increase of number of assumed shapes results in significant increase of computational costs. On an additional note on accuracy, the number of normal modes of the system is equivalent to the number of assumed shapes employed, where the accuracy the lower frequency modes increases if a prevailing mode is present [1].

The functions used for the assumed shapes are typically in polynomial, trigonometric or hyperbolic form [1]. The shapes must obey the kinematic boundary conditions of the system and the accuracy in the portrayal of the system will increase if the natural boundary conditions are also satisfied [1]. The kinematic and natural boundary conditions for the wing are likewise to a cantilever beam with a built-in support and free end; in a two-degree of freedom wing in bending and torsion, the boundary conditions are:

Kinematic boundary conditions:

$z(y,t) = 0$ at $y = 0$ Bending (1)

$\dot{z}(y,t) = 0$ at $y = 0$ (2)

$\theta(y,t) = 0$ at $y = 0$ Torsion (3)

$\dot{\theta}(y,t) = 0$ at $y = 0$ (4)

Natural boundary conditions:

$z(y,t) = 0$ at $y = s$ Bending (5)

$\dot{z}(y,t) = 0$ at $y = s$ (6)

$\dot{\theta}(y,t) = 0$ at $y = s$ Torsion (7)

Where z and θ are the bending and torsion deformation respectively and s the semi-span of the wing. The function used for bending and torsion assumed shapes were simple polynomial functions for simplicity.

$$z(y,t) = \sum_{i=1}^{N} \Psi_i q_i(t) = \left(\frac{y}{s}\right)^2 q_1(t) + \left(\frac{y}{s}\right)^3 q_2(t) \ldots . \left(\frac{y}{s}\right)^N q_N(t) \quad (8)$$

$$\theta(y,t) = \sum_{i=1}^{M} \varphi_i q_i(t) = \left(\frac{y}{s}\right) q_1(t) + \left(\frac{y}{s}\right)^2 q_2(t) \ldots . + \left(\frac{y}{s}\right)^M q_M(t) \quad (9)$$

Where $q_i(t)$ is the generalised coordinate, N the number of bending assumed shapes, and M the number of torsional assumed shapes. These simple assumed shapes satisfy the kinematic boundary conditions, however, not the natural boundary conditions which reduces the accuracy of the method.

2.3 Developing structural matrices for the wing

Mass and stiffness matrices were required to calculate the natural frequencies for comparison with the results in the paper [2] by Runyan and Sewall.

2.3.1 Lagrange's equation

The mass and stiffness matrices were formed with the use of Lagrange's equation which are a system of differential energy equations expressed in generalised coordinates, $q(t)$. Equation (10) below is the Lagrange's equation of a single degree of freedom system (SDOF).

$$\frac{d}{dt}\left(\frac{\partial T}{\partial \dot{q}}\right) - \frac{\partial T}{\partial q} + \frac{\partial \tau}{\partial \dot{q}} + \frac{\partial U}{\partial q} = Q_q \tag{10}$$

Where, T, U, τ and Q_q are the kinetic energy, potential energy, dissipative function and generalised force respectively. Implementing the single degree of freedom energy equations, the equation of motion of the wing surmounts to the ordinary second-order differential equation:

$$m\ddot{x} + c\dot{x} + kx = f(t) \tag{11}$$

2.3.2 Kinetic and potential energy

The continuous form of kinetic and potential energy of the wing for bending and torsion involve the integral over the span of the wing and therefore resulting in the total kinetic and potential energy. The energy equations for bending, T_b and U_b, are displayed bellow:

$$T_b = \frac{1}{2}\int_0^s \mu \dot{z}^2 \, dy \tag{12}$$

$$U_b = \frac{1}{2}\int_0^s EI \left(\frac{\partial^2 z}{\partial y^2}\right)^2 dy \tag{13}$$

Where, μ is the mass per length and EI the flexural rigidity. For the torsional kinetic and potential energy, T_t and U_t:

$$T_t = \frac{1}{2}\int_0^s \chi \dot{\theta}^2 \, dy \tag{14}$$

$$U_t = \frac{1}{2}\int_0^s GJ \left(\frac{\partial \theta}{\partial y}\right)^2 dy \tag{15}$$

Where, χ is the moment of inertia in twist per unit length, and GJ the torsional rigidity. Notably, the variables μ, EI, χ and GJ are currently within the continuous integral; this would be the case if the wing was tapered as these variables would vary in length [1], however, for this assignment the variables are constant since the wing is un-tapered (straight).

2.3.3 Matrix approach

In order to analyse multiple modes in bending and torsion, the calculations were set in matrix form and the assumed shape for bending and torsion become:

$$z(y,t) = \sum_{i=1}^{N} \Psi_i q_i(t) = \begin{bmatrix} \left(\frac{y}{s}\right)^2 \\ \left(\frac{y}{s}\right)^3 \\ \cdots \\ \left(\frac{y}{s}\right)^N \end{bmatrix} [q_1(t)\ q_2(t)\ \ldots\ q_N(t)] = \boldsymbol{\Psi q} \tag{16}$$

$$\theta(y,t) = \sum_{i=1}^{M} \varphi_i q_i(t) = \begin{bmatrix} \left(\frac{y}{s}\right) \\ \left(\frac{y}{s}\right)^2 \\ \cdots \\ \left(\frac{y}{s}\right)^M \end{bmatrix} [q_1(t)\ q_2(t)\ \ldots\ q_M(t)] = \boldsymbol{\varphi q} \tag{17}$$

Using the bending as an example, implementing the matrix form of the assumed shapes into the equations for kinetic and potential energy yields:

$$T_{b,MF} = \frac{1}{2}\mu \dot{\boldsymbol{q}}^{\boldsymbol{T}} \left[\int_0^s \boldsymbol{\Psi}\boldsymbol{\Psi}^{\boldsymbol{T}} dy\right] \dot{\boldsymbol{q}} = \frac{1}{2}\dot{\boldsymbol{q}}^{\boldsymbol{T}} \boldsymbol{M_b} \dot{\boldsymbol{q}} \tag{18}$$

$$U_{b,MF} = \frac{1}{2} EI \boldsymbol{q}^{\boldsymbol{T}} \left[\int_0^s \boldsymbol{\Psi}''\boldsymbol{\Psi}''^{\boldsymbol{T}} dy\right] \boldsymbol{q} = \frac{1}{2}\boldsymbol{q}^{\boldsymbol{T}} \boldsymbol{K_b} \boldsymbol{q} \tag{19}$$

Where, $\boldsymbol{\Psi\Psi^T}$ and $\boldsymbol{\varphi\varphi^T}$, are simply the squared value of the matrix of assumed shapes. Implementing the resultant energy equations into Lagrange's equation requires matrix differentiation; here, the matrix differentiation rule of $E = \dot{\boldsymbol{q}}^{\boldsymbol{T}} \boldsymbol{M_b} \dot{\boldsymbol{q}}$ [3], where $\boldsymbol{M_b}$ is symmetric then, $\partial E/\partial \dot{\boldsymbol{q}} = 2\boldsymbol{M_b}\dot{\boldsymbol{q}}$. This results in the Lagrange's equation for bending:

$$\left[\mu \int_0^s \boldsymbol{\Psi}\boldsymbol{\Psi}^{\boldsymbol{T}} dy \right] \ddot{\boldsymbol{q}} + \left[EI \int_0^s \boldsymbol{\Psi}''\boldsymbol{\Psi}''^{\boldsymbol{T}} dy\right] \boldsymbol{q} = \boldsymbol{\Psi}(a) f(t) \tag{20}$$

$$\boldsymbol{M_b}\ddot{\boldsymbol{q}} + \boldsymbol{K_b}\boldsymbol{q} = \boldsymbol{\Psi}(a) f(t) \tag{21}$$

Combining both bending and torsion into a single mass and stiffness matrix yields [3]:

$$M_{wing} = \begin{bmatrix} \mu \int_0^s \boldsymbol{\Psi}\boldsymbol{\Psi}^{\boldsymbol{T}} dy & -\mu x_A b \int_0^s \boldsymbol{\Psi}\boldsymbol{\varphi}^{\boldsymbol{T}} dy \\ -\mu x_A b \int_0^s \boldsymbol{\varphi}\boldsymbol{\Psi}^{\boldsymbol{T}} dy & \chi_{wing} \int_0^s \boldsymbol{\varphi}\boldsymbol{\varphi}^{\boldsymbol{T}} dy \end{bmatrix} \tag{22}$$

$$K_{wing} = \begin{bmatrix} EI \int_0^s \boldsymbol{\Psi}''\boldsymbol{\Psi}''^{\boldsymbol{T}} dy & \mathbf{0} \\ \mathbf{0} & GJ \int_{\mathbf{0}}^{\boldsymbol{s}} \boldsymbol{\varphi}''\boldsymbol{\varphi}''^{\boldsymbol{T}} dy \end{bmatrix} \tag{23}$$

Where, b is the semi-chord and x_A is the distance between the elastic and mass axis. The term x_A is required to define the system at the so-called flexural axis in which is the position of the shear centres along the wing. At the shear centre, shear causes no twist to occur and torque causes no bending [1], therefore, this decouples the bending and torsion modes and allows the matrices to be symmetrical.

The number of modes dictates the size of the matrices, where the product of the assumed shape matrices will always produce a square matrix, thus the total wing matrix size can be related to $(N + M)$ by $(N + M)$. This highlights the increase in computational costs with increase in mode shapes. Equation (24) below shows depicts the above where, x_i and y_i are the sparse matrix indices:

$$\begin{bmatrix} x_1, y_1 & \cdots & x_{N+M}, y_1 \\ \vdots & \ddots & \vdots \\ x_1, y_{N+M} & \cdots & x_{N+M}, y_{N+M} \end{bmatrix} \tag{24}$$

2.3.4 Implementing the mass storage

The mass storage matrix is analogous to the mass matrix of the wing, however, the energy equations are not integrated over a length as it is considered as a concentrated point mass. Furthermore, the mass store does not affect the stiffness of the wing. Equation (25) below displays the mass store matrix, where m_s is the mass of the store and x_s is the distance between the store centre of mass and the flexural axis.

$$M_{store} = \begin{bmatrix} m_s \boldsymbol{\Psi}\boldsymbol{\Psi}^{\boldsymbol{T}} & m_s x_s \boldsymbol{\Psi}\boldsymbol{\varphi}^{\boldsymbol{T}} \\ m_s x_s \boldsymbol{\varphi}\boldsymbol{\Psi}^{\boldsymbol{T}} & \chi_{store} \boldsymbol{\varphi}\boldsymbol{\varphi}^{\boldsymbol{T}} \end{bmatrix} \tag{25}$$

The total mass matrix derives from the addition of the wing and storage mass matrices:

$$M_{total} = M_{wing} + M_{store} \tag{26}$$

$$M_{total} = \begin{bmatrix} \mu \int_0^s \boldsymbol{\Psi}\boldsymbol{\Psi}^{\boldsymbol{T}} dy & -\mu x_A b \int_0^s \boldsymbol{\Psi}\boldsymbol{\varphi}^{\boldsymbol{T}} dy \\ -\mu x_A b \int_0^s \boldsymbol{\varphi}\boldsymbol{\Psi}^{\boldsymbol{T}} dy & \chi_{wing} \int_0^s \boldsymbol{\varphi}\boldsymbol{\varphi}^{\boldsymbol{T}} dy \end{bmatrix} + \begin{bmatrix} m_s \boldsymbol{\Psi}\boldsymbol{\Psi}^{\boldsymbol{T}} & m_s x_s \boldsymbol{\Psi}\boldsymbol{\varphi}^{\boldsymbol{T}} \\ m_s x_s \boldsymbol{\varphi}\boldsymbol{\Psi}^{\boldsymbol{T}} & \chi_{store} \boldsymbol{\varphi}\boldsymbol{\varphi}^{\boldsymbol{T}} \end{bmatrix} \tag{27}$$

2.3.5 Natural frequency, Eigenvectors and Eigenvalues

Eigenvectors and values are the linear transformation coordinates of a system, where the eigenvector is the transformation and eigenvalue the scaling. They are widely used in many technical sectors including engineering; in terms of the aeroelastic analysis, the eigenvectors are the undamped normal modes of the system and eigenvalues represent the natural frequency of the mode [1]. These are calculated in equation (28).

$$\left|[K_{wing}] - \lambda[M_{total}]\right| = 0 \tag{28}$$

$$\omega^2 = \lambda \tag{29}$$

2.4 Implementation into Matlab

The components of the resulting mass and stiffness matrices were labelled and referred to within the code were as follows:

$$M_{wing} = \begin{bmatrix} \mu \int_0^s \boldsymbol{\Psi}\boldsymbol{\Psi}^{\boldsymbol{T}} dy & -\mu x_A b \int_0^s \boldsymbol{\Psi}\boldsymbol{\varphi}^{\boldsymbol{T}} dy \\ -\mu x_A b \int_0^s \boldsymbol{\varphi}\boldsymbol{\Psi}^{\boldsymbol{T}} dy & \chi_{wing} \int_0^s \boldsymbol{\varphi}\boldsymbol{\varphi}^{\boldsymbol{T}} dy \end{bmatrix} = \begin{bmatrix} \mu\, \boldsymbol{DEL} & -\mu x_A b\, \boldsymbol{C} \\ -\mu x_A b\, \boldsymbol{C^T} & \chi_{wing}\, \boldsymbol{D} \end{bmatrix} \tag{30}$$

$$M_{store} = \begin{bmatrix} m_s \boldsymbol{\Psi}\boldsymbol{\Psi}^{\boldsymbol{T}} & m_s x_s \boldsymbol{\Psi}\boldsymbol{\varphi}^{\boldsymbol{T}} \\ m_s x_s \boldsymbol{\varphi}\boldsymbol{\Psi}^{\boldsymbol{T}} & \chi_{store} \boldsymbol{\varphi}\boldsymbol{\varphi}^{\boldsymbol{T}} \end{bmatrix} = \begin{bmatrix} \mu\, \boldsymbol{DELS} & -\mu x_A b\, \boldsymbol{Cs} \\ -\mu x_A b\, \boldsymbol{Cs^T} & \chi_{wing}\, \boldsymbol{Ds} \end{bmatrix} \tag{31}$$

$$K_{wing} = \begin{bmatrix} EI \int_0^s \boldsymbol{\Psi''}\boldsymbol{\Psi''}^{\boldsymbol{T}} dy & 0 \\ 0 & GJ \int_{\boldsymbol{0}}^{\boldsymbol{s}} \boldsymbol{\varphi''}\boldsymbol{\varphi''}^{\boldsymbol{T}} dy \end{bmatrix} = \begin{bmatrix} EI\, \boldsymbol{B} & 0 \\ 0 & GJ\, \boldsymbol{T} \end{bmatrix} \tag{32}$$

The matrices above were determined using two mathematical procedures for integration: symbolic and trapezoidal. Symbolic was also used as it represents the assumed shapes as a continuous system, whereas, the trapezoidal approximates the function discretely. Therefore, the symbolic should provide an increase in accuracy.

Figure 2.1 displays a flowchart of the trapezoidal integration which outlines the code structure for executing the methodology. The symbolic integration code varies to the trapezoidal as the integration is only executed once to obtain a symbolic expression for each of the matrices, whereas, the trapezoidal requires integration to be executed for every change in store position as the total mass matrix is redefined; this is indicated on the flowchart.

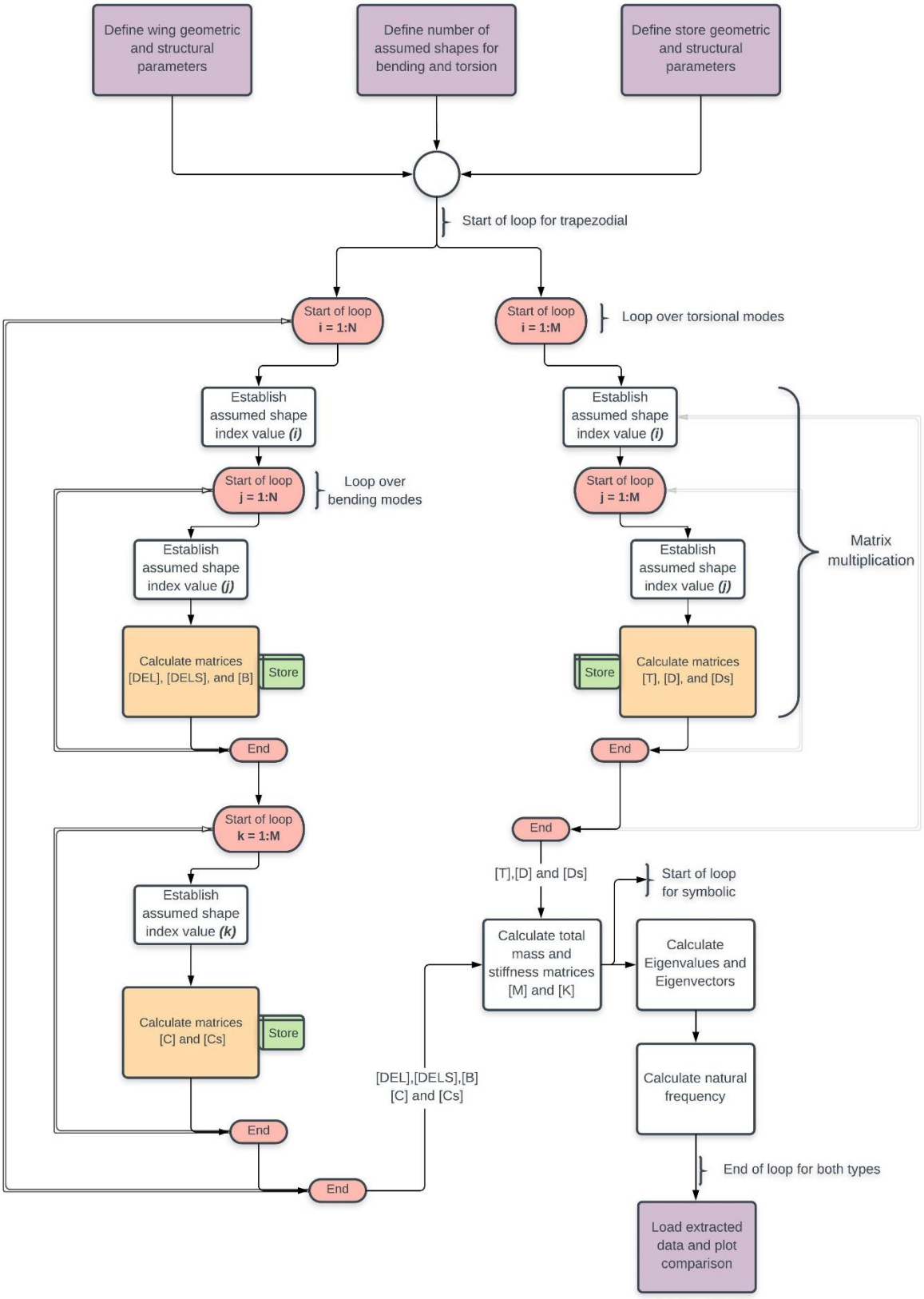


Figure 2.1 – Flow chart of procedure for Assignment 1

2.5 Results of assumed shape approach

The results of the assumed shape method using symbolic and trapezoidal integration were compared with the experimental results from the [2]. The simulation of the methodology used $N = 3$ bending modes and $M = 2$ torsional modes which results in a total of $N + M$ mode shapes. The first two modes are the first and second bending modes and the third mode is the first torsional mode. Against store position, figures 2.2 → 2.4 display the results which entail the frequency in cycles per second using trapezoidal integration, the first and second bending mode normalized frequencies and first torsion mode normalized frequency.

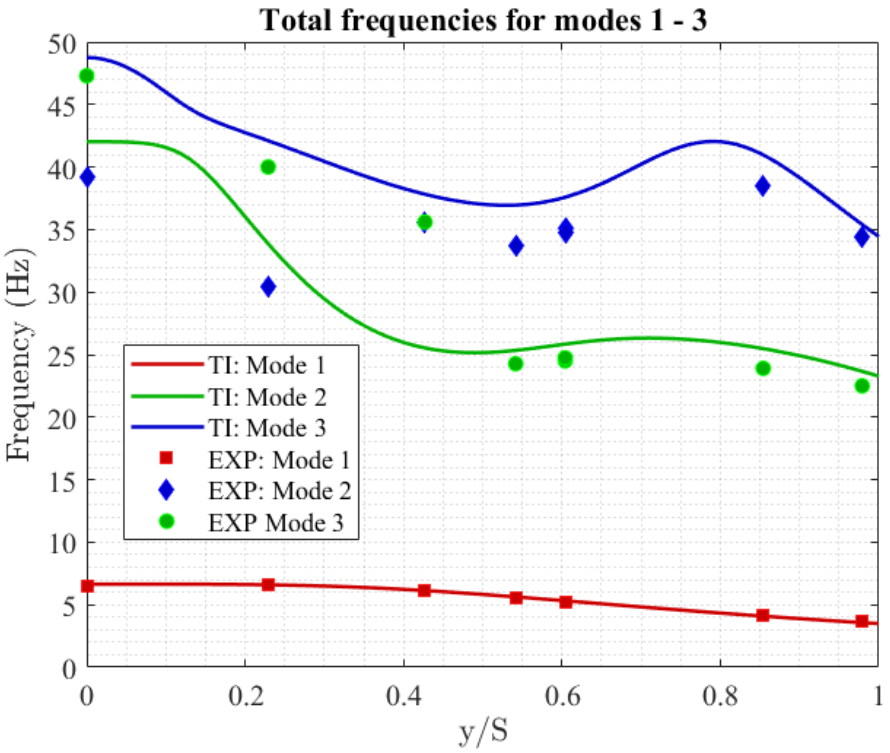


Figure 2.2 – Frequency (Hz) of wing against store position, y/S, where TI and EXP denote trapezoidal integration and experimental respectively.

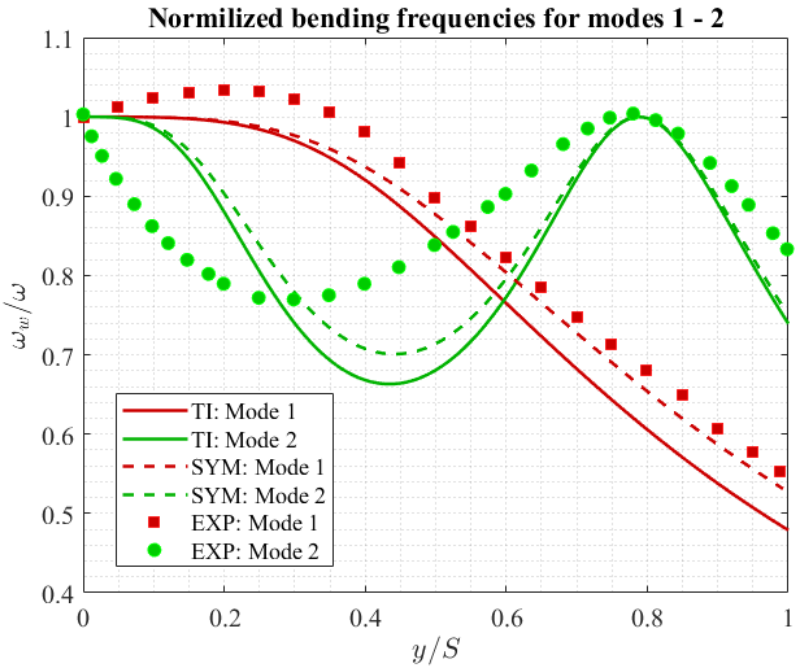


Figure 2.3 – Normalized bending frequency (mode 1 and 2) against store position, y/S. Where SYM, TI and EXP denote symbolic, trapezoidal integration and experimental respectively.

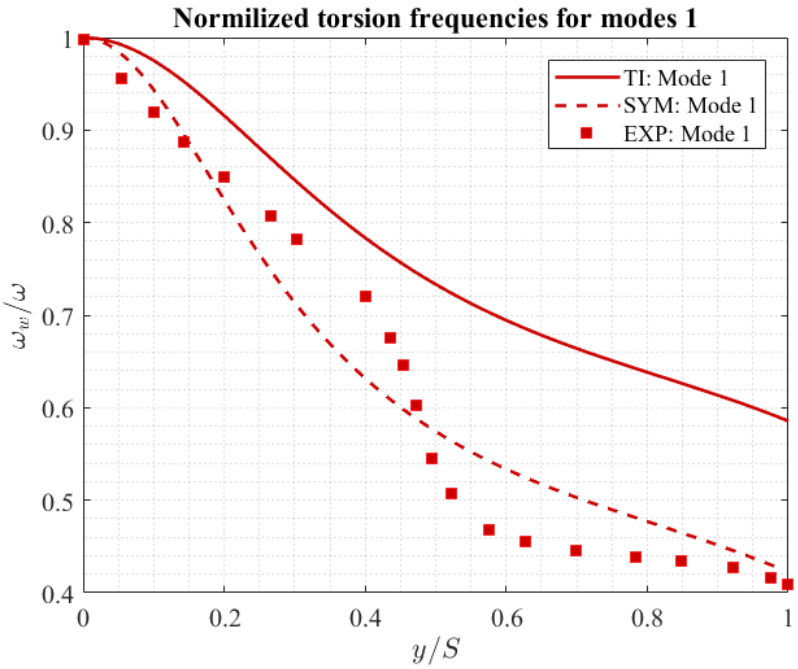


Figure 2.4 – Normalized torsion frequency (mode 3) against store position, y/S. Where SYM, TI and EXP denote symbolic, trapezoidal integration and experimental respectively.

The symbolic integration correlates significantly more accurately than the trapezoidal integration; this is due to the continuous nature of the integration. The first bending mode compares well with the experimental results which is due to the presence of the subsequent modes, however, the results for the second bending mode do not agree well with the experimental results despite the presence of a third bending mode. The symbolic approach for the torsional mode is in some agreement as it passes through experimental results, however, does not capture the changes over the span accurately. The difference in the results can be due to several factors:

- The natural boundary conditions were not satisfied, therefore, detrimental to the physical approximation of the solution. This is a result of the assumed shape being too simplistic.
- Treating the mass as a concentrated point mass is an assumption. Physically, it is a distributed load over an area and therefore will have a different effect on the frequency of the wing.
- Inadequate quantity of assumed shapes, this will be explored in the following section.

2.5.1 Investigating different number of modes

Two additional configurations of assumed modes will be used to investigate the effect of number of assumed modes:

- Configuration (1) – $N = 2,\ M = 1$
- Configuration (2) – $N = 4,\ M = 3$

Figures 2.5 and 2.6 displays the results for configuration (1) and (2) for the first and second bending modes respectively, figure 2.7 displays the first torsional mode for both configurations:

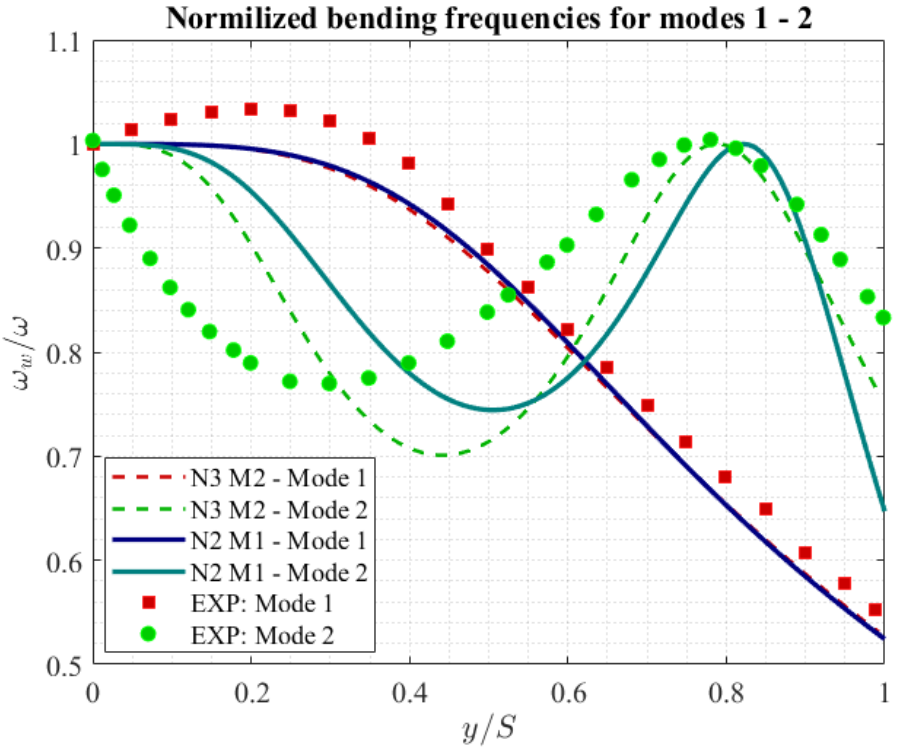


Figure 2.6 – Normalized bending frequency (mode 1 and 2) against store position, y/S. Comparison between N=3, M=2 and N =2, M =1 number of mode shapes.

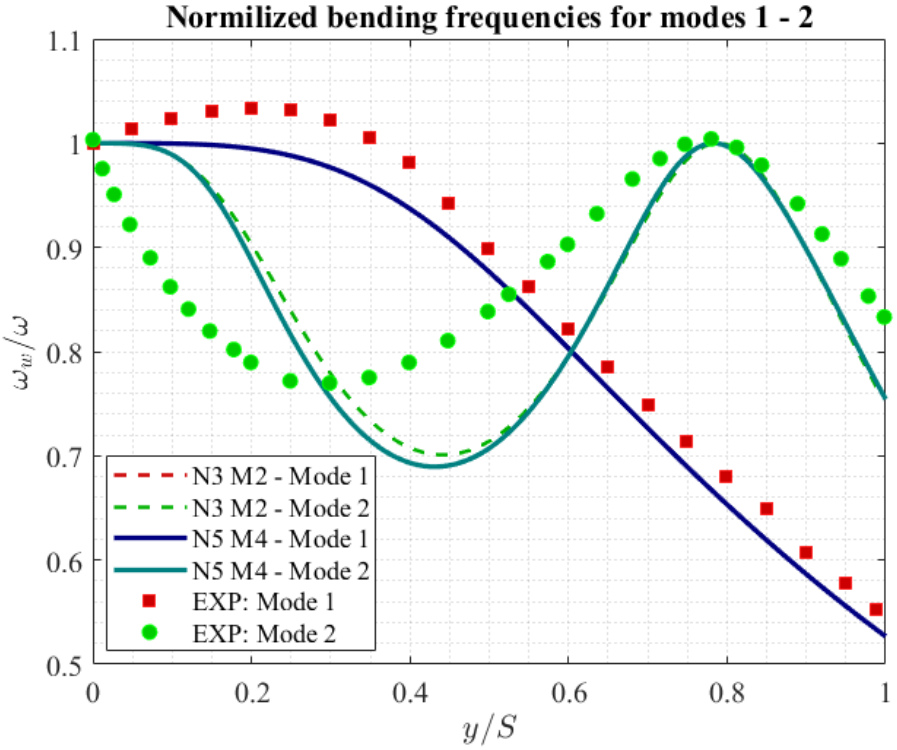


Figure 2.7 – Normalized bending frequency (mode 1 and 2) against store position, y/S. Comparison between N=3, M=2 and N =5, M =4 number of mode shapes.

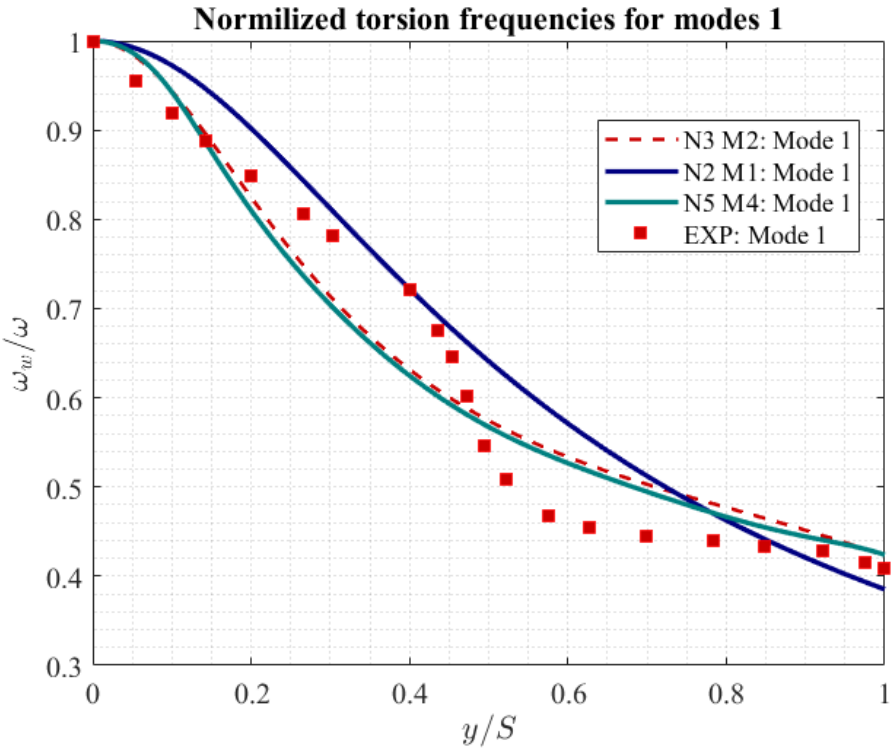


Figure 2.7 - Normalized torsional frequency (mode 3) against store position, y/S. Comparison between N=3, M=2 and configuration (1) and (2).

For configuration (1), the decrease in mode shapes evidently decreases the accuracy of the simulation as the second bending and first torsion modes further deviate from the experimental results, however, the first bending mode is similar to the original results which is to be expected due to the subsequent mode shape (N = 2). For configuration (2), negligible change occurs when compared. Therefore, the deviations with the experimental results must be due to the lack of satisfaction of the natural boundary conditions and a more adept assumed shape should be used. Furthermore, the deviations could be a result of the assumption that the mass store can be treated as a point mass results in significant error, and therefore, a more complicated model is required.

Chapter 3:

Elastic wing with aerodynamics

3.1 Overview and objectives

This chapter entails the creation of a trimmer model for an aircraft in cruise conditions which has an elastic wing able to deform under aerodynamic loads. The trimmer model should alter the wing pitch angle to account for the elastic lift produced by the bending and twist, and finally, converge when the model total lift is equivalent to its weight.

Vortex Lattice method (VLM) code was created to establish the aerodynamic loads. Furthermore, the VLM code was to be validated against another VLM code, the **VLMpc** from [4] to ensure fidelity. This was then implemented into the trimmer model of the aircraft as a subroutine, where the aerodynamics were to be recalculated as the wing bends and twists, until the model converges. Furthermore, the effect of sweep on the wing will also be considered. Wing sweep is typically employed to reduce compressibility effects for transonic and supersonic flight [5], however, wing sweep also alters the spanwise lift distribution and the effects of which will also be overviewed.

3.2 Background of Vortex Lattice method

Vortex Lattice Method (VLM) was an innovative method developed in the 1930's and used to gain insight into three-dimensional wing effects [6]. It follows the same principles to panel methods, where the method is based on idealized flow theory and the solution to the well-known Laplace equation.

$$\nabla^2\phi = 0 \qquad \text{Laplace equation} \qquad (33)$$

$$\frac{\partial^2\phi}{\partial x} + \frac{\partial^2\phi}{\partial y} + \frac{\partial^2\phi}{\partial x} = 0 \qquad \text{Three-dimensional Laplace for the potential function} \qquad (34)$$

$$\frac{\partial^2\psi}{\partial x} + \frac{\partial^2\psi}{\partial y} + \frac{\partial^2\psi}{\partial x} = 0 \qquad \text{Three-dimensional Laplace for the stream function} \qquad (35)$$

These equations are linear partial differential equations; the importance of this is that the superposition concept is applicable, therefore, solutions can be added together to formulate complete flow systems. For a vortex, the derived functions of the tangential, potential and stream function are displayed below [6].

$$V_\theta = \frac{\Gamma}{2\pi r} \tag{36}$$

$$\phi = -\frac{\Gamma}{2\pi}\theta \tag{38}$$

$$\psi = \frac{\Gamma}{2\pi}\ln(r) \tag{39}$$

These are the so-called building-blocks of potential flow systems entailing the use of rotational flows [6]; through which and using Bernoulli's equation the lift per unit span can of a continuous system can be found as [1]:

$$L = \rho V \Gamma \tag{40}$$

This is known as the Kutta-Joukowski theorem [6], which states that the lift per unit span over any arbitrary body is directly proportional to the circulation.

This introduces the Vortex Lattice method, where in order to model the circulation distribution across the wing, the wing is discretized with a series of panels each entailing a horseshoe vortex. Element to element interaction occurs between horseshoe vortexes and therefore accounts for three-dimensional effects, where the highest change in circulation occurs at the tip of the wing; strong tip vortex occurs due to the pressure equalization between the upper and lower surfaces [7]. Figure 3.1 displays an exemplary discretization of an arbitrary wing.

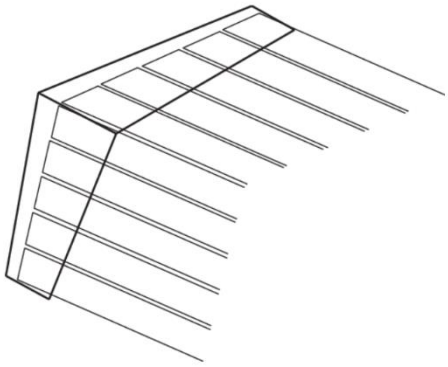


Figure 3.1 – Example of panel distribution, each entailing a horseshoe vortex. Image courtesy of [10]

Horseshoe vortexes typically consist of a closed vortex filament and are based of Helmholtz first and second theorem of vortexes.

(1) The strength of a vortex filament is constant along its length.
(2) The vortex filament cannot end but extend to either the fluid domain boundary or form a closed loop.

In the case of horseshoe vortexes for a wing, the filament is in the streamwise direction can extend to infinity. This satisfies Helmholtz second theory and simplifies the model; furthermore, extending to infinity accounts for the entire wing wake history. Extending to infinity is not applicable to all systems, for example, helicopter rotors cannot employ this as the rotors can be in the wake of another rotor. The horseshoe vortexes are split into three components: $\infty - A$, $A - B$ and $B - \infty$. The collocation point, C, is typically located on the three-quarter chord and is where the aerodynamic force and incidence angle are calculated. The vortex filament between A and B is the bound vortex on the wing and is typically located on the quarter chord. Figure 3.2 displays a typical horseshoe vortex with points A, B and C; image courtesy of Dr Barakos [7].

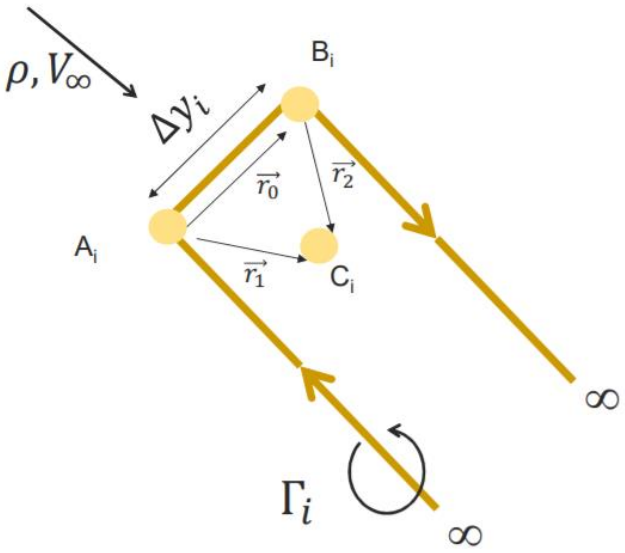


Figure 3.2 – Horseshoe vortex displaying points, vortex filaments and spatial vectors. This is a typical vortex within each panel. Image courtesy of Dr Barakos [7].

The main purpose of discretizing the wing into a series of horseshoe vortexes is to produce a vortex field in which the downwash, or induced velocity, can be determined [7]. The induced velocity has significant implications for the design of wings and rotors which must be considered. These are:

- The change of effective angle of attack. Since an additional velocity component is added to the system in the downwards direction, the effective angle attack will change. This alters the lift curve slope.
- The change in effective angle of incidence changes the direction of the lift vector. Typically, in two-dimensional flows the lift vector is perpendicular to the flow direction, however, with the induced velocity the lift vector tilts backwards and the horizontal component of which is the induced drag.
- The above has implications on the stability of the aircraft.

Furthermore, it is necessary to account for the induced velocity as the flow used for the analysis is highly idealized and therefore can violate fundamental laws of fluid dynamics; accounting for the induced velocity helps satisfy the conservation of mass fundamental law [7]. The Biot-Savart law is used to calculate the induced velocity at the collocation point of each horseshoe vortex [7].

$$V_{A_m B_m}^{C_l} = \frac{\Gamma_l}{4\pi} \frac{\underline{r_1} \times \underline{r_2}}{|\underline{r_1} \times \underline{r_2}|^2} \left[\underline{r_0} \cdot \left(\frac{\underline{r_1}}{|\underline{r_1}|} - \frac{\underline{r_2}}{|\underline{r_2}|} \right) \right] \tag{41}$$

$$w_{A_m \infty}^{C_l} = \frac{\Gamma_l}{4\pi} \left[\frac{(z_C - z_A)_j + (y_A - y_C)_k}{(z_C - z_A)^2 + (y_A - y_C)^2} \right] \left[1 + \frac{(x_C - x_A)}{\sqrt{(x_C - x_A)^2 + (y_C - y_A)^2 + (z_C - z_A)^2}} \right] \tag{42}$$

$$w_{A_m \infty}^{C_l} = -\frac{\Gamma_l}{4\pi} \left[\frac{(z_C - z_B)_j + (y_B - y_C)_k}{(z_C - z_B)^2 + (y_B - y_C)^2} \right] \left[1 + \frac{(x_C - x_B)}{\sqrt{(x_C - x_B)^2 + (y_C - y_B)^2 + (z_C - z_B)^2}} \right] \tag{43}$$

Where subscripts l, m denote the collocation point under analysis with respect to neighbouring horseshoe vortex points, j, k denote the vector position, $\underline{r_i}$ the spatial vectors and x, y, z the coordinate of the spatial dimensions. Note the difference in sign for $w_{A_m \infty}^{C_l}$ and $w_{B_m \infty}^{C_l}$ which is due to the different circulation directions. The Biot-Savart equations can be solved and input in matrix form to produce a so-called influence coefficient matrix. This matrix is of scaler quantities which scale the circulation of each section in accordance with the neighbouring sections. The downwash, w, is equivalent to the influence coefficient matrix multiplied by the circulation vector, $\boldsymbol{w} = \boldsymbol{\eta \Gamma}$, where $\boldsymbol{\eta}$ is the influence coefficient matrix of the elements.

$$\boldsymbol{\eta} = \begin{bmatrix} \eta_{11} & \eta_{12} & \cdots & \eta_{1N} \\ \eta_{21} & \eta_{22} & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots \\ \eta_{N1} & \cdots & \cdots & \eta_{NN} \end{bmatrix} \tag{44}$$

$$\begin{bmatrix} w_1 \\ w_2 \\ \cdots \\ w_N \end{bmatrix} = \begin{bmatrix} \eta_{11} & \eta_{12} & \cdots & \eta_{1N} \\ \eta_{21} & \eta_{22} & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots \\ \eta_{N1} & \cdots & \cdots & \eta_{NN} \end{bmatrix} \begin{bmatrix} \Gamma_1 \\ \Gamma_2 \\ \cdots \\ \Gamma_N \end{bmatrix} \tag{45}$$

Where N, for this instance, denotes the number of panels across the span. From equation (45), this matrix clearly describes the interaction between each element and the resultant circulation.

In order to obtain a realistic physical solution from the highly idealized, unrealistic flow; the Kutta condition is used as a boundary condition. This is a viscous flow boundary condition which forces the potential flow to behave in a realistic manner and is based off observations that the flow on the upper and lower surfaces of an aerofoil leaves the trailing edge uniformly. From this, a hypothesis can be made that the velocity and pressure on both surfaces are equal at the trailing edge [6]. This resulted in the normal velocity component is always equal to zero, $\vec{V} \cdot \hat{n} = 0$. This is applied at the control point which results in equation (46) of the resultant of the downwash and airflow with the Kutta-condition:

$$V(\cos \boldsymbol{\alpha} + \sin \boldsymbol{\alpha}) + \boldsymbol{\eta \Gamma} = 0 \tag{46}$$

Therefore, the equivalent circulation of each panel: $\boldsymbol{\Gamma} = -V(cos\ \boldsymbol{\alpha} + \sin \boldsymbol{\alpha}) / \boldsymbol{\eta}$

$$\boldsymbol{\alpha} = \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \dots \\ \alpha_N \end{bmatrix} \tag{47}$$

Note, $-V(cos\ \boldsymbol{\alpha} + \sin \boldsymbol{\alpha})$ is a matrix with likewise dimensions to the influence coefficient matrix; each section will have a different angle of attack due to the induced velocity. Furthermore, the angle of attack of each section will change in bending and twist. Now, the lift distribution across the wing can be established, where the lift for a discretized system is:

$$L = \rho V \Gamma \Delta y \tag{48}$$

3.3 VLM Implementation into Matlab

Figure 3.3 depicts a flowchart which outlines the procedure required to execute Vortex Lattice method.

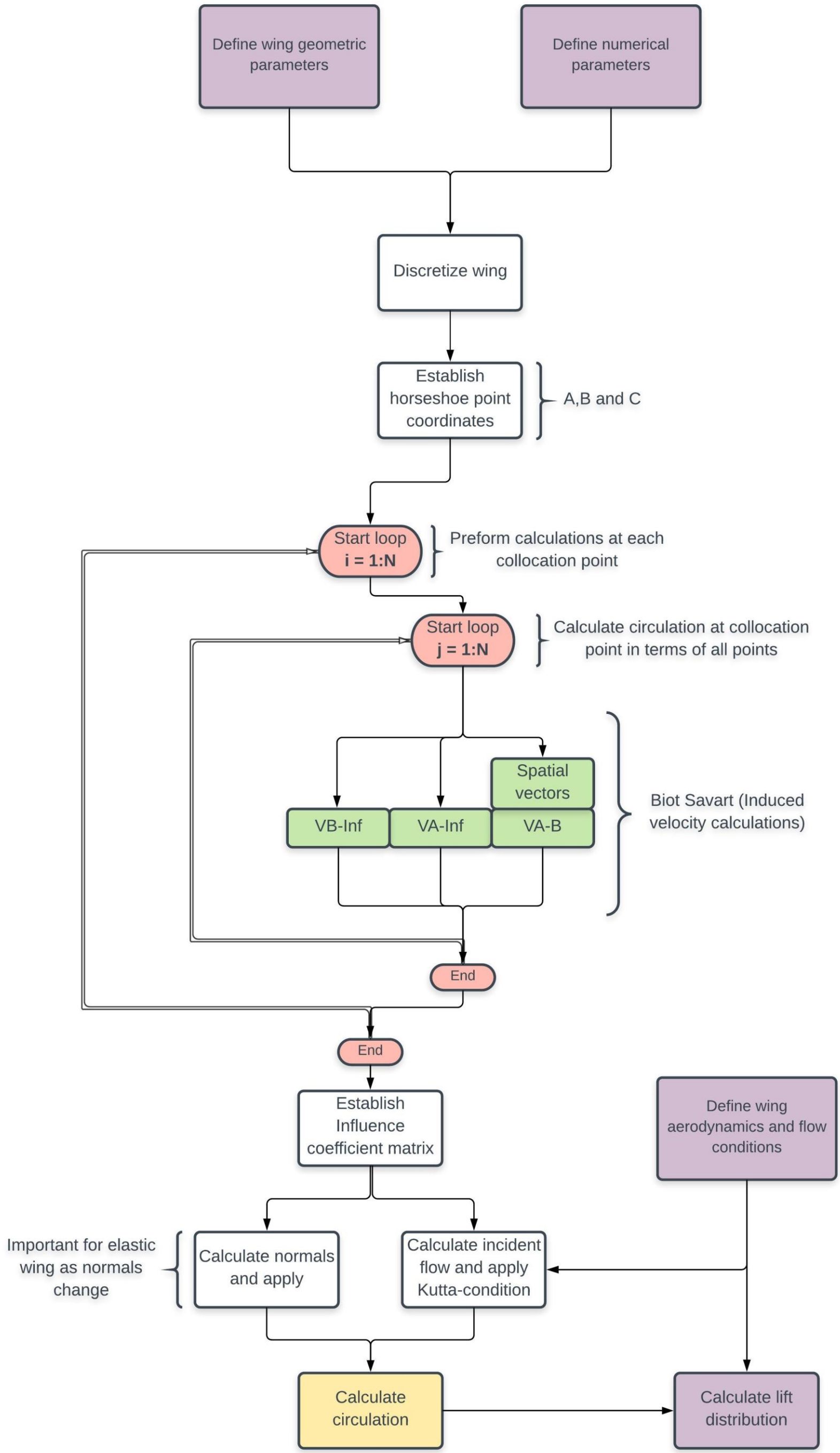


Figure 3.3 – Flowchart depicting the procedure required for executing Vortex Lattice Method

3.4 Validation of the Vortex Lattice Method code and numerical considerations

For validation, the Vortex Lattice method code was compared with the VLMpc code results of [4] for the Warren 12 wing where the aerodynamic and geometric characteristics are displayed in table 3.1. The geometric layout is displayed in figure 3.4.

Table 3.1 – Warren 12 wing characteristics

Characteristics	Values	Units
Aspect ratio	$2\sqrt{2}$	(-)
Sweep angle - mid chord	45	(^{o})
Wing area	$2\sqrt{2}$	(m^2)
Lift curve slope	2.743	($/rad$)

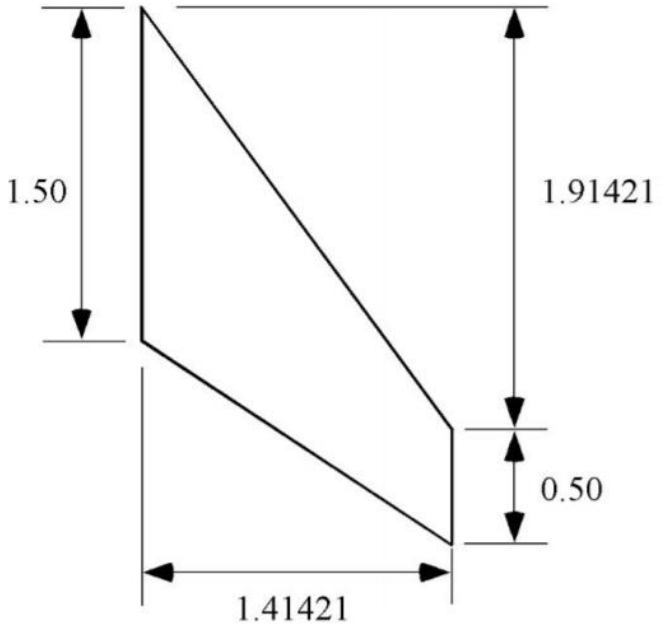


Figure 3.4 – Warren 12 wing geometric layout displaying values of root and tip chord with span. Image courtesy of Dr Barakos [7]

The spatial discretization of the wing, aft-swept for this example, can be seen in figure 3.5, where the bound and collocation points are depicted for the full span. Note, a course discretization is used for clarity of display.

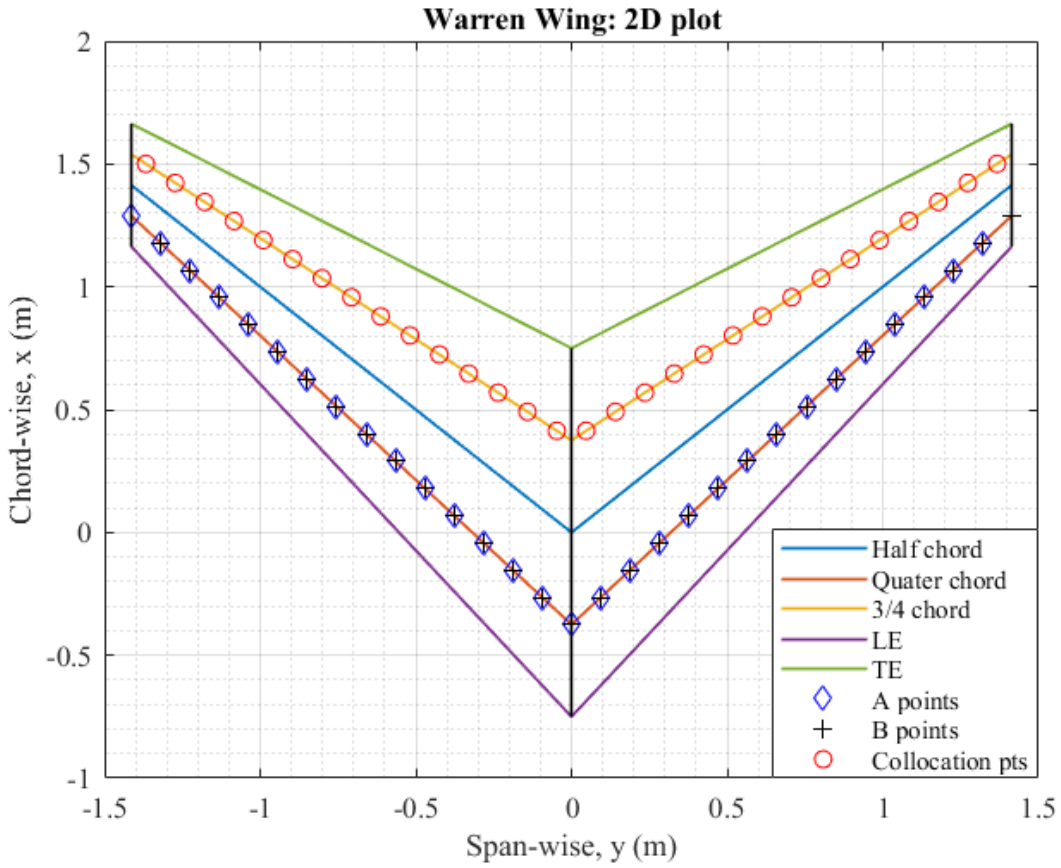


Figure 3.5 – Course example of spatial discretization of Warren 12 wing (aft-swept) displaying horseshoe vortex points in relation to characteristic chords

3.4.1 Validating results and effect of wing sweep

The results obtained by the code and VLMpc are displayed in figure 3.6, 3.7 and 3.8 for three different sweep configurations:

- Forward swept, 45^o.
- Straight.
- Aft swept, 45^o.

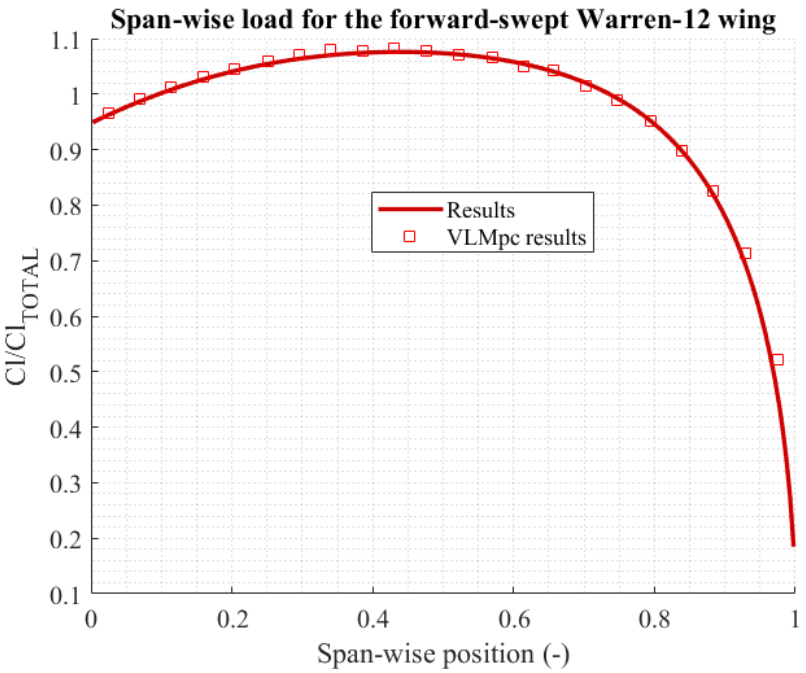


Figure 3.6 – Coefficient of lift for forward-swept Warren 12 wing compared with VLMpc code

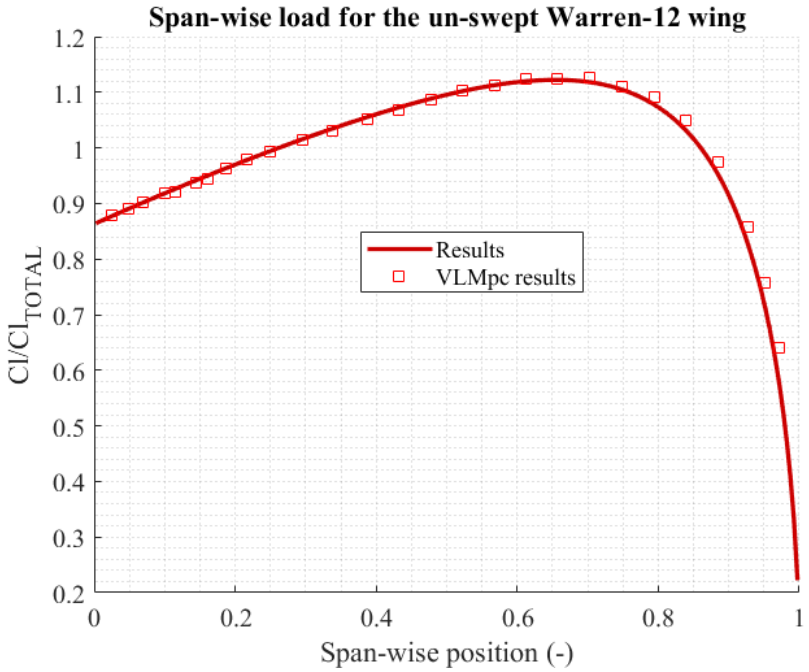


Figure 3.7 – Coefficient of lift for un-swept (straight) Warren 12 wing compared with VLMpc code

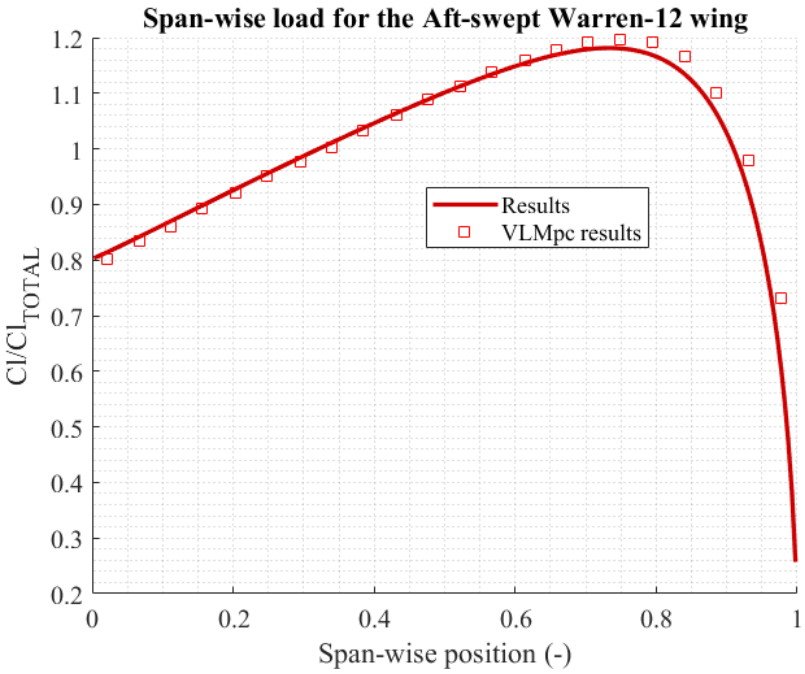


Figure 3.8 – Coefficient of lift for aft-swept Warren 12 wing compared with VLMpc code

The results compare well with the VLMpc code results and therefore the code produced has been validated and can be applied to the elastic wing trimmer model.

Effect of sweep

Wing sweep is typically employed to reduce compressibility effects at transonic speeds [1]. This reduction occurs for both forward and aft swept wings, however, an important difference between the configurations is the positional onset of flow separation due to the spanwise load distribution; flow separation initially occurs at the location of maximum lift coefficient [4]. For aft swept wings, this occurs initially near the wing tips since the load distribution increases towards the outboard sections, thus leading to detrimental stall and stability characteristics. Ailerons are typically positioned near the wing tip to maximize the control effectiveness and for the aft swept wing; flow separation near the tips introduces aileron reversal. This is avoided for forward swept wings where the spanwise load distribution is increased towards the inboard sections and has a significantly more evenly distributed load across the total span in which results in better stall and stability characteristics. With the above considered, the forward swept wing has an increased useful angle of attack range where the aircraft is can be considered stable which results in increased manoeuvrability and subsonic performances [8].

In terms of aeroelasticity, forward and aft swept wings are coupled in bending and torsion, however, both behave statically differently. In bending, consider a spanwise strip on each of the configurations, where X denotes the strip on the leading edge and Y on the trailing edge; for the forward-sweep, X moves upwards in relation to point Y and vice-versa occurs for the aft-sweep. This results in an increase and decrease of the effective angle of incidence for the configurations respectively [1]. This has significant effects on the divergence speed where the forward and aft have a decreased and increased divergence speed respectively [1]. Due to the decrease in divergence speed, i.e. aircraft wing failure, the forward-swept wings have not been readily employed in the past. However, with the advancement of composite laminate materials, the aeroelastic issue has been reduced significantly [5]; examples of aircraft that use the forward-swept configuration are the X-29 and the Sukhoi-47 [5].

3.4.2 Numerical considerations

Numerical considerations of the code were explored in order to select the most effective setting to be used in the trimmer method, in which were as follows:

- Number of panels, or horseshoe vortexes, across the span
- Cosine spacing of the panels.

The number and spacing of the panels is significant as it allows the rate of change of circulation in the spanwise direction to be captured more accurately. Since this rate of change is more prominent at the tips and roots, cosine spacing should be ideal as this concentrates of the panels at these regions.

Number of panels

The number of panels used for this analysis were 30, 100, and 400. figure 3.9 displays the results for different spatial discretion configurations in relation to the percentage difference at variant spanwise positions for the forward-swept Warren 12 wing.

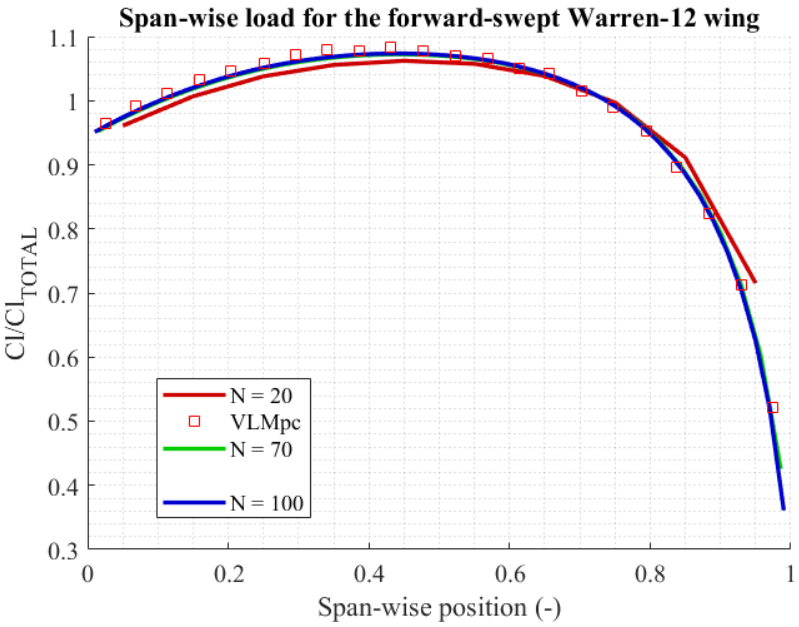


Figure 3.9 – Coefficient of lift distribution for forward-swept Warren 12 wing for different spacial discretization.

The number of panels is significant between a course discretization of $N = 20$ and $N = 70$, where the accuracy significantly increases. Furthermore, the accuracy does not increase significantly between the latter number of panel configurations, however, the distance towards the tip increases since the colocation points are in-between points A and B. The increase in accuracy will be due to the higher capture of the rate of change of circulation across the span, where when $N \to \infty$, the discretization will represent the true continuous system where the vortex strength is denoted as, $\gamma = \frac{\partial \Gamma}{\partial y}$ [7].

Cosine distribution

Figure 3.10 displays the spatial discretion with the cosine distribution for a course spacing to highlight the distribution for a forward-swept wing.

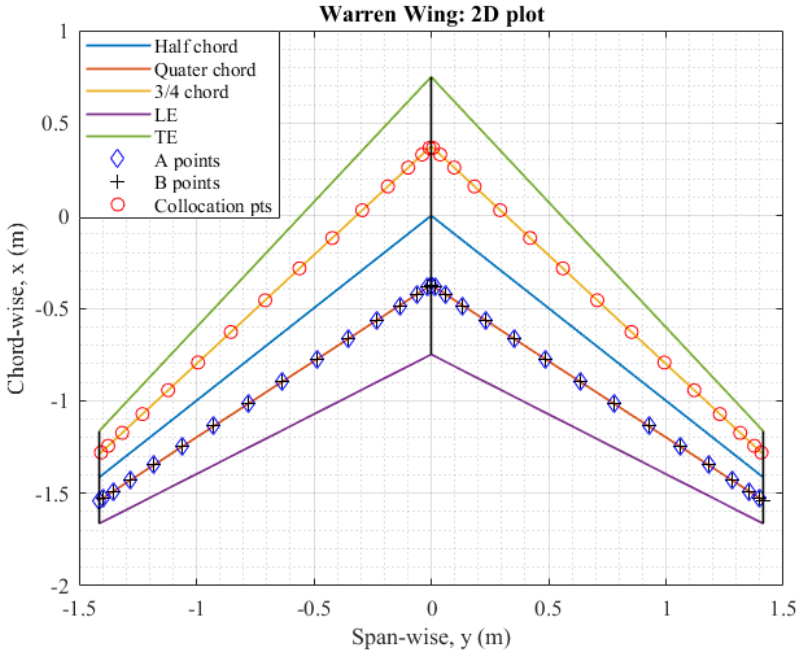


Figure 3.10 – Coarse cosine distribution for forward swept Warren 12 wing.

The cosine distribution can be computed through equation (49) below, where y_{space} is a vector ranging between $-\pi$ and π, OF the initial offset and S the wingspan.

$$y_{c,space} = OF + \frac{1}{2}(S - OF)\left(1 + y_{space}\right) \tag{49}$$

Initially considering a course number of panels ($N = 20$), the distribution allows the near tip region to be captured better. This was due to collocations points being placed nearer the tip, however, consequently the points are significantly courser in the mid-span region. When compared with the uniform distribution, the tip region was captured more accurately as the collocation points are congregated towards the tip, however, this resulted in the mid-board sections being significantly less accurate than the uniform distribution due to the courser

refinement in that section. Figure 3.11 displays results for the course spatial refinement of the cosine and uniform distribution.

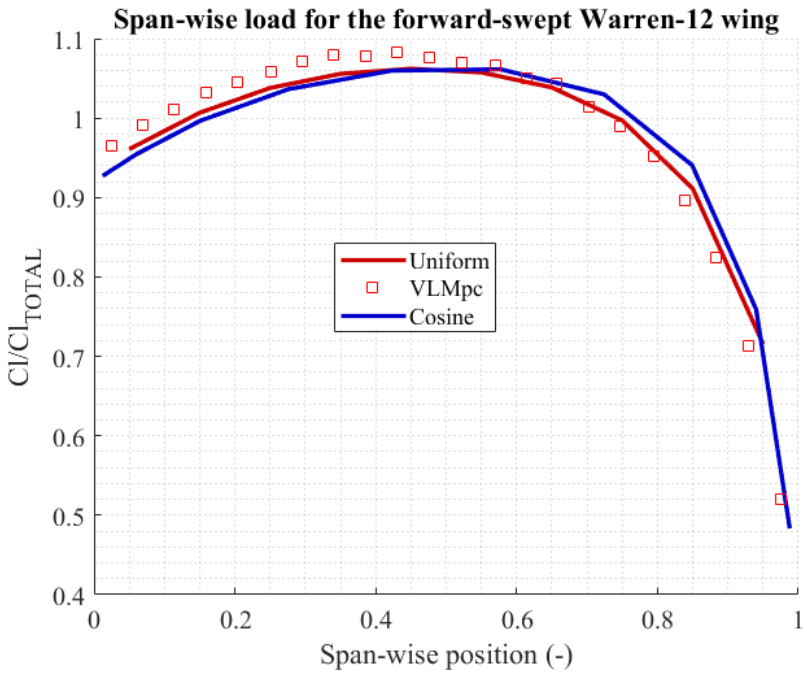


Figure 3.11 – Cosine against uniform distribution for forward-swept wing with N=20 panels.

From this, it concludes that the number of panels used is significantly more important than the spacial distribution as highly refining the tip regions sacrifices the accuracy of the mid regions. Furthermore, the reduced accuracy at the mid regions impinges on the accuracy of the root and tip due to the circulation interactions, thus concluding the use of cosine spacing non-sensical when using a course spatial discretion. With a sufficient number of panels ($N = 100$), no difference between the uniform and cosine distribution was found.

3.5 Trimmer model for elastic wing

The objective of the trimmer model is to simulate an aircraft in cruise with elastic wings. For cruise flight, the lift must be equivalent to the weight. A change of lift occurs due to the aeroelastic effects as a change in bending or torsion result in a change of angle of incidence, which in turn, results in a change of circulation across the span and therefore lift distribution, which follows another change in bending and torsion displacements. An iterative process is required, where a change of wing pitch is inputted at every iteration in an attempt to equalize the lift to the weight of the aircraft. Figure 3.12 displays the iterative procedure and table 3.2 displays the aircraft parameters used for the simulation.

The total procedure is displayed in figure 3.13 and 3.14 as detailed flowcharts of the initialization and trim loop respectively.

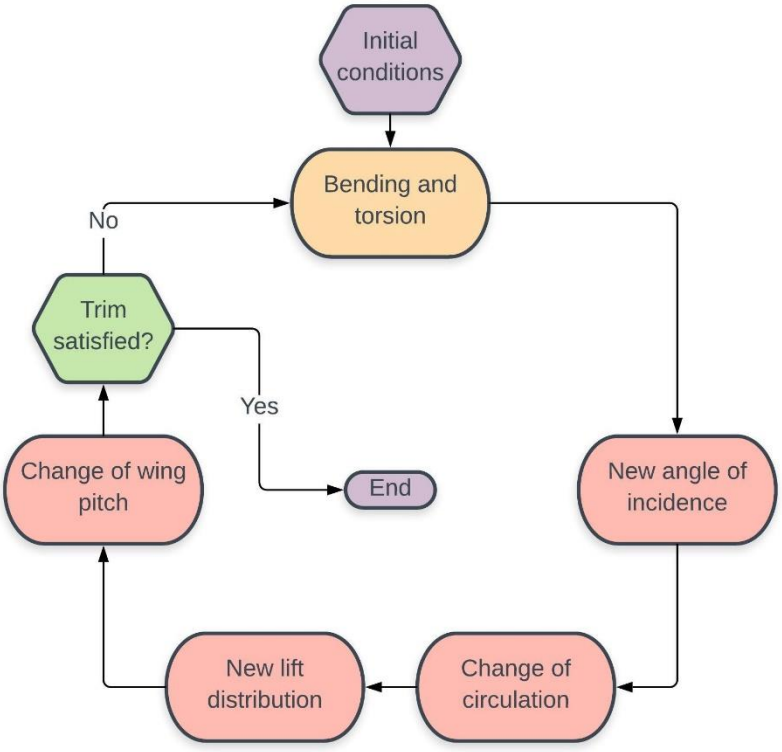
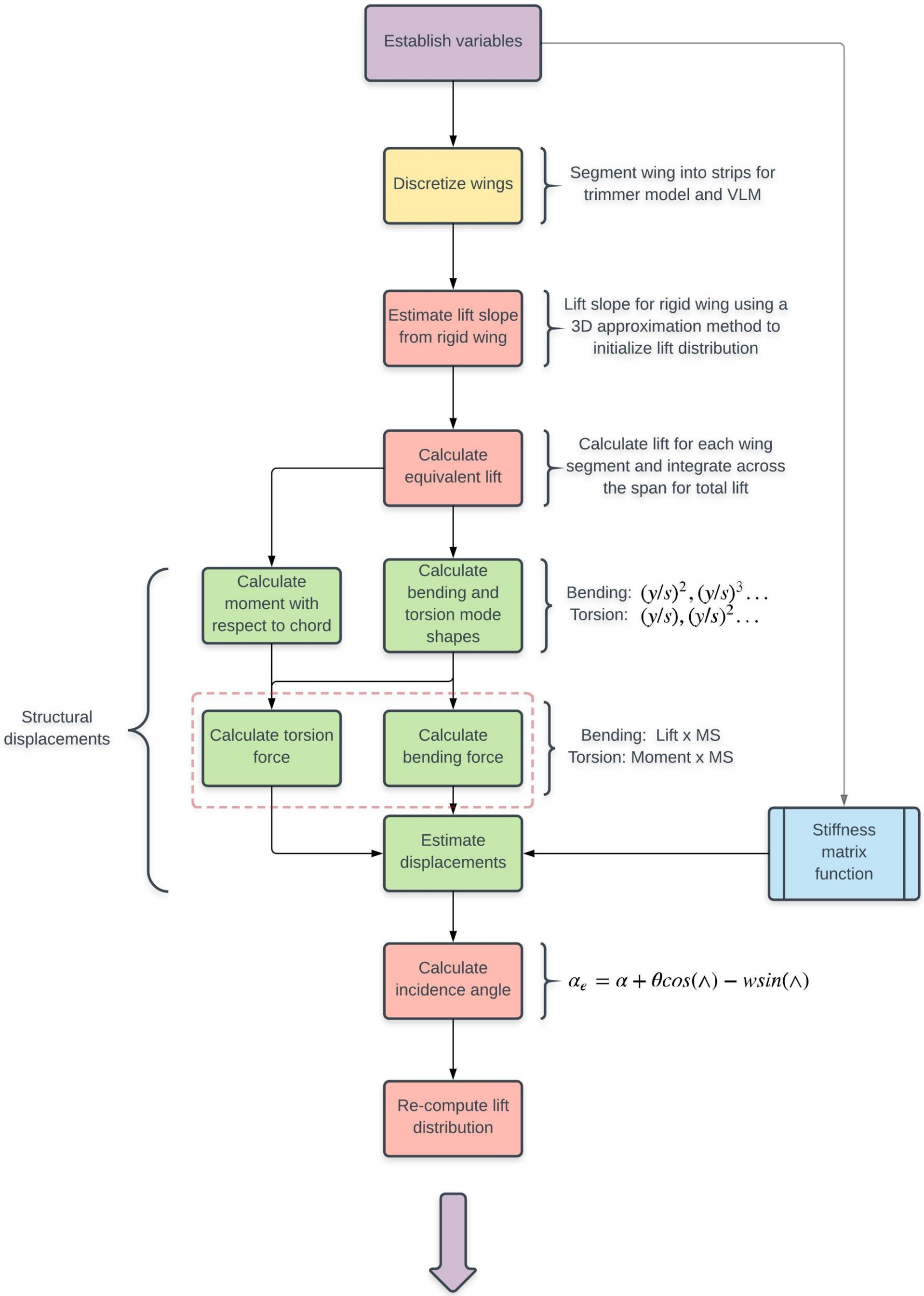


Figure 3.12 – Overview of trimmer model for elastic wing aircraft

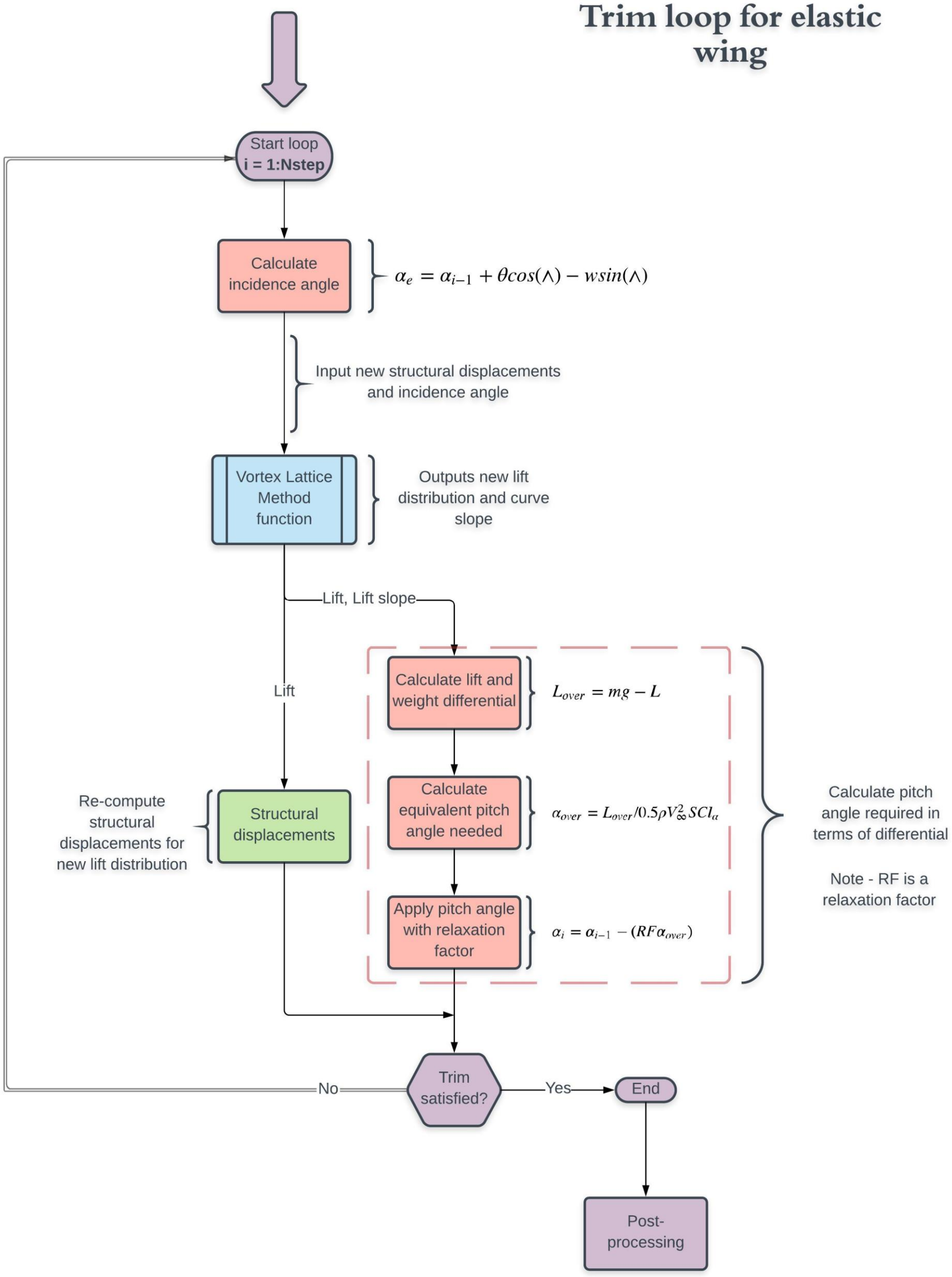
Table 3.2 – Aircraft parameters in SI units

Physical parameters	Values	Units
Chord	1.5	m
Wing semi-span	6	m
Sweep	0	^{o}
Taper ratio	0.4	(-)
Weight	98,100	N
Torsional rigidity	2×10^6	Nm^2
Flexural rigidity	5×10^5	Nm^2
Aerodynamic parameters		
Freestream velocity	150	m/s
Altitude	6096	m
Freestream density	0.614655	kg/m^3

Initializing trimmer model for elastic wing



Trim loop for elastic wing



3.6 Results of the trimmer model for the elastic wing

The aircraft was simulated with the test case displayed in table 3.2 and for a forward- and aft-sweep case of 30^o each with $N = 100$ with cosine spacing and a relaxation factor of 0.1. The un-swept, aft- and forward-swept are displayed in figure 3.15, 3.16 and 3.17 respectively.

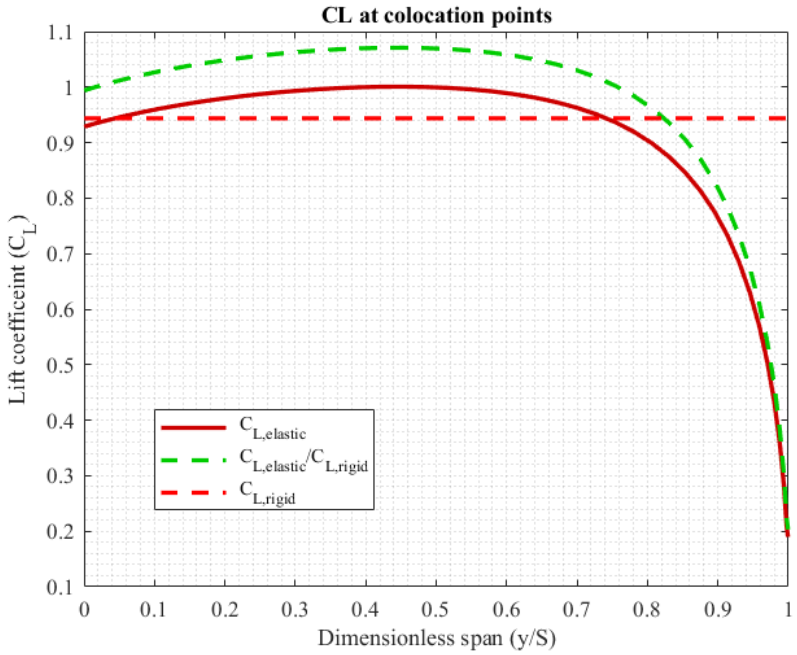


Figure 3.15 – Elastic and rigid coefficient of lift comparison of un-swept wing

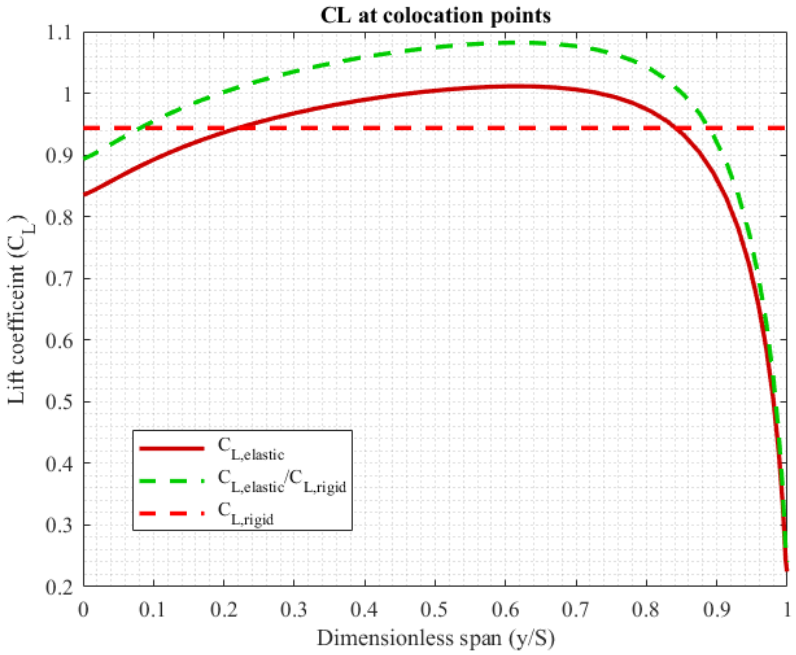


Figure 3.16 – Elastic and rigid coefficient of lift comparison of aft-swept wing

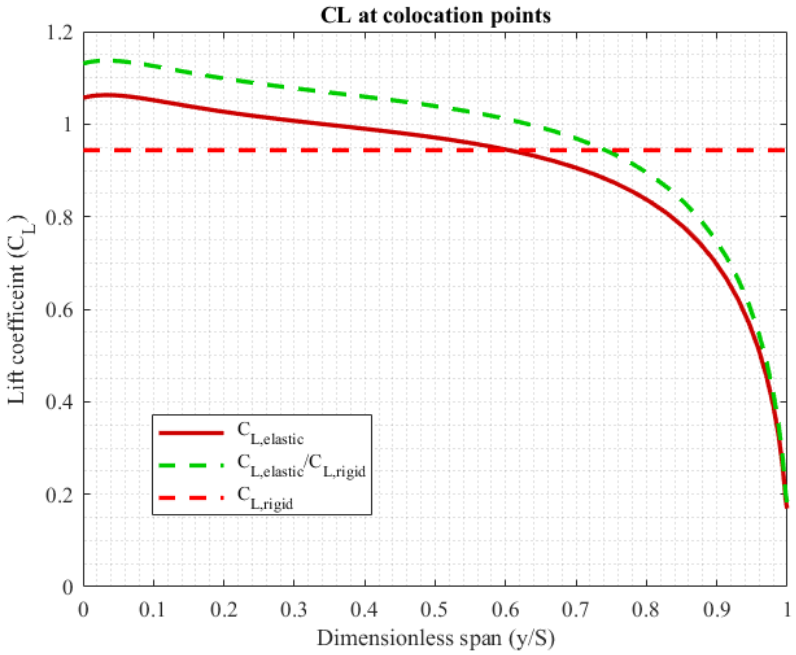


Figure 3.17 – Elastic and rigid coefficient of lift comparison of forward-swept wing

The coefficient of lift distribution across the span was taken at the colocation points, C, from the Vortex Lattice method function. $C_{L,rigid}$ is the total coefficient of the wing and therefore is used as a reference for the occurrence of increase or decrease of lift coefficient under elastic effects. Considering the magnitude of lift coefficient for each wing configuration, it can be observed that:

- For all configurations, the lift coefficient decreases near the tip, however, this is due to the pressure equalization at the wing tip. Note, this should tend to zero as there is zero pressure differential at the tip, however, the figures are a plot of colocation points, where the furthest outboard point is not on the wing tip.
- The forward-swept wing lift coefficient significantly increases at the inboard sections due to elastic effects.
- The aft-swept wing lift coefficient significantly increases on the outboard sections.
- The un-swept wing lift coefficient increases in the mid-board sections.

The static aeroelastic effects of each wing configuration are analogous with the rigid span-wise load distribution of each configuration where the spanwise load distribution of the rigid wings for the forward, aft and un-swept configurations are primarily at the inboard, outboard and mid-

board respectively. Therefore, a hypothesis can be made that the static aeroelastic effects augment the wing configuration characteristics. With the aeroelastic augmentation, stall on the primarily loaded regions is a significant risk and should be accounted for in the design procedure; for example, the aft-swept wing will have further detriment the stall and stability characteristics due to this augmentation. Figures 3.18 → 3.21 displays the aeroelastic angle, torsion angle, and bending displacement respectively over the dimensionless span, where the aeroelastic angle is the summation of the angles due to pitch, bending and torsion.

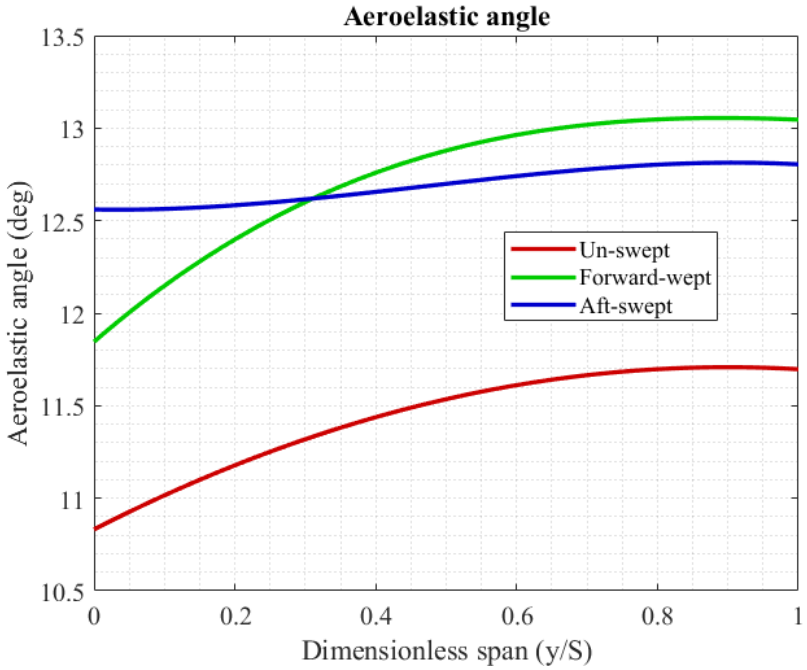


Figure 3.18 – Aeroelastic angle for all configurations across dimensionless span.

The angle at the root of the wing is the overall pitch angle since no bending or twist occur at the root. Observing the outboard region, the forward swept wing has the highest resultant angle, followed by the aft-swept and un-swept configuration. This result is as expected as the bending and torsion of the forward swept wing, as mentioned previously, both increase the angle of attack. This can be clarified by observing the aeroelastic angle equation:

$$\alpha_e = \alpha_p + \theta \cos(\Lambda) - w sin(\Lambda) \tag{50}$$

Where for a forward swept, $\Lambda < 0$, which results in $(-\sin(\Lambda)) > 0$. This coupling between bending and torsion is the reason for lower divergence speeds of the forward swept wing. Furthermore, the opposite occurs for the aft-swept wing as the bending reduces the aeroelastic angle as $\Lambda > 0$, resulting in higher divergence speeds. For the un-swept wing, bending has no effect on the effective angle of attack since $\Lambda = 0$ and therefore $\sin(\Lambda) = 0$.

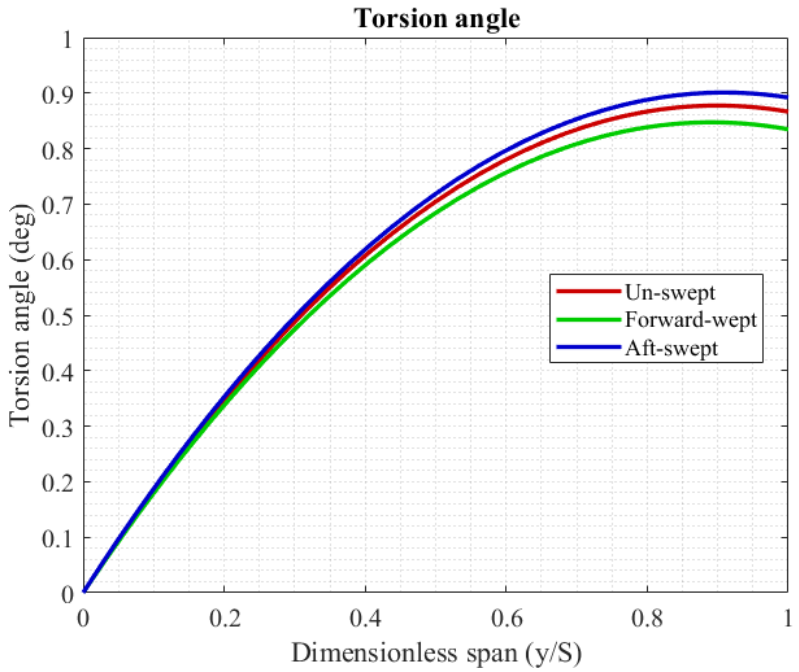


Figure 3.19 – Torsion angle for all configurations across dimensionless span

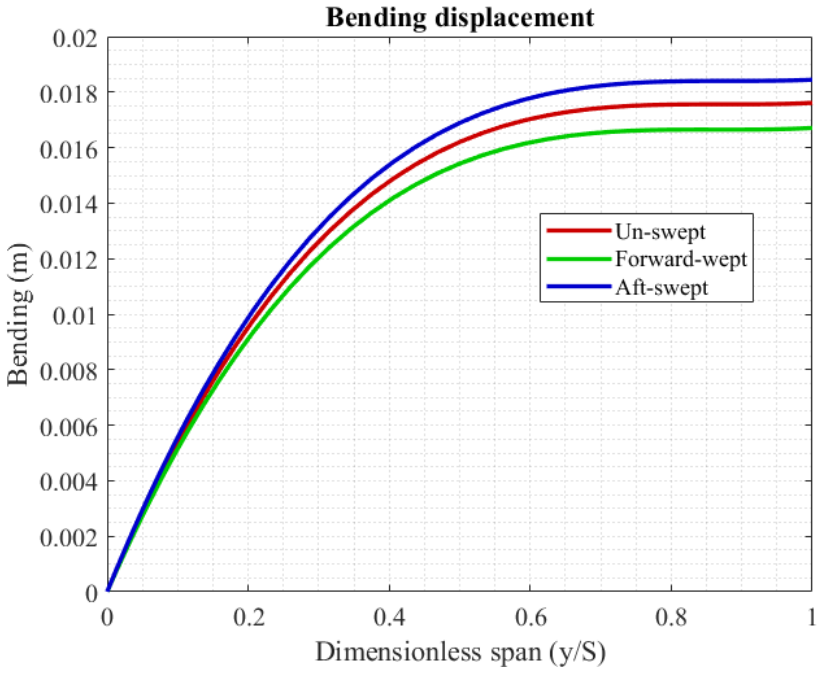


Figure 3.20 – Bending displacements for all configurations across dimensionless span

From figure 3.19 and 3.20 of the torsion angles and bending displacements, it can be observed that the forward-swept has less magnitudes than its counterparts despite having the highest aeroelastic angle; this is due to the decrease in spanwise loading on the outboard sections. The opposite occurs for the aft-swept wing as the spanwise load is primarily on the outboard sections, however, the aft-swept wing has a higher elastic angle than the un-swept wing due to the bending also contributing to the effective angle of attack. The above highlights the aeroelastic issues with wing sweep due to the coupling of bending and torsion modes.

Chapter 4:

Predicting flutter

4.1 Introduction and objectives

Flutter is an aeroelastic phenomenon in which entails the coupling of vibrational modes which results in the wing extracting energy from the freestream. This leads to self-excited, unstable vibrations of the wing which can potentially lead to wing failure, known as divergence [1].

There are variant types of flutter which stem from different effects; the flutter this assignment will consider is known as the classical flexural and torsion flutter. This occurs when aerodynamic or inertial coupling are present which derive from a spatial difference between the elastic axis and aerodynamic centre (AC) or centre of gravity (CG) of the wing respectively. Figure 4.1 displays a two-dimensional aerofoil with the aerodynamic centre, centre of mass and elastic axis positions, where the centre of mass lies on the mid-chord for this diagram. Base image courtesy of Wright and Cooper [1].

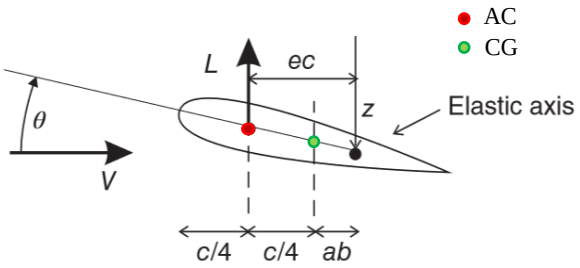


Figure 4.1 – Aerofoil with elastic axis, aerodynamic centre and centre of mass. Base image from Wright and Cooper [1]

An important design parameter for wings or rotors is the so-called flutter speed which is where the self-excited, unstable vibrations occur due to the flight speed or disturbances [1], [9]. The objective of this assignment is to calculate the flutter speed of the wing from with different mass configurations, "5" and "7e", from the NACA TN paper which are varied across spanwise locations. These masses have a different resultant weight and position in relation to the elastic axis of the wing; this significantly effects the magnitude of the flutter speed of the wing which will be discussed. Furthermore, unsteady aerodynamics will be covered in the prevailing

section as this is crucial for analysing flutter as if it is not included, i.e. assuming steady or quasi-steady aerodynamics conditions, the flutter speed will be computed as $0\ m/s$, which is unphysical [9].

4.2 Modelling unsteady aerodynamics

4.2.1 Steady, Quasi-steady and Unsteady aerodynamics

Steady aerodynamics is where the forces and moments are constant with time and are calculated from a fixed aerofoil profile relative to the aircraft wind axes. Quasi-steady aerodynamics is where frequency dependent effects are not considered and an aerofoil undergoing bending and twist is calculated instantaneously from the new geometric and flow characteristics [1]. Considering frequency dependent effects introduces unsteady aerodynamics, where frequency dependent effects arise from the dynamic motion of the aerofoil [1]. This provides considerations of the instantaneous changes and motion of the aerofoil in which will be considered harmonic for this assignment. There are two key functions in analysing the unsteady aerodynamic properties: the Wagner function, in which analyses the general motion in the time domain, and the Theodorsen's function, which differs in approach where the frequency domain is used instead [1]. For flutter calculations, the frequency domain is commonly used to model the unsteady behaviour and therefore the Theodorsen's function will be used.

Effectively, these functions consider the time taken for the circulation re-establish and produce the equivalent lift of the quasi-steady case, if it is ever met, which introduces phase lag and different amplitudes in the Theodorsen's function [1]. The phase lag and amplitude change are functions of the so-called reduced frequency and frequency parameter which, by definition, are number of cycles related to the time taken for the travelling to cross the chord or semi-chord:

$$k = \frac{\omega b}{V} \tag{51}$$

$$v = \frac{\omega c}{V} \tag{52}$$

Where, ω is the frequency of oscillations, b, c the semi and total chord, and V the corresponding velocity. The Theodorsen's function is modelled using modified Bessel functions of the second kind [9], where $F(k)$ and $iG(k)$ are the isolated real and imaginary part of the function respectively.

$$C(k) = F(k) + iG(k) = \frac{K_1(ik)}{K_0(ik) + K_1(ik)} \tag{53}$$

To calculate the lift and moment of the aerofoil, the contributing force can be split into two categories, circulatory and non-circulatory terms, where circulatory encompasses the contributed force due to circulation and the non-circulatory to the apparent inertia forces from the displaced air by the aerofoil motion [1].

The lift and moment of a symmetric aerofoil can be calculated as:

$$L = \pi\rho b^2\left(\ddot{z} + V\dot{\theta} - ab\ddot{\theta}\right) + 2\pi\rho VbC(k) + \left(\dot{z} + V\theta + b\left(\frac{1}{2} - a\right)\dot{\theta}\right) \tag{54}$$

$$M = \pi\rho b^2\left[ab\ddot{z} - Vb\left(\frac{1}{2} - a\right)\dot{\theta} - b^2\left(\frac{1}{8} + a^2\right)\ddot{\theta}\right] \ldots$$

$$+2\pi\rho Vb^2\left(a + \frac{1}{2}\right)C(k)\left[\dot{z} + V\theta + b\left(\frac{1}{2} - a\right)\dot{\theta}\right] \tag{55}$$

Where z and θ are the bending and torsion displacements which are represented in oscillatory harmonic motion:

$$z = \bar{z}e^{i\omega t} \tag{56}$$

$$\theta = \bar{\theta}e^{i\omega t} \tag{57}$$

Now, applying the time differentials, substituting the complex form of the Theodorsen's yields the lift and moment of the aerofoil in terms of aerodynamic oscillatory derivatives.

$$L = \rho V^2 \left(L_z z + L_{\dot{z}} \frac{b\dot{z}}{V} + L_\theta b\theta + L_{\dot{\theta}} \frac{b^2\dot{\theta}}{V} \right) \tag{58}$$

$$M = \rho V^2 \left(M_z bz + M_{\dot{z}} \frac{b^2\dot{z}}{V} + M_\theta b^2\theta + M_{\dot{\theta}} \frac{b^3\dot{\theta}}{V} \right) \tag{59}$$

In matrix form,

$$\begin{Bmatrix} L \\ M \end{Bmatrix} = \rho V \begin{bmatrix} bL_{\dot{z}} & b^2 L_{\dot{\theta}} \\ b^2 M_{\dot{z}} & b^3 M_{\dot{\theta}} \end{bmatrix} \begin{Bmatrix} \dot{z} \\ \dot{\theta} \end{Bmatrix} + \rho V^2 \begin{bmatrix} L_z & bL_\theta \\ bM_z & b^2 M_\theta \end{bmatrix} \begin{Bmatrix} z \\ \theta \end{Bmatrix} \tag{60}$$

This is the basis to the aerodynamic matrices for an aeroelastic system of a wing.

4.3 Mass balancing

The concept of mass balancing is important for aeroelastic analysis as it directly effects the aeroelastic behaviour of a wing or rotor and therefore the flight speed limitations [1]. Coupling is present if the centre of gravity is not aligned with the elastic axis, however, this coupling can be used to increase the flutter speed. Figure 4.2 displays the change of flutter speed of a baseline model with respect to change of centre of gravity position, where the elastic axis is positioned at $0.48c$. Image *and* results courtesy of Wright and Cooper [1].

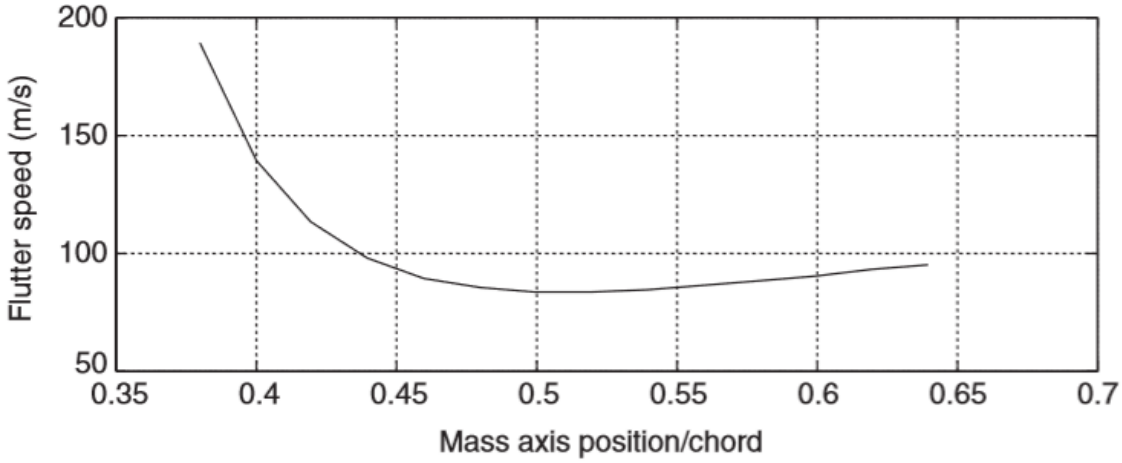


Figure 4.2 – Change of flutter speed in accordance to centre of gravity position. Image and results courtesy of Wright and Cooper [1].

From this, the advantages of mass-balancing are clear, where the most significant benefit occurs when the centre of mass is forward of the elastic axis. This technique is used to increase the flutter speed in many applications from control-surfaces to entire wings, where the latter is achieved by placing the engines far forward of the wing [9]. To highlight the concept, consider a torsional vibration which will tend to increase and decrease the angle of attack of the wing about the elastic axis, which in turn, introduces aerodynamic to inertial coupling due change of aerodynamic forces caused by the variant angle of attack. If the weight vector is forward of the elastic axis, the force exerted by the weight will counteract the increase of angle of attack part of the vibration, and, therefore, act as a damper to the motion. In terms of the assignment, the exact principles are employed in the NACA TN paper where the different classification of weights is varied across the chord line and therefore changing the effective centre of mass of the wing.

4.4 Calculating flutter

The general form of the classical flutter equation [9] is displayed in equation (61) where $DEL, D,$ and C are the structural matrices from chapter 2.

$$-\pi\rho b^2 \begin{bmatrix} \boldsymbol{DEL} & ab\boldsymbol{C} \\ ab\boldsymbol{C^T} & b^2\left(a^2+\frac{1}{8}\right)\boldsymbol{D} \end{bmatrix} \ddot{\boldsymbol{q}} \ldots \quad (61)$$

$$-\pi\rho bU \begin{bmatrix} 2C_{theo}(k)\boldsymbol{DEL} & -b(1+2\left(\frac{1}{2}-a\right)C_{theo}(k)\boldsymbol{C} \\ 2b\left(\frac{1}{2}+a\right)C_{theo}(k)\boldsymbol{C^T} & b^2\left(\frac{1}{2}-a\right)(1-2\left(\frac{1}{2}+a\right)C_{theo}(k)\boldsymbol{D} \end{bmatrix} \dot{\boldsymbol{q}} \ldots$$

$$-\pi\rho bU^2 \begin{bmatrix} \boldsymbol{0} & -2C_{theo}(k)\boldsymbol{C} \\ \boldsymbol{0} & -b(1+2a)C_{theo}(k)\boldsymbol{D} \end{bmatrix} \boldsymbol{q}$$

From this the characteristic polynomial can be derived to establish the flutter frequencies and damping, which in turn, allows the flutter speed and divergence speed to be calculated [9].

$$\emptyset(s) = |s^2M - sA^* + K - B^*| = 0 \quad (62)$$

Where s is the eigenvalue of the characteristic polynomial. The flutter and divergence speeds can be set by considering the conditions. For the flutter condition, the real and imaginary part of the eigenvalue is equivalent to zero and greater than zero respectively [9]. Whereas, for the divergence speed both real and imaginary parts are equivalent to zero [9], i.e. no oscillations as the system has diverged.

$$Flutter\ condition\colon\ \ \emptyset(i\omega) = 0 \quad (63)$$

$$Divergence\ condition\colon\ \emptyset(s) = 0 \quad (64)$$

Executing the determinant, normalizing by dynamic head, $\frac{1}{2}\rho U^2$, and substituting in equation ? results in the unsteady aerodynamics matrix, Q, in which for the flutter condition is displayed

below, where the change of sign and imaginary parts, j, arises from the powers of the eigenvalue, $j\omega$, where $j^2 = -1$.

$Q =$

$$2\pi bk^2 \frac{1}{\omega^2} \begin{bmatrix} \boldsymbol{DEL} & ab\boldsymbol{C} \\ ab\boldsymbol{C^T} & b^2 \left(a^2 + \frac{1}{8}\right)\boldsymbol{D} \end{bmatrix} \ldots \tag{65}$$

$$-2\pi ki \frac{1}{\omega} \begin{bmatrix} 2C_{theo}(k)\boldsymbol{DEL} & -b(1 + 2\left(\frac{1}{2} - a\right)C_{theo}(k)\boldsymbol{C} \\ 2b\left(\frac{1}{2} + a\right)C_{theo}(k)\boldsymbol{C^T} & b^2\left(\frac{1}{2} - a\right)(1 - 2\left(\frac{1}{2} + a\right)C_{theo}(k)\boldsymbol{D} \end{bmatrix} \dot{\boldsymbol{q}} \ldots$$

$$-2\pi b \begin{bmatrix} \boldsymbol{0} & -2C_{theo}(k)\boldsymbol{C} \\ \boldsymbol{0} & -b(1 + 2a)C_{theo}(k)\boldsymbol{D} \end{bmatrix} \boldsymbol{q}$$

$$Q = \frac{1}{\omega^2}A^* + \frac{1}{\omega}B^* + C^* \tag{66}$$

Where A^*, B^* and C^* are the apparent mass, aerodynamic damping and stiffness matrices respectively; where the aerodynamics are classified in terms of structural terms as the behaviour they impart on the wing is analogous to their structural counterparts. In order to obtain the flutter frequencies and damping, relation of the aerodynamics to the structural is required; this is executed by arranging into the form:

$$A\ddot{q} + Eq = A^*\ddot{q} + B^*\dot{q} + C^*q \tag{67}$$

Assuming harmonic motion likewise to the oscillatory derivatives,

$$q = \bar{q}e^{i\omega t} \tag{68}$$

$$\dot{q} = i\omega\bar{q}e^{i\omega t} \tag{69}$$

$$\ddot{q} = -\omega^2\bar{q}e^{i\omega t} \tag{70}$$

Therefore, the equation results in:

$$(A\omega^2 + E)\bar{q}e^{i\omega t} = (-\omega^2 A^* + i\omega B^* + C^*)\bar{q}e^{i\omega t}$$

$$(A\omega^2 + E)\bar{q} = (-\omega^2 A^* + i\omega B^* + C^*)\bar{q} \tag{71}$$

Where, $(-\omega^2 A^* + i\omega B^* + C^*)\bar{q}$ is the aerodynamic response matrix. Un-normalizing by dynamic head and combining with equation ? and rearranging results in:

$$\left[\frac{1+ig}{\omega^2}\right]\bar{q} = E^{-1}\left[A + \frac{1}{2}\frac{\rho b^2}{k^2}\hat{A}(jk)\right]\bar{q} = \tau \tag{72}$$

Where $\hat{A}$ is the aerodynamic response matrix and $\left[\frac{1+ig}{\omega^2}\right]$ is the eigenvalue. The flutter frequency and damping can be determined from the eigenvalue:

$$\omega = \sqrt{\frac{1}{\Re(\tau)}} \tag{73}$$

$$g = \frac{\omega^2\Im(\tau)}{i} \tag{74}$$

4.5 Results of flutter calculations

The frequency and damping of the system were obtained over a range of reduced frequencies, $k = 0.01 \rightarrow 1.5$. In accordance to the flutter condition, as mentioned previously, the flutter speed was obtained when the imaginary part of the eigenvalue was equivalent to zero. The structural matrices were executed with three bending modes, $N = 3$, and two torsional modes, $M = 2$. Table 4.1 displays the mass parameters and relation to the elastic axis of the wing.

Table 4.1 – Mass parameters

	Mass 5	Mass 7e
W_{mass}/W_{wing}	0.636	0.954
e_w	0.687 b	0.034 b
I_{mass}/I_{wing}	2.68	1.56

Where e_w is the distance between the elastic axis and centre of gravity of weight in relation to semi-chord and a positive value indicates the weight is aft of the elastic axis. For mass "5", figures 4.3 → 4.5 display the normalized flutter speed over spanwise positions, damping and frequency for corresponding velocity at the minimum flutter speed.

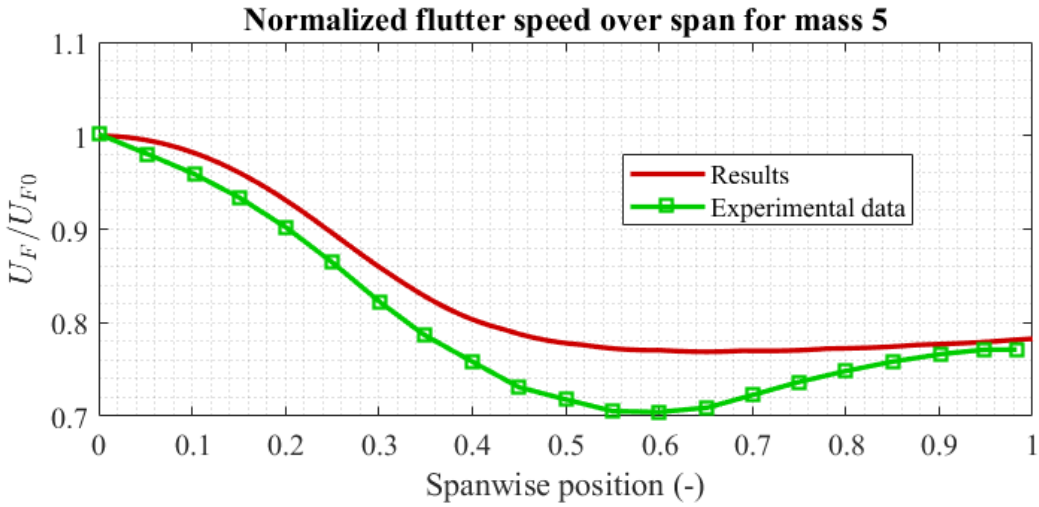


Figure 4.3 – Normalized flutter speed over span locations for mass "5", compared with experimental data from NACA TN paper [2]

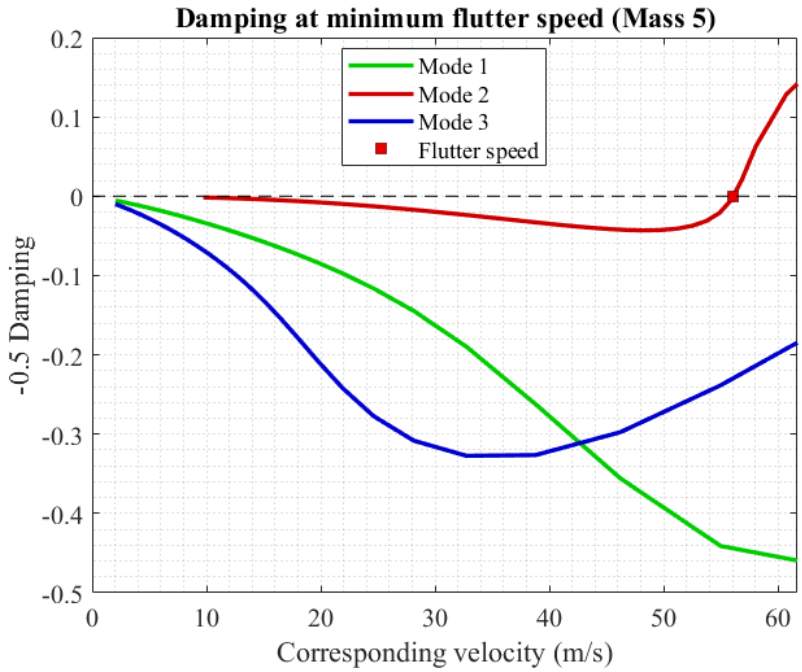


Figure 4.4 – Damping for corresponding minimum flutter speed, which occurred at $y/S = 0.71$.

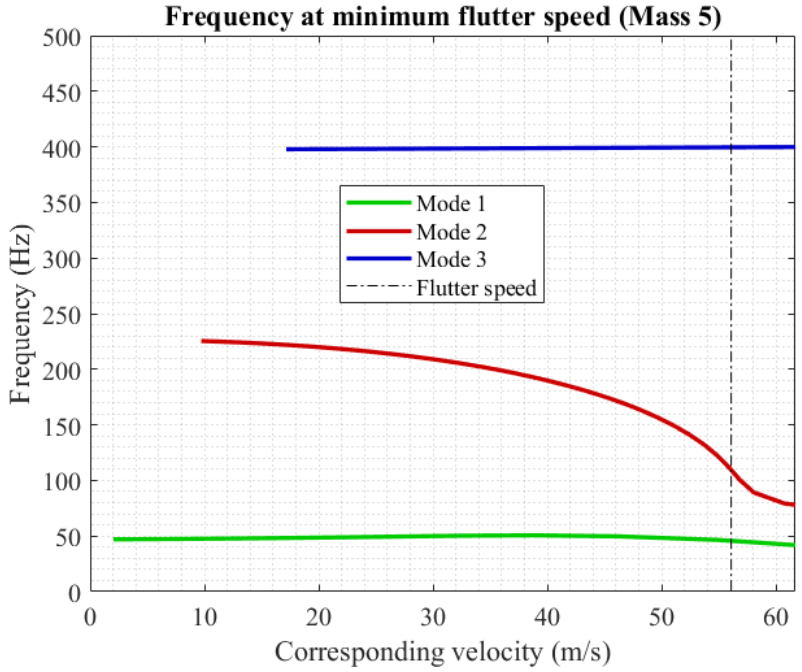


Figure 4.5 – Frequency for corresponding minimum flutter speed which occurred at $y/S = 0.71$

Mass “5” is a wing-weight configuration where the centre of gravity is pushed aft of the elastic axis and the minimum flutter speed occurs on the second bending mode at $U_F = 56.07\ m/s$ which occurred at a normalized spanwise distance of $y/S = 0.71$. The magnitude of the gradient around the flutter point is significant as it dictates how quickly flutter occurs [1]. From figure 4.4, hard flutter will occur as the damping sharply increases at approximately $54\ m/s$. Below this velocity, the wing is significantly damped, therefore, this change will appear suddenly in a pilot’s perspective [9], which is highly unsafe as there is a lack of warning beforehand.

For mass “7e”, figures 4.6 → 4.8 display the normalized flutter speed over spanwise positions, damping and frequency for corresponding velocity at the minimum flutter speed.

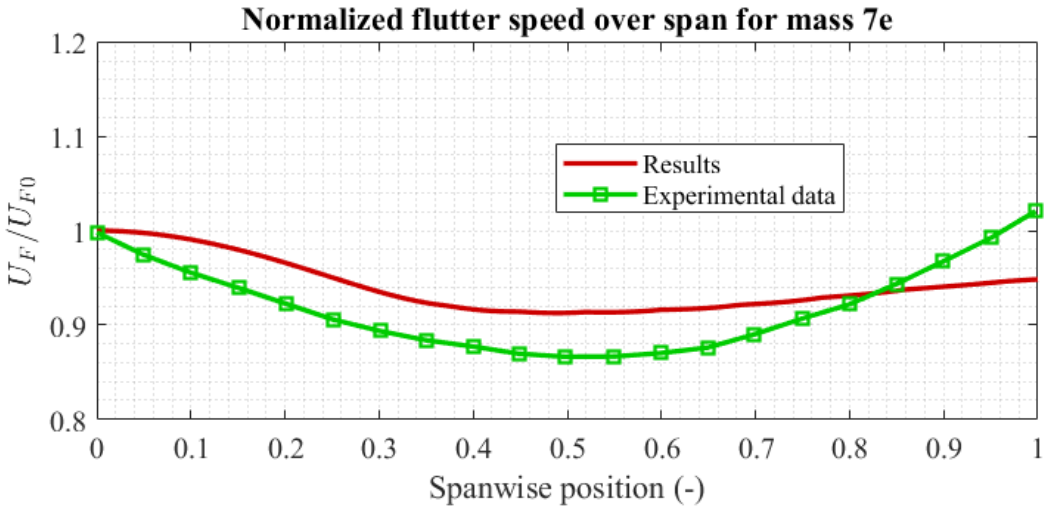


Figure 4.6 – Normalized flutter speed over span locations for mass “7e”, compared with experimental data from NACA TN paper

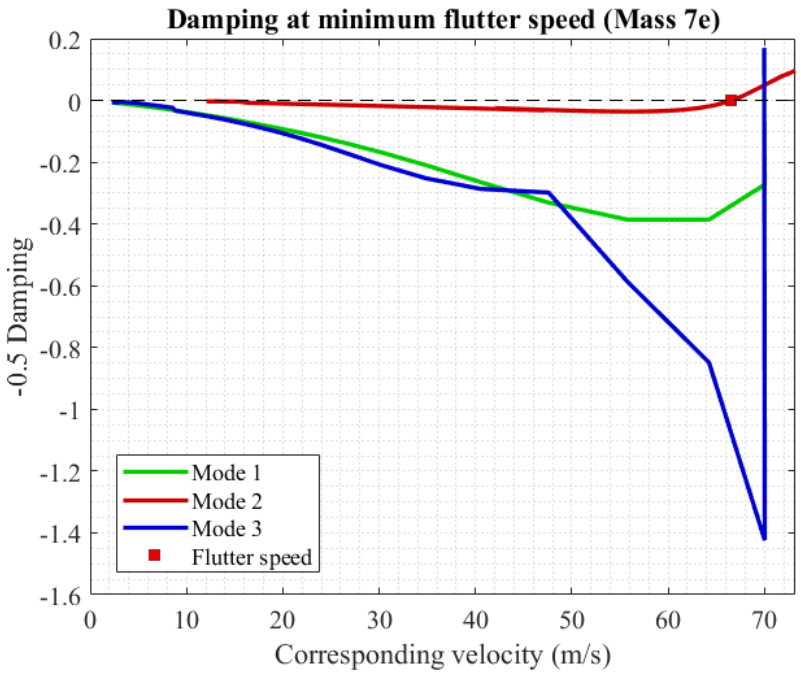


Figure 4.7 – Damping for corresponding minimum flutter speed, which occurred at $y/S = 0.54$.

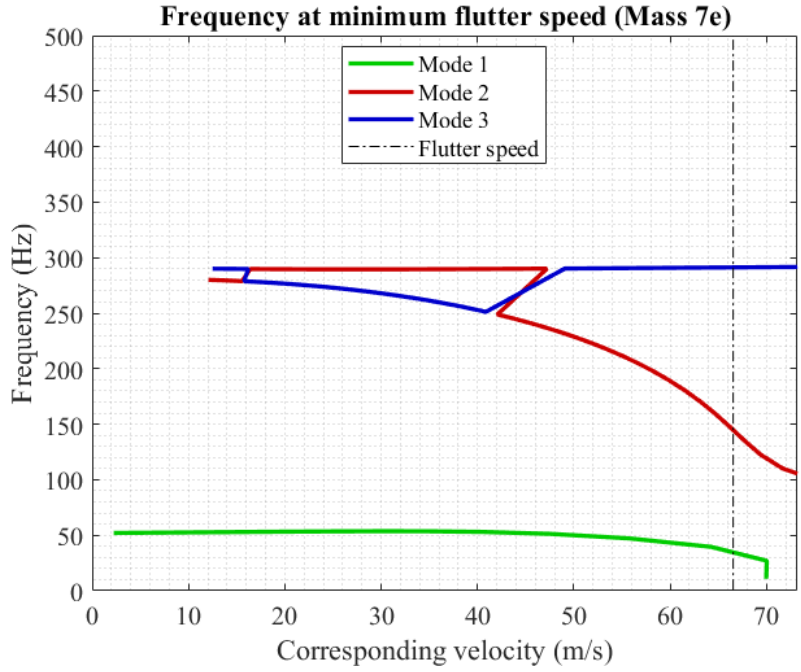


Figure 4.8 – Frequency for corresponding minimum flutter speed which occurred at $y/S = 0.54$

Mass “7e” is a wing-weight configuration where the centre of gravity is pushed significantly towards and slightly aft of the elastic axis. The minimum flutter speed occurs on the second bending mode at $U_F = 66.5\ m/s$ which occurred at a normalized spanwise distance of $y/S = 0.54$. The gradient of damping near the flutter point differs from mass “5” and is significantly more gradual towards the flutter point which starts at approximately $62\ m/s$. However, the damping is consistently and nearly underdamped (near zero) throughout all corresponding velocities, therefore, despite the gradual gradient of damping towards the flutter point, flutter will occur suddenly and without significant warning since the wings will be vibrating consistently throughout the flight. Furthermore, from the damping and frequency plots, it is clear that a strong coupling is present between the second and third modes. This coupling is initially favourable as the frequency of the second mode reduces due to the approach of the flutter point, the frequency of the third mode increases respectively and therefore the damping of the third mode increases, as can be seen in figure 4.7. However, once the flutter condition occurs, the instability of the second mode causes the third mode to become highly unstable.

5. References

[1] C. Wright, Introduction to Aicraft Aeroelasticity and loads, Chichster: John Wiley & Sons ltd., 2007, pp. 3-199, 381-415.

[2] H. L. Runyan and J. L. Sewall, “Experimental investigation of the effects of concentrated weights on flutter characteristics of a straight cantelever wing,” National Advisory Committee for Aeronautics , Washington, 1948.

[3] G. Barakos, “ASSUMED SHAPES METHOD - Lecture 2,” University of Glasgow, Glasgow, 2018.

[4] W. H. Mason, Applied Computational Aerodynamics 1st edition, New York: Cambridge University press, 1998.

[5] E. Saltzman and J. Hicks, “In-Flight Lift-Drag Characteristics for a Forward-Swept Wing Aircraft,” NASA, California , 1994.

[6] R. M. Cummings, W. H. Mason, S. A. Morton and D. R. McDaniel, Applied Computational Aerodynamics - A Modern Engineering Approach, Cambridge: Cambridge University Press, 2015, pp. 1 - 338.

[7] G. Barakos, “Aerodynamics of Finite Wings - Lecture 3,” University of Glasgow, Glasgow, 2018.

[8] M. KNIGHT and R. W. NOYES, “SPAN -LOAD DISTRIBUTION AS A FACTOR IN STABILITY IN ROLL,” National Advisory Commitee For Aeronautics, Washington, 1931.

[9] G. Barakos, “INTRODUCTION TO FLUTTER - Lecture 4,” University of Glasgow, Glasgow, 2018.

[10] D. Hunsaker, “A Numerical Lifting-Line Method Using Horseshoe Vortex Sheets,” in *Conference: Rocky Mountain Space Grant Consortium Meeting Proceedings*, Logan, Utah, 2011.

6. Appendices

6.1 Assignment 1 code:

6.1.1 Using trapezoidal integration

```
%% -- Assignment 1: Mode shapes and natural frequency (Trapezodial integration) --

clear ; clc; close all
%% Wing parameters
% Unswept cantilever
% clc
% clear
% close all
CTRL.MSplot = 'yes';
VAR.mw      = 1.5818;           % 1.578501;         % wing mass                  (kg)
( Altered) tick
VAR.c       = 0.2032;           % Chord at 70% span         (m)tick
```

```
VAR.b       = VAR.c/2;          %                           (m)tick
VAR.S       = 1.2192;           % Semi span                 (m)tick
VAR.EA      = 0.437;            % Elastic axis              (% of chord)
VAR.IA      = 0.454;            % Inertial axis (CG)        (% of chord)
VAR.EI      = 404.76;           % 4.0376e+02;       % Bending Stiffness         (Nm^2)
VAR.GJ      = 199.076;          % 198.5813;         % Torsional stiffness       (Nm^2)
VAR.mu      = VAR.mw/VAR.S;     % Mass per length           (kg/m)
VAR.X       = (4.349e-3)/VAR.S;  % MoI per twist about EA    (kgm^2)
VAR.rho     = VAR.mu/(pi*(VAR.b^2)*32.6); %                 (kg/m^3)
VAR.xa      = (VAR.IA - VAR.EA)*VAR.c/VAR.b;  % Distance between Elastic and CG axis
normilized by semichord /VAR.b

% Store
VAR.ms = 0.636*VAR.mw;  % Mass of store
VAR.xs = 0.625*VAR.b;   % Distance between store CG and wing EA.
VAR.Is = 1.91*VAR.X*VAR.S;    % Store intertia
% Modes
VAR.N = 3;                      % Bending number of assumed shapes
VAR.M = 2;                      % Torsional number of assumed shapes
% Establishing variables
N = VAR.N;                      % Bending number of assumed shapes
M = VAR.M;                      % Torsional number of assumed shapes

VAR.y_L = 0:0.01:1;                             % Spanwise increments
Ss      = 0:VAR.S/(length(VAR.y_L)-1):VAR.S;    % Store position

for eigind = 1:length(Ss)
%% Bending
for i = 1:N
        WV.PSI      = (VAR.y_L).^(i+1);
        WV.PSIstore = (Ss(eigind)/VAR.S).^(i+1);
        WV.PSIDD    = ((i*(i+1))/VAR.S.^2)*(VAR.y_L.^(i-1));
        %Bending jth component for matrix multiplication
        for j = 1:N
            WV.PSIj      = (VAR.y_L).^(j+1);
            WV.PSIstorej = (Ss(eigind)/VAR.S).^(j+1);
            WV.PSIDDj    = ((j*(j+1))/VAR.S.^2)*(VAR.y_L.^(j-1));
            DEL(i,j)     =  VAR.S*trapz  (VAR.y_L,(WV.PSI.*WV.PSIj));   % Wing mass
matrix (DEL): Symbolic integration of matix (dy) Kinetic
            B  (i,j)     =  VAR.S*trapz  (VAR.y_L,(WV.PSIDD.*WV.PSIDDj));   % Wing
stiffness matrix (B): Symbolic integration of matrix (dy) potenial
            DELS(i,j)    = (WV.PSIstore.*WV.PSIstorej);
        end
        for k = 1:M
            WV.PHIj      = (VAR.y_L).^(k);
            WV.PHIDj     = (k/VAR.S)*(VAR.y_L.^(k-1));
            WV.PHIstorej = (Ss(eigind)/VAR.S).^(k);
            C(i,k)       = -VAR.S*trapz(VAR.y_L,(WV.PSI.*WV.PHIj));   %*VAR.b
            Cs(i,k)      = (WV.PSIstore.*WV.PHIstorej);         %/VAR.b
        end
end
clear WV i j k
%% Torsional modes
for i = 1:M
WV.PHI      = (VAR.y_L).^(i);
WV.PHIstore = (Ss(eigind)/VAR.S).^(i);
WV.PHID     = ((i)/VAR.S)*(VAR.y_L.^(i-1));
            for j = 1:M
            WV.PHIj      = (VAR.y_L).^(j);
            WV.PHIstorej = (Ss(eigind)/VAR.S).^(j);
            WV.PHIDj     = (j/VAR.S)*(VAR.y_L.^(j-1));
            T (i,j)      = VAR.S*trapz(VAR.y_L,(WV.PHID.*WV.PHIDj));   % Wing stiffness
matrix, T
            D (i,j)      = VAR.S*trapz (VAR.y_L,(WV.PHI.*WV.PHIj));   % Wing mass
matrix, D
            Ds(i,j)      = (WV.PHIstore.*WV.PHIstorej);
            end
end

%% Combining matrices for eigan calculations

% Stiffness matrix => | B   0 |
```

```
%                     | 0   T |

Kq  = [VAR.EI*B  , zeros(N,M)  ; zeros(N,M)',  VAR.GJ*T];

% Mass matrix => Atotal = Awing + Astore
%                       = |Del      xa*b*C|  + |DelS    xs(Cs) |
%                         |xa*b*C'     D   |    |xs(Cs')    DS   |

Mqw  = [VAR.mu*DEL,   VAR.mu*VAR.xa*VAR.b*C;   % C*VAR.b
        VAR.mu*VAR.xa*VAR.b*C',     VAR.X*D];  % + mass store
Mqs  = [VAR.ms*DELS,       VAR.ms*VAR.xs*Cs;  VAR.ms*VAR.xs*Cs'     , VAR.Is*Ds];
Mqt  = Mqw + Mqs;


%% Calculating frequencies etc using the Eigenvalue approach  (Torsion AND bending)
%% TOTAL BENDING AND TORSION: Frequency calcs

[EV.V, EVal.LAM] = eig(Kq,Mqt);
WV.omega = (diag(EVal.LAM)).^(1/2); % rad/s
WV.freq  = WV.omega/(2*pi);       % Hz
R.freqstore(:,eigind) = WV.freq;

if eigind == 1 % 1 = basically just the wing, store at root
    Vstore_Ss0      = EV.V;
    LAMstore_Ss0    = EVal.LAM;
end

%% Bending AND Torsion: Frequency calcs WING ONLY

[EV.W_V, EVal.WLAM] = eig(Kq,Mqw);
WV.W_omega = (diag(EVal.WLAM)).^(1/2); %rad/s
WV.W_freq  = WV.W_omega/(2*pi);       %Hz
R.W_freqstore(:,eigind ) = WV.W_freq;

%% BENDING: Frequency calcs

% Wing + store
WV.B_Mqt = DEL + DELS;

[EV.B_V, EVal.B_LAM] = eig(B,WV.B_Mqt);
WV.B_omega = (diag(EVal.B_LAM)).^(1/2);
WV.B_freq  = WV.B_omega/(2*pi);
R.B_freqstore(:,eigind ) = WV.B_freq;


[EV.BW_V, EVal.BW_LAM] = eig(B,DEL);
WV.BW_omega = (diag(EVal.BW_LAM)).^(1/2);
WV.BW_freq  = WV.BW_omega/(2*pi);
R.BW_freqstore(:,eigind) = WV.BW_freq;

%% TORSION: Frequency calcs
% Wing plus store
WV.T_Mqt = D + Ds;
[EV.T_V, EVal.T_LAM] = eig(T,WV.T_Mqt);
WV.T_omega = (diag(EVal.T_LAM)).^(1/2);
WV.T_freq  = WV.T_omega/(2*pi);
R.T_freqstore(:,eigind) = WV.T_freq;

% Only wing
[EV.TW_V, EVal.TW_LAM] = eig(T,D);
WV.TW_omega = (diag(EVal.TW_LAM)).^(1/2);
WV.TW_freq  = WV.TW_omega/(2*pi);
R.TW_freqstore(:,eigind) = WV.TW_freq;

end
clear i j eigind

%% Get experimental data
addpath('C:\Users\nicho\OneDrive\Documents\MATLAB\aeroelasticity 5\Experimental data')
% Using GRABIT
% Bending mode 1
load('B_M1.mat')
```

```
if B_M1(1,2) < 0
   B_M1(1,2) = 0;
end
load('B_M2.mat')
if B_M2(1,1) < 0
   B_M2(1,1) = 0;
end
% Bending mode 2
load('Bending_M2.mat')
if Bending_M2(1,1) < 0
    Bending_M2(1,1) = 0;
end
% torsion mode 1
load('T_M1.mat')
if T_M1(1,1) < 0
    T_M1(1,1) = 0;
end
% From data sheet  (comparing with unnormilized frequencies)
SP =   [0     29     11      47    41    20.5  29     26   ].*0.0254;
EBFM1 =[6.45  5.22   6.64   3.65   4.17  6.14  5.18   5.56 ];
EBFM2 =[39.2  35.10  30.41  34.35  38.5  35.6  34.77  33.69];
ETFM1 =[47.3  24.5   40     22.5   23.9  35.6  24.74  24.27];

%% Get symbolic
% Modes N = 3, M = 2
load('Sfreqstore.mat')
load('SW_freqstore.mat')
load('N_freqSYM.mat')
load('NB_freqSYM.mat')
load('NT_freqSYM.mat')

% Modes N = 2, M = 1
load('N2M1_B_SYM.mat')
load('N2M1_T_SYM.mat')
% Modes N = 5, M = 4;
load('N5M4_B_SYM.mat')
load('N5M4_T_SYM.mat')

y_L1 = 0:1/(length(N2M1_B)-1):1;
%% Plotting results
ind = 0:0.01:VAR.S;

% Full frequency matrice
figure
set(gcf,'color','w')
% Trapz
plot(VAR.y_L,R.freqstore(1,1:end),'color',[0.8 0 0],'LineWidth',1.5); hold on
plot(VAR.y_L,R.freqstore(2,1:end),'color',[0 0.7 0],'LineWidth',1.5); hold on
plot(VAR.y_L,R.freqstore(3,1:end),'color',[0 0 0.8],'LineWidth',1.5); hold on
% Sym
plot(ind/VAR.S,Sfreqstore(1,1:end-1),'--','color',[0.8 0 0]); hold on
plot(ind/VAR.S,Sfreqstore(2,1:end-1),'--','color',[0 0.8 0]); hold on
plot(ind/VAR.S,Sfreqstore(3,1:end-1)); hold on
% Exp
plot(SP/VAR.S,EBFM1,'s','MarkerFaceColor',[0.8 0 0],'MarkerEdgeColor',[1 0 0])
plot(SP/VAR.S,EBFM2,'d','MarkerFaceColor',[0 0 0.8],'MarkerEdgeColor',[0 0 1])
plot(SP/VAR.S,ETFM1,'o','MarkerFaceColor',[0 0.7 0],'MarkerEdgeColor',[0 1 0])
title('Total frequencies for modes 1 - 3');
xlabel('y/S','Interpreter', 'Latex');
ylabel('Frequency (Hz)','Interpreter', 'Latex')
legend('TI: Mode 1','TI: Mode 2','TI: Mode 3','EXP: Mode 1','EXP: Mode 2','EXP Mode 3')
set(gca,'FontName','Times New Roman','FontSize',12);
grid minor


%% Normilized frequencies
%% Bending

R.B_Norm = R.B_freqstore./R.BW_freqstore;
R.T_Norm = R.T_freqstore./R.TW_freqstore;
figure
set(gcf,'color','w')
```

```
% % Trapz
% plot(VAR.y_L,R.B_Norm(1,1:end),'color',[0.8 0 0],'LineWidth',1.5); hold on
% plot(VAR.y_L,R.B_Norm(2,1:end),'color',[0 0.7 0],'LineWidth',1.5); hold on

% Sym N3 M2
plot(ind/VAR.S,NB_freq(1,1:end-1),'--','color',[0.8 0 0],'LineWidth',1.5);hold on
plot(ind/VAR.S,NB_freq(2,1:end-1),'--','color',[0 0.7 0],'LineWidth',1.5);hold on
% Sym N2 M1
% plot(y_L1,N2M1_B(1,:),'-','color',[0 0 0.5],'Linewidth',2 ); hold on
% plot(y_L1,N2M1_B(2,:),'-','color',[0 0.5 0.5],'Linewidth',2 ); hold on
% Sym N5 M4
plot(y_L1,N5M4_B(1,:),'-','color',[0 0 0.5],'Linewidth',2 ); hold on
plot(y_L1,N5M4_B(2,:),'-','color',[0 0.5 0.5],'Linewidth',2 ); hold on

% Experi
plot(B_M1(:,2),B_M1(:,1),'s','MarkerFaceColor',[0.8 0 0],'MarkerEdgeColor',[1 0 0]);
hold on
plot(B_M2(:,1)/VAR.S,B_M2(:,2),'o','MarkerFaceColor',[0 0.8 0],'MarkerEdgeColor',[0 1
0]); hold on
title('Normilized bending frequencies for modes 1 - 2');
xlabel('$$y/S$$','Interpreter', 'Latex');
ylabel('$$\omega_{w}/\omega$$','Interpreter', 'Latex')
grid minor
% legend('TI: Mode 1','TI: Mode 2','SYM: Mode 1', 'SYM: Mode 2','EXP: Mode 1','EXP: Mode
2')
% legend('N3 M2 - Mode 1','N3 M2 - Mode 2', 'N2 M1 - Mode 1', 'N2 M1 - Mode 2', 'EXP:
Mode 1', 'EXP: Mode 2')
legend('N3 M2 - Mode 1','N3 M2 - Mode 2', 'N5 M4 - Mode 1', 'N5 M4 - Mode 2', 'EXP: Mode
1', 'EXP: Mode 2')
set(gca,'FontName','Times New Roman','FontSize',12);


%% Torsion

figure
set(gcf,'color','w')
% % Trapz
% plot(VAR.y_L,R.T_Norm(1,1:end),'color',[0.8 0 0],'LineWidth',1.5); hold on
% Sym N3M2
plot(ind/VAR.S,NT_freq(1,1:end-1),'--','color',[0.8 0 0],'LineWidth',1.5);hold on
% SYM N2M1
plot(y_L1,N2M1_T(1,:),'-','color',[0 0 0.5],'Linewidth',2 ); hold on
% Sym N5 M4
plot(y_L1,N5M4_T(1,:),'-','color',[0 0.5 0.5],'Linewidth',2 ); hold on


% Exp
plot(T_M1(:,1)/VAR.S,T_M1(:,2),'s','MarkerFaceColor',[0.8 0 0],'MarkerEdgeColor',[1 0
0]);
title('Normilized torsion frequencies for modes 1');
xlabel('$$y/S$$','Interpreter', 'Latex');  ylabel('$$\omega_{w}/\omega$$','Interpreter',
'Latex')
% legend('TI: Mode 1','SYM: Mode 1','EXP: Mode 1')
legend('N3 M2: Mode 1','N2 M1: Mode 1','N5 M4: Mode 1','EXP: Mode 1')
set(gca,'FontName','Times New Roman','FontSize',12);
grid minor
```

6.1.2 Using symbolic integration

```
%% -- Assignment 1: Mode shapes and natural frequency (Symbolic integration) --

% Done in symbolic to have more so-called exact answers instead of
% approximate trapezodial integration, however, this most likely increases
% CPU and memory costs.
```

```
% Note, report plots done on trapz file

switch init
    case 0
        clear; clc; close all
%% Wing parameters
% Unswept cantilever
% clc
% clear
% close all
CTRL.MSplot = 'yes';
VAR.mw      = 1.578501;         % wing mass                 (kg) ( Altered)
VAR.c       = 0.2032;           % Chord at 70% span         (m)
VAR.b       = VAR.c/2;          %                           (m)
VAR.S       = 1.2192;           % Semi span                 (m)
VAR.EA      = 0.437;            % Elastic axis              (% of chord)
VAR.IA      = 0.454;            % Inertial axis (CG)        (% of chord)
VAR.EI      = 4.0376e+02;       % Bending Stiffness         (Nm^2)
VAR.GJ      = 198.5813;         % Torsional stiffness       (Nm^2)
VAR.mu      = VAR.mw/VAR.S;     % Mass per length           (kg/m)
VAR.X       = (4.349e-3)/VAR.S;         % MoI per twist about EA    (kgm^2)
VAR.rho     = VAR.mu/(pi*(VAR.b^2)*32.6); %                 (kg/m^3)
VAR.xa      = (VAR.IA - VAR.EA)*VAR.c/VAR.b;  % Distance between Elastic and CG axis
normilized by semichord /VAR.b

% Store
VAR.ms = 0.636*VAR.mw;  % Mass of store
VAR.xs = 0.625*VAR.b;   % Distance between store CG and wing EA.
VAR.Is = 1.91*VAR.X*VAR.S;    % Store intertia
% Modes
VAR.N = 3;                      % Bending number of assumed shapes
VAR.M = 2;                      % Torsional number of assumed shapes
% Establishing variables
N = VAR.N;                      % Bending number of assumed shapes
M = VAR.M;                      % Torsional number of assumed shapes

% Establishing symbolic variables
syms y S muu EI X GJ xa b ms xs Is
%% Wing matrices
% Bending of wing
% N = bending modes, M = torsional modes

% Bending modes of wing
for i = 1:N
        WV.PSI   (i,1)  = (y/S)^(i+1);                              % Establishing
polynomial assumed shape
        WV.PSIDD(i,1)  = diff((diff(WV.PSI(i,1),y)),y);             % d^2z/dq^2Double
differenciation of assumed shape
        WV.PSIT (1,i)  = (y/S)^(i+1);
        WV.PSITDD(1,i) = diff((diff(WV.PSIT(1,i),y)),y);
end
% Torsional assumed shapes
for k = 1:M
        WV.PHI   (k,1) = (y/S)^(k);
        WV.PHID (k,1) = ((diff(WV.PHI(k,1),y)));
        WV.PHIT (1,k) = (y/S)^(k);
        WV.PHITD(1,k) = ((diff(WV.PHIT(1,k),y)));
end

% Computing matrix components for mass and stiffness
        DEL = muu*int( (WV.PSI*WV.PSIT)     ,y,0,S);   % Wing mass matrix (DEL):
Symbolic integration of matix (dy) Kinetic
        B   = EI*int ( (WV.PSIDD*WV.PSITDD) ,y,0,S);   % Wing stiffness matrix (B):
Symbolic integration of matrix (dy) potenial
        D   = X*int  ( (WV.PHI*WV.PHIT)     ,y,0,S);   % Wing mass matrix, D
        T   = GJ*int ( (WV.PHID*WV.PHITD)   ,y,0,S);   % Wing stiffness matrix, T
        C   = -muu*xa*b*int((WV.PSI*WV.PHIT),y,0,S);
        CT  = -muu*xa*b*int((WV.PHI*WV.PSIT),y,0,S);

% Store matrix components for mass (Stifness uneffected by the store)
% Following similar procedure to before, however, do not need to calculate
```

```
% potential energy as store does NOT effect stiffness matrix.
        DELS = ms*(WV.PSI*WV.PSIT);
        Ds   = Is*(WV.PHI*WV.PHIT);
        Cs   = ms*xs*(WV.PSI*WV.PHIT);
        CsT  = ms*xs*(WV.PHI*WV.PSIT);

%% Combining matrices for eigan calculations

% Stiffness matrix => | B   0 |
%                     | 0   T |

Kq  = [B  , zeros(N,M)  ; zeros(N,M)',  T];

% Mass matrix => Atotal = Awing + Astore
%                       = |Del      xa*b*C|  + |DelS    xs(Cs) |
%                         |xa*b*C'     D   |    |xs(Cs')    DS   |

Mqw  = [DEL, C;  CT, D];  % + mass store
Mqs  = [DELS Cs; CsT, Ds];
Mqt  = Mqw + Mqs;

%% Displaying matrices for checking
fprintf('\n\nBending wing kinetic, DEL =\n')
DEL
fprintf('\n\nBending wing potential, B=\n')
B
fprintf('\n\nTorsion kinetic, D =\n')
D
fprintf('\n\nTorsion potential, T =\n')
T
fprintf('\n\nMatrix C =\n')
C
fprintf('\n\nWing Mass Matrix, Mq =\n')
Mqw
fprintf('\n\nWing Stiffness Matrix, Kq =\n')
Kq
fprintf('\n\nStore Mass Matrix, Mqs =\n')
Mqs
fprintf('\n\nTotal Mass Matrix, Mqt =\n')
Mqt

init = 1;
    case 1
        clear freqstore W_freqstore B_freqstore BW_freqstore T_freqstore TW_freqstore
        close all
end

%% Calculating frequencies etc using the Eigenvalue approach  (Torsion AND bending)

VAR.y= 0;
ind = 0:0.01:VAR.S;

for eigind = 1:1:length(ind)+1
%% TOTAL BENDING AND TORSION: Frequency calcs

Kqtval = double(vpa(subs(Kq,[EI S GJ],[VAR.EI VAR.S VAR.GJ]),20));
Mqtval = double(vpa(subs(Mqt,[S muu ms y xs b xa Is X],[VAR.S VAR.mu VAR.ms VAR.y VAR.xs
VAR.b VAR.xa VAR.Is VAR.X]),20));

[V, LAM,W] = eig(Kqtval,Mqtval);
omega = (diag(LAM)).^(1/2); %rad/s
freq  = omega/(2*pi);       %Hz
freqstore(:,eigind) = freq;

% Getting eigenvector for selected store position
% Store position VAR.S/2 (approx 0.6)
if eigind == 1 % 1 = basically just the wing, store at root
    Vstore = V;
    LAMstore = LAM;
    Wstore = W;
end
%% Bending AND Torsion: Frequency calcs WING ONLY
```

```
Kqwval = double(vpa(subs(Kq,[EI S GJ],[VAR.EI VAR.S VAR.GJ]),20));
Mqwval = double(vpa(subs(Mqw,[EI GJ S muu ms y xs b xa Is X],[VAR.EI VAR.GJ VAR.S VAR.mu
VAR.ms VAR.y VAR.xs VAR.b VAR.xa VAR.Is VAR.X]),20));

[WV, WLAM] = eig(Kqwval,Mqwval);
W_omega = (diag(WLAM)).^(1/2); %rad/s
W_freq  = W_omega/(2*pi);       %Hz
W_freqstore(:,eigind) = W_freq;

%% BENDING: Frequency calcs

% Wing + store
B_Mqt = DEL + DELS;
B_Kqt = B;
B_Mqtval= double(vpa(subs(B_Mqt,[EI GJ S muu ms y xs b xa Is X],[VAR.EI VAR.GJ VAR.S
VAR.mu VAR.ms VAR.y VAR.xs VAR.b VAR.xa VAR.Is VAR.X]),20));
B_Kqtval= double(vpa(subs(B_Kqt,[EI S GJ],[VAR.EI VAR.S VAR.GJ]),20));

[B_V, B_LAM] = eig(B_Kqtval,B_Mqtval);
B_omega = (diag(B_LAM)).^(1/2);
B_freq  = B_omega/(2*pi);
B_freqstore(:,eigind) = B_freq;

% Only wing
BW_Mqw = DEL;
BW_Kq  = B;
BW_Mqwval= double(vpa(subs(BW_Mqw,[EI GJ S muu ms y xs b xa Is X],[VAR.EI VAR.GJ VAR.S
VAR.mu VAR.ms VAR.y VAR.xs VAR.b VAR.xa VAR.Is VAR.X]),20));
BW_Kqwval= double(vpa(subs(BW_Kq,[EI S GJ],[VAR.EI VAR.S VAR.GJ]),20));

[BW_V, BW_LAM] = eig(BW_Kqwval,BW_Mqwval);
BW_omega = (diag(BW_LAM)).^(1/2);
BW_freq  = BW_omega/(2*pi);
BW_freqstore(:,eigind) = BW_freq;

%% TORSION: Frequency calcs

% wing + store
T_Mqt = D + Ds;
T_Kqt = T;
T_Mqtval= double(vpa(subs(T_Mqt,[EI GJ S muu ms y xs b xa Is X],[VAR.EI VAR.GJ VAR.S
VAR.mu VAR.ms VAR.y VAR.xs VAR.b VAR.xa VAR.Is VAR.X]),20));
T_Kqtval= double(vpa(subs(T_Kqt,[EI S GJ],[VAR.EI VAR.S VAR.GJ]),20));

[T_V, T_LAM] = eig(T_Kqtval,T_Mqtval);
T_omega = (diag(T_LAM)).^(1/2);
T_freq  = T_omega/(2*pi);
T_freqstore(:,eigind) = T_freq;

% only wing
TW_Mqw = D;
TW_Kqw = T;
TW_Mqwval= double(vpa(subs(TW_Mqw,[EI GJ S muu ms y xs b xa Is X],[VAR.EI VAR.GJ VAR.S
VAR.mu VAR.ms VAR.y VAR.xs VAR.b VAR.xa VAR.Is VAR.X]),20));
TW_Kqwval= double(vpa(subs(TW_Kqw,[EI S GJ],[VAR.EI VAR.S VAR.GJ]),20));

[TW_V, TW_LAM] = eig(TW_Kqwval,TW_Mqwval);
TW_omega = (diag(TW_LAM)).^(1/2);
TW_freq  = TW_omega/(2*pi);
TW_freqstore(:,eigind) = TW_freq;

%% Incrementing spanwise position
VAR.y = VAR.y + 0.01;
end

switch CTRL.MSplot
    case 'yes'
%% Experimental data
% Using GRABIT
% Bending mode 1
load('B_M1.mat')
if B_M1(1,2) < 0
```

```
   B_M1(1,2) = 0;
end
B_M1 = [(B_M1(:,1)),(B_M1(:,2)*1.2192)];
% Bending mode 2
load('B_M2.mat')
if B_M2(1,1) < 0
    B_M2(1,1) = 0;
end
% torsion mode 1
load('T_M1.mat')
if T_M1(1,1) < 0
    T_M1(1,1) = 0;
end
% From data sheet  (comparing with unnormilized frequencies)
SP =   [0     29     11      47    41    20.5  29     26   ].*0.0254;
EBFM1 =[6.45  5.22   6.64   3.65   4.17  6.14  5.18   5.56 ];
EBFM2 =[39.2  35.10  30.41  34.35  38.5  35.6  34.77  33.69];
ETFM1 =[47.3  24.5   40     22.5   23.9  35.6  24.74  24.27];

%% Deflections
% Vstore for selected store position.
 y_L=0:0.01:VAR.S;
 for i=1:2 % first two modes
     z_EA(i,:)=V(1,i)'*(y_L).^2;
 theta_EA(i,:)=V(2,i)'*(y_L);
     z_EA(i,:)=V(1,i)'*(y_L).^2;
 theta_EA(i,:)=V(2,i)'*(y_L);
 end

%% UNNORMILIZED Plots
% -- Combined frequencies --
figure(1)
plot(ind/VAR.S,freqstore(1,1:length(freqstore)-1)); hold on
plot(ind/VAR.S,freqstore(2,1:length(freqstore)-1)); hold on
plot(ind/VAR.S,freqstore(3,1:length(freqstore)-1)); hold on
plot(SP/VAR.S,EBFM1,'o','MarkerSize',7); hold on
plot(SP/VAR.S,EBFM2,'d','MarkerSize',7); hold on
plot(SP/VAR.S,ETFM1,'o','MarkerSize',7);
grid minor
xlabel('Span Position (m)'); ylabel('Frequency (Hz)'); title('Total frequencies')
legend('Mode 1','Mode 2','Mode 3','CD-Mode 1', 'CD-Mode 2', 'CD-Mode 3')
hold off

%% NORMILIZED PLOTS
% -- Combined frequencies --
% ind = 0:0.01:VAR.S

% Normilizing frequencies
% Total (Bending + torsion)
N_freq = freqstore./W_freqstore;
% Bending only
NB_freq = B_freqstore./BW_freqstore;
% Torsion only
NT_freq = T_freqstore./TW_freqstore;

% -- Bending frequencies --
figure(5)
plot(ind/VAR.S,NB_freq(1,1:end-1)); hold on
plot(ind/VAR.S,NB_freq(2,1:end-1)); hold on
plot(B_M1(:,2)/VAR.S,B_M1(:,1),'o','MarkerSize',7)
plot(B_M2(:,1)/VAR.S,B_M2(:,2),'x','MarkerSize',7)
grid minor
xlabel('Span Position (m)'); ylabel('omegaw,b/omega,b')
title('Normilized Bending frequencies')
legend('BM-1','BM-2','CD-BM1','CD-BM1')
hold off

% -- Torsion frequencies --
figure(6)
plot(ind/VAR.S,NT_freq(1,1:end-1)); hold on
plot(T_M1(:,1)/VAR.S,T_M1(:,2),'o')
grid minor
```

```
xlabel('Span Position (y/S)'); ylabel('omega_{w,t}/omega_{t}')
title('Normilized Torsion frequencies')
legend('TM-1','CD-TM1')
    case 'no'
end
```

6.2 Assignment 2 code

6.2.1 Vortex Lattice method (standalone)

```
%% Vortex lattice method A2
% Notes
% Structured matrices:
% - plt  = plots
% - CTRL = Controls
% - X is chordwise direction, Y is spanwise, Z is up and down

clc; clear; close all
%% Numerical parameters and controls
Np= 100; % number of panels
CTRL.spacing = 'uniform';   % 'cosine' or 'uniform'
swept  = 'forward';             % 'forward' 'aft' 'unswept'


% Aerodynamics - Warren wing parameters
Cla    = 2.743;            % Lift slop (/rad)
Cma    = 3.10;             % Coef moment (/rad)
alpha  = 6*pi/180;         % AoA in radians
Vinf   = 150;
rho    = 1.225;

% Geometric parameters
switch swept
    case 'forward'
         cLam = -45*pi/180;
    case 'unswept'
         cLam = 0*pi/180;
    case 'aft'
         cLam = 45*pi/180;
end
Cr   = 1.5;             % Root chord
Ct   = 0.5;             % Tip chord                0.9          0.5
lam  = Ct/Cr;           % Taper ratio
Cr_t = 1.91421;         % Root tip to tip tip
S    = (2)^(1/2);       % Semi span length         6            (2)^(1/2)
HSa  = (2)^(1/2);       % Half Wing area       14.4         (2)^(1/2)
OF   = 0;               % Offset fuselage
%% Spatial discretion
% Distribution of panels
% dy = spacing
switch CTRL.spacing
    case 'uniform'
         DY = 2*S/(Np);    % -1 as grid points are on root and tip
         y  = -S:DY:S;
    case 'cosine'
        CSpace = linspace(0,pi,(Np/2)+1);
        y  = OF + (S - OF)*0.5.*(1 + cos(CSpace));
        y  = [-y(1:end-1),fliplr(y)];  % for root chord starting at index 1
end

%% Geometric calculations
% Chord varies with taper
% Distribution:
c_yR= (2*HSa)/((1+lam)*S).*(1-(((1-lam)/S).*y(((Np/2)+1):end)));
```

```
c_y = [fliplr(c_yR),c_yR(2:end)];

% Establish 1/4, 3/4, 1/2 chord and LE, TE lines for the wing
% Calculating x position for each
% - Origin on half chord at root chord.
% - Using half chord as reference line

x_hc  = sign(y).*y.*tan(cLam);
x_tqc = x_hc + c_y./4;
x_qc  = x_hc - c_y./4;
x_te  = x_hc + c_y./2;
x_le  = x_hc - c_y./2;

% Z_coords = 0 atm
z_hc = repmat(0,1,length(x_hc));
z_tqc =repmat(0,1,length(x_tqc));
z_qc = repmat(0,1,length(x_qc));
z_te = repmat(0,1,length(x_te));
z_le = repmat(0,1,length(x_le));

%% Colocation points and reference points
% A,B on quater chord.
% C on 3/4 chord (AC).

y_A = y(1:1:end-1); % Start point and every point after.
y_B = y(2:1:end); % Start at second point and every  point after.
y_C = (y_A + y_B)/2; % inbetween A and B

x_A = x_qc(1:1:end-1);    % |
x_B = x_qc(2:1:end);    % |> On Quater chord.
x_C = (x_tqc(1:1:end-1)+ x_tqc(2:1:end))/2; % On 3 quater chord
x_C2=interp1(y,x_tqc,y_C);

z_A = repmat(0,1,length(x_A));
z_B = repmat(0,1,length(x_B));
z_C = repmat(0,1,length(x_C));

% Matrices for colocation points
A = [x_A; y_A; z_A];
B = [x_B; y_B; z_B];
C = [x_C; y_C; z_C];

% Note, 100 points (n) = 50 panels
%% Biot-Savert: influence coefficient calculations
% r0,r1,r2
% Need to loop for all points. A(1,1) will be paired with B(1,1), C(1,1).

% Need to loops to calculate each HSV with respect to other HSV. Should get
% n x n matrix of AIC
for j = 1:(Np)  % Defining colocation point for calculations
    for k = 1:(Np)  % Changing colocation points for colocation pt,i

    r_0 = [(x_B(1,k) - x_A(1,k));(y_B(1,k) - y_A(1,k));(z_B(1,k) - z_A(1,k))];
    r_1 = [(x_C(1,j) - x_A(1,k));(y_C(1,j) - y_A(1,k));(z_C(1,j) - z_A(1,k))];
    r_2 = [(x_C(1,j) - x_B(1,k));(y_C(1,j) - y_B(1,k));(z_C(1,j) - z_B(1,k))];



VAB(j,k,:)=((1/(4*pi))*((cross(r_1,r_2))/((norm(cross(r_1,r_2))).^2))*(dot(r_0,((r_1/(no
rm(r_1)))-(r_2/(norm(r_2))))))');

%   C = j indice as want to analyse C for every A and B
%   Every row is a colocation point values influenced by the other points
    VAcomp1 = (1/(4*pi));
    VAcomp2 = (([ 0, (z_C(1,j)-z_A(1,k)) , (y_A(1,k)-y_C(1,j))]));
    VAcomp3 =  1/(((z_C(1,j)-z_A(1,k))^2) + ((y_A(1,k)-y_C(1,j))^2));
    VAcomp4 = (1 + ((x_C(1,j)-x_A(1,k))/(( ((x_C(1,j)-x_A(1,k))^2) + ((y_C(1,j)-
y_A(1,k))^2) + ((z_C(1,j)-z_A(1,k))^2) )^(1/2))));
    VAI(j,k,:) = VAcomp2.*VAcomp1.*VAcomp3.*VAcomp4;

    VBcomp1 = (-1/(4*pi));
    VBcomp2 = (([ 0, (z_C(1,j)-z_B(1,k)) , (y_B(1,k)-y_C(1,j))]));
```

```
    VBcomp3 = 1/(((z_C(1,j)-z_B(1,k))^2) + ((y_B(1,k)-y_C(1,j))^2));
    VBcomp4 = (1 + ((x_C(1,j)-x_B(1,k))/(( ((x_C(1,j)-x_B(1,k))^2) + ((y_C(1,j)-
y_B(1,k))^2) + ((z_C(1,j)-z_B(1,k))^2) )^(1/2))));
    VBI(j,k,:) = VBcomp2.*VBcomp1.*VBcomp3.*VBcomp4;

    end
end
clear VAcomp1 VAcomp2 VAcomp3 VAcomp4 VBcomp1 VBcomp2 VBcomp3 VBcomp4

 IC =VAB +VAI + VBI;
% Sorting Influence coefficients into x,y,z components
ICx = squeeze(IC(:,:,1));
ICy = squeeze(IC(:,:,2));
ICz = squeeze(IC(:,:,3));

% Calculate normals for each panel (V.n = 0;)
% So that boundary conditions of      ^^   can be applied and formulated
% equation circ*IC + V(cos(alpha)+sin(alpha)) = 0;
for i = 1:1:(Np)
    n = cross(A(:,i)-C(:,i),A(:,i)-B(:,i)); % unit normal from plane formed by A,B,C
    nval(:,i)=n/((dot(n,n))^(1/2));
end
% Applying normals
for i=1:(Np)
    for j=1:(Np)
        % Multipling normal vectors with respective AIC direction (eg, x
        % normal component with AICx)
        AIC(i,j)=(((nval(1,j)*ICx(i,j)) + (nval(2,j)*ICy(i,j)) + (nval(3,j)*ICz(i,j))));
    end
end
% Calculating incident flow velocity and apply V.n=0 boundary condition
% cos(a) = x direction, sin(a) = y direction
Velcond = -((Vinf*((cos(alpha)*nval(1,:)) + (sin(alpha)*nval(3,:)))));

% Circulation at colocation points
circulation = AIC\Velcond';

%% Method 1: From Dr Barakos

switch CTRL.spacing
    case 'uniform'
    qinf=0.5*rho*Vinf*Vinf;          % dynamic head
    Li=rho*Vinf.*circulation.*DY;    % calculate local lift
    Ltot1=sum(Li);                   % total lift
    Cltot1=Ltot1/(qinf*2*HSa);       % wing lift coefficient
    Cla1=Cltot1/alpha;               % Compute Lift-slope

    % Calculate local lift coefficients
    for i=1:Np
        localc(i)=(c_y(i)+c_y(i+1))/2.;
        Cl(i)=Li(i)/(qinf*localc(i)*DY);
    end
    case 'cosine'
    qinf=0.5*rho*Vinf*Vinf;
    for i=1:Np
        DY = y(i+1)-y(i);
        Li(i)=rho*Vinf*circulation(i)*DY;
        localc(i)=(c_y(i)+c_y(i+1))/2.;
        Cl(i)=Li(i)/(qinf*localc(i)*DY);
    end
    Ltot1=sum(Li); % total lift
    Cltot1=Ltot1/(qinf*2*HSa); % wing lift coefficient
    Cla1=Cltot1/alpha; % Compute Lift-slope
end
% plot out CL/CL_total to show spanwise lift coefft distribution
figure(2);
title('Span-wise load for the forward-swept Warren-12
wing','FontSize',14,'FontWeight','Bold'); hold on
plot ((y_C((Np/2)+1:Np)/S),Cl((Np/2)+1:Np)/Cltot1,'LineWidth',2,'color',[0.8 0
0]); %/Cltot1
hold on
switch swept
```

```
    case 'forward'
        FSD = load('Forwardswept.mat');
        plot(FSD.Forwardswept(:,1),FSD.Forwardswept(:,2),'sr')
    case 'unswept'
        USD = load('Unswept.mat');
        plot(USD.Unswept(:,1),USD.Unswept(:,2),'sr')
    case 'aft'
        BSD = load('Aftswept.mat');
        plot(BSD.Aftswept(:,1),BSD.Aftswept(:,2),'sr')
end
xlabel ('Span-wise position (-)','FontSize',12);
ylabel ('Cl/Cl_{TOTAL}','FontSize', 12);
grid minor
set(gcf,'color','w');
legend('Results','VLMpc results')
set(gca,'FontName','Times New Roman','FontSize',12);



%% Plots

% Combining lines for plotting
plt.geolinesX  =[x_hc;x_qc;x_tqc;x_le;x_te];
plt.geolinesY = [y;y;y;y;y];
plt.geolinesZ = [z_hc;z_qc;z_tqc;z_le;z_te];

% Creating root and tip lines for plotting
plt.tiplineXL  = x_le(1,1):0.01:x_te(1,1);
plt.tiplineYL  = repmat(-S,1,length(plt.tiplineXL));
plt.tiplineZL  = repmat(0,1,length(plt.tiplineXL));
plt.rootlineX = x_le(1,(Np/2)+1):0.01:x_te(1,(Np/2)+1);
plt.rootlineY = zeros (1,length(plt.rootlineX));
plt.rootlineZ = repmat(0,1,length(plt.rootlineX));
plt.tiplineXR  = x_le(1,end):0.01:x_te(1,end);
plt.tiplineYR  = repmat(S,1,length(plt.tiplineXR));
plt.tiplineZR  = repmat(0,1,length(plt.tiplineXR));

figure
set(gcf,'color','w');
plot3(y,plt.geolinesX,plt.geolinesZ,'LineWidth',1)
hold on
plot3(plt.tiplineYL,plt.tiplineXL,plt.tiplineZL,'LineWidth',1)
hold on
plot3(plt.rootlineY,plt.rootlineX,plt.rootlineZ,'LineWidth',1)
hold on
plot3(plt.tiplineYR,plt.tiplineXR,plt.tiplineZR,'LineWidth',1)
hold on
% Quater chord A,B poitns
plot3(y_A,x_A,z_A,'db','MarkerSize',5)
hold on
plot3(y_B,x_B,z_B,'+g','MarkerSize',5)
hold on
plot3(y_C,x_C,z_C,'or')
legend('Half chord','Quater chord','3/4 chord', 'LE','TE')
title('Warren Wing: 3D plot')
set(gca,'FontName','Times New Roman');
xlabel('Y')
ylabel('X')
zlabel('Z')
grid minor
grid on

%2D plot of wing
figure
set(gcf,'color','w');
plot(y,plt.geolinesX,'LineWidth',1)
hold on
% Quater chord A,B poitns
plot(y_A,x_A,'db'); hold on
plot(y_B,x_B,'+k'); hold on
plot(y_C,x_C,'or'); hold on
plot(plt.tiplineYL,plt.tiplineXL,'k','LineWidth',1)
```

```
hold on
plot(plt.tiplineYR,plt.tiplineXR,'k','LineWidth',1)
hold on
plot(plt.rootlineY,plt.rootlineX,'k','LineWidth',1)
hold on
legend('Half chord','Quater chord','3/4 chord', 'LE','TE',...
        'A points','B points','Collocation pts')
title('Warren Wing: 2D plot')
xlabel('Span-wise, y (m)')
ylabel('Chord-wise, x (m)')
set(gca,'FontName','Times New Roman');
grid minor
grid on



%% Method 2: equation using gamma
for k = 1:1:(Np)
    gammaseg(1,k) = sum(circulation(k,:));
end
Li2  = rho*Vinf.*gammaseg*DY;
Ltot2=sum(Li2);                    % total lift
Cltot2=Ltot2/(qinf*2*HSa);         % wing lift coefficient
Cla2=Cltot2/alpha;                 % Compute Lift-slope

% calculate local lift coefficients
% c is the chord at the c points
for i=1:1:Np
    localc(i)= (c_y(1,i) + c_y(1,i+1))/2;
    Cl2(i)=(Li2(i)/(qinf*localc(i)*DY)); % *DY % DY is span of local segment, so
localc*DY = local area
end
figure,clf;
title('M2- Span-wise load for the Warren-12 wing','FontSize',14,'FontWeight','Bold')
hold on
grid on
plot (y_C((Np/2)+1:Np),Cl2((Np/2)+1:Np)/Cltot2,'LineWidth',2,'color',[0.8 0 0]);
xlabel ('Span-wise position (-)','FontSize',16);
ylabel ('Cl/Cl_{TOTAL}','FontSize', 16);
grid minor
set(gcf,'color','w');
set(gca,'FontName','Times New Roman');

%% Method 3

Liftcomp = eye(Np).*(DY*rho*Vinf);
Lift = Liftcomp.*circulation;
Li3 = diag(Lift);
for i = 1:1:(Np)
localc(i)= (c_y(1,i) + c_y(1,i+1))/2;
Cl3(i)=(Li3(i)/(qinf*localc(i)*DY)); % *DY % DY is span of local segment, so localc*DY =
local area
end
Ltot3=sum(Li3);                        % total lift
Cltot3=Ltot3/(qinf*2*HSa);
Cla3 = Cltot3/(alpha);

figure,clf;
title('M3 - Span-wise load for the Warren-12 wing','FontSize',14,'FontWeight','Bold')
hold on
grid on
plot (y_C((Np/2)+1:Np),Cl3((Np/2)+1:Np)/Cltot3,'LineWidth',2,'color',[0.8 0 0]);
xlabel ('Span-wise position (-)','FontSize',16);
ylabel ('Cl/Cl_{TOTAL}','FontSize', 16);
grid minor
set(gcf,'color','w');
set(gca,'FontName','Times New Roman');
```

6.2.2 Main code for elastic trimmer

```
%% Elastic trimmer model (A2)

% Function variables
% Using structured matrices for easily transfered variables between
% functions.

clc; clear; close all

%% ------------------------ASSIGNMENT 2 ---------------------------------------------
---

% Numerical parameters
VAR.Np              = 100;          % Number of panels
CTRL.spacing        = 'cosine';     % uniform or cosine
CTRL.trimVLM        = 'yes';        % Use VLM? 'yes' 'no'
CTRL.showmatrices   = 'no';         % Display structural matrices
CTRL.relaxation     = 0.1;          % Relaxation factor
CTRL.swept          = 'unswept';    % unswept, aft, forward

%% Elast strip theory vairiables
% Aero-parameters
VAR.mg    = 10*1000*9.81;      % Aircraft weight 10 ton aircraft
VAR.rho   = 0.5238*1.225;      % kg/m^3 at 20,000 ft
VAR.Vinf  = 150;               % Freestream velocity
VAR.qinf  = 0.5*VAR.rho*VAR.Vinf*VAR.Vinf;   % Dynamic head
% Elastic properties
VAR.EI    = 2e6;                     % Nm^2
VAR.GJ    = 5e5;                     % Nm^2/rad
% Number of modes
VAR.N     = 3;                      % Bending number of assumed shapes
VAR.M     = 2;                      % Torsional number of assumed shapes

%% Geometry
VAR.Cr    = 1.5;                         % Chord
VAR.lam   = 0.4;                         % Taper ratio
VAR.Ct    = VAR.Cr*VAR.lam;              % Tip chord
VAR.e     = 0.25;                        % Elastic axis location relative to quarter
chord
VAR.S     = 6;                           % Semi span
VAR.OF    = 0;                           % Offset fuselage

switch CTRL.swept
    case 'forward'
         VAR.cLam = -45*pi/180;
    case 'unswept'
         VAR.cLam = 0*pi/180;
    case 'aft'
         VAR.cLam = 45*pi/180;
end

%% Spatial discretion
% Distribution of panels
% DY = spacing
switch CTRL.spacing
    case 'uniform'
         VAR.DY = 2*VAR.S/(VAR.Np);     % -1 as grid points are on root and tip
         VAR.y  = 0:VAR.DY:VAR.S;
    case 'cosine'
        CSpace = linspace(0,pi,((VAR.Np/2)+1));
        VAR.y  = VAR.OF + (VAR.S - VAR.OF)*0.5.*(1 + cos(CSpace));
        VAR.y  = fliplr(VAR.y);         % for root chord starting at index 1
        for i = 1:length(CSpace)-1
        VAR.DY(i) = VAR.y(i+1) - VAR.y(i);
        end
end
```

```
%% Geometric calculations
% Chord varies with taper
% Distribution:
VAR.y_L  = VAR.y/VAR.S;
c_yR     = VAR.Cr*(1-VAR.lam*VAR.y_L);
VAR.c_y  = [fliplr(c_yR(2:end)),c_yR];
% VAR.c_y  = [fliplr(c_yR(2:end)),c_yR(2:end)];

switch CTRL.spacing
    case 'uniform'
    VAR.Sa   = trapz(VAR.c_y)*VAR.DY;             % Wing area TOTAL
    case 'cosine'
    VAR.DY   = [VAR.DY,VAR.DY];
    for i = 1:length(VAR.c_y)-1
        VAR.Sa(i) = trapz([VAR.c_y(i+1), VAR.c_y(i)])*VAR.DY(i);
    end
    VAR.Sa = sum(VAR.Sa);
end
VAR.AR   = ((VAR.S*2)^2)/VAR.Sa;              % Aspect ratio
VAR.HSa  = VAR.Sa/2;                          % Half of wing area

[Kqwval] = Kmatfunction(VAR,CTRL);
VAR.Kq = Kqwval;


% NOTE: VLMfunction works with a FULL WING SPAN whereas trimmer works with
% SEMI WING SPAN
switch CTRL.trimVLM
    case 'no'
        VAR.cLam = 0*pi/180;                   % Sweep in rads
        [CL] = trimmerNOVLM(VAR,CTRL);
    case 'yes'
        [CL] = trimmerWITHVLM(VAR,CTRL);
end
```

6.2.3 Stiffness matrix function for trimmer

```
function [Kqwval] = Kmatfunction(VAR,CTRL)

% Establishing variables
N = VAR.N;                      % Bending number of assumed shapes
M = VAR.M;                      % Torsional number of assumed shapes

% Establishing symbolic variables
syms y S muu EI X GJ xa b ms xs Is
%% Wing matrices
% Bending of wing
% N = bending modes, M = torsional modes

% Bending modes of wing
for i = 1:N
        WV.PSI  (i,1)  = (y/S)^(i+1);                          % Establishing
polynomial assumed shape
        WV.PSIDD(i,1)  = diff((diff(WV.PSI(i,1),y)),y);        % d^2z/dq^2Double
differenciation of assumed shape
        WV.PSIT (1,i)  = (y/S)^(i+1);
        WV.PSITDD(1,i) = diff((diff(WV.PSIT(1,i),y)),y);
end
% Torsional assumed shapes
for k = 1:M
        WV.PHI  (k,1) = (y/S)^(k);
        WV.PHID (k,1) = ((diff(WV.PHI(k,1),y)));
        WV.PHIT (1,k) = (y/S )^(k);
        WV.PHITD(1,k) = ((diff(WV.PHIT(1,k),y)));
```

```
end

% Computing matrix components for mass and stiffness
        B   = EI*int ( (WV.PSIDD*WV.PSITDD) ,y,0,S);   % Wing stiffness matrix (B):
Symbolic integration of matrix (dy) potenial
        T   = GJ*int ( (WV.PHID*WV.PHITD)   ,y,0,S);   % Wing stiffness matrix, T

%% Combining matrices for eigan calculations

% Stiffness matrix => | B   0 |
%                     | 0   T |

Kq  = [B  , zeros(N,M)  ; zeros(N,M)',  T];


%% Displaying matrices for checking

switch CTRL.showmatrices
    case 'yes'

fprintf('\n\nBending wing potential, B=\n')
B
fprintf('\n\nTorsion potential, T =\n')
T
fprintf('\n\nWing Stiffness Matrix, Kq =\n')
Kq
    case 'no'
end

Kqwval = double(vpa(subs(Kq, [EI S GJ],[VAR.EI VAR.S VAR.GJ]),20));

end
```

6.2.3 Function for trimmer with VLM

```
function [CL] = trimmerWITHVLM(VAR,CTRL)

% NOTE: Uses half wing
%% Change a3 to VLM

% Establishing variables from structured matrix

c      = VAR.Cr;         % Chord
lam    = VAR.lam;        % Taper ratio
S      = VAR.S;          % Semi span
e      = VAR.e;          % Elastic axis location relative to quarter chord
cLam   = VAR.cLam;       % sweep in degrees
Np     = VAR.Np;         % Number of panels
DY     = VAR.DY/VAR.S;   % Dimensionless panel span
y_L    = VAR.y_L;        % Panel position along one wing
c_Y    = VAR.c_y((VAR.Np/2)+1:end); % chord along the span due to taper
Sa     = VAR.Sa;         % Wing area
HSa    = VAR.HSa;        % Half wing area (USE)
AR     = VAR.AR;         % Aspect ratio
mg     = VAR.mg;         % Aircraft weight 10 ton aircraft
Vinf   = VAR.Vinf;       % m/s
rho    = VAR.rho;        % kg/m^3 at 20,000 ft

% Number of bending and torsional model
N      = VAR.N; %Bending polynomials
M      = VAR.M; %Torsion polynomials


%% ESTABLISHING INITIAL CONDITIONS OF A RIGID WING
% ESTIMATING Lift curve slope from VLM
% Rigid wing, no bending/torsion
```

```
VAR.theta = 0;
VAR.wd = 0;
VAR.alphaeVLM = 2*pi/180;    % Random alpha for Cla equation
[Li,plt] = VLMfunction(VAR,CTRL);
clear VAR.theta VAR.wd VAR.alphaeVLM
a3 = plt.Cla1;

% a3 = plt.Cla1;
a3=2*pi*AR/(2+sqrt(4+AR^2)); % Lift Curve slope 3D approximation for NON SWEEP WING

trimstep=1;
% Alpha of a rigid wing, no bending or twist.
alpha(trimstep) = mg/(0.5*rho*(Vinf^2)*(Sa)*a3);                %% FULL SPAN

% Initial lift estimate with respect to varying chord.
Li=0.5*rho*Vinf^2*(c*(1-lam*(y_L)))*a3*alpha(trimstep);          %% HALF SPAN

% Initial todal lift check.
Ltot_R = trapz(y_L,Li)*S*2;    %*2 as S is half wing span        %% FULL SPAN

formatSpec = 'TrimStep 0, Ltot %f, mg %f, alpha %f, \n';
fprintf (formatSpec, Ltot_R, mg, alpha*180/pi)

% Rigid wing CL along span
CL_y_R = Li./(0.5*rho*Vinf.^2*(c*(1-lam*y_L)));                 %% HALF SPAN

% Calculating initial displacements due to lift and moment.
% Lift   -  Bending
% Moment -  Torsion
Mi=e*c*Li;                                                      %% ALL HALF SPAN
for i=1:N
    psi_i(i,:) =(y_L).^(i+1);           % ith bending function wing
    psi_id(i,:)=((i+1)/S)*((y_L).^i);   % first derivative of the ith bending function
of the wing
    F(i,:)=trapz(y_L,Li.*(psi_i(i,:)))*S;
end
for i=1:M
    phi_i(i,:)= (y_L).^(i);             % ith torsion function wing
    F(i+N,:)=trapz(y_L,Mi.*(phi_i(i,:)))*S;
end

% Determining displacements due to twist and bending
eta    = VAR.Kq\F;
% Spanwise twist and bending for the first trim step
theta = phi_i'*eta(N+1:N+M);       % Torsion
wd    = psi_id'*eta(1:N);          % Bending                    %% BOTH HALF SPAN

% Resultant angle of attack due to initial alpha, bending and torsion.
% (effective angle of attack or elastic angle.)
alphae = alpha(trimstep) + theta*cos(cLam) - wd*sin(cLam);


% Recomputing lift along the span with new effective angle of attack.
Li    = (0.5*rho*(Vinf^2)*(c*(1  -lam*(y_L)))*a3).*alphae';                    %% HALF
SPAN
Ltot = trapz(y_L,Li)*S*2;     % Calculate total wing lift                  %% FULL SPAN
formatSpec = 'TrimStep %i, Ltot %f, mg %f, Lover %f, alpha %f, alphae(NP) %f\n';
fprintf (formatSpec,trimstep, Ltot, mg, Ltot-mg, alpha, alphae(Np/2)*180/pi)

%% START OF TRIM LOOP
for trimstep = 2:400  % Start at 2 as alpha(1) has been stated previously


% -- Elastic angle of attack for one wing --

% Gives matrix [N/2 +1 x 1] where first row is y_L = 0;
% - alpha(trimstep -1) is the alpha corrected in previous iteration.
% - theta and wd have also been recalculated
alphae = alpha(trimstep-1) + theta*cos(cLam) - wd*sin(cLam);                %% HALF
SPAN
% Need to convert this into [alphae;alphae]
```

```
%% Sub-routine for VLM
% This needs to include movement of points due to bending and torsion
% Calculates new lift due to the displacement.

% Note, this is incidence at points A and B NOT C
VAR.alphaeVLM = [flipud(alphae(2:end));alphae(1:end)];                          %% FULL
SPAN
VAR.wd    = wd(1:end);
VAR.theta = theta(1:end);

[Li,plt,y_C]  = VLMfunction(VAR,CTRL);                                              %% FULL
SPAN
a3        = plt.Cla1;

% Integrating for total lift.(Summation for VLM)
Ltot(1,trimstep) = sum(Li);                                                     %% FULL SPAN

% Finding difference between desired lift (mg) and actual.
Lover = Ltot(1,trimstep) - mg;                                                  %% FULL SPAN

% Calculating angle of attack which requires to be reduced or increased for
% desired lift (mg)
aover = Lover/((1/2)*rho*(Vinf^2)*(Sa)*a3);                                     %% FULL SPAN

% Reducing alpha in terms of calculated aover (anew = a - aover*RF)
% where RF is a relaxation factor to aid convergence.
alpha(trimstep) = alpha(trimstep-1) - (aover*CTRL.relaxation);

formatSpec = 'TrimStep %i, Ltot %f, mg %f, Lover %f, aover %f, alpha %f\n';
fprintf (formatSpec,trimstep, Ltot(1,trimstep), mg, Lover, aover,
alpha(trimstep)*180/pi)

% Recalulating displacements due to new angle of attack, hence lift(force),
% to be inputted into next iteration
% Moment calculation where e is elastic axis with respect to the quater
% chord.

% Note this is lift at C not A and B
LiHSC   = Li((Np/2)+1:end)';   % Trimmer works in half span, this is the half  %% ALL
HALF SPAN

% Reconverting back to A and B
LiHS(1,1) = LiHSC(1);   % A and B in wing centre average = first collocation points on
either side (e.g. (2+2)/2 = 2)
for i = 1:length(LiHSC)
    if i ~=length(LiHSC)
    LiHS(1,i+1) = (LiHSC(i+1) + LiHSC(i))/2;
    else
    LiHS(1,i+1) = LiHSC(i);
    end
end

Mi=e*c*LiHS;
for i=1:N
    psi_i(i,:)  = (y_L).^(i+1); % ith bending function wing
    psi_id(i,:) = ((i+1)/S)*((y_L).^i); % first derivative of the ith bending ufnction
of the wiing
    F(i,:)      = trapz(y_L,LiHS.*(psi_i(i,:)))*S;
end
for j=1:M
    phi_i(j,:)  = (y_L).^(j); % ith torsion function wing
    % j+N for indexing force due to torsion in the same matrix as bending
    F(j+N,:)    = trapz(y_L,Mi.*(phi_i(j,:)))*S;
end

% Solve system of equations for deflections "eta"
eta   = VAR.Kq\F;
% Spanwise displacements of bending and twist.
theta = phi_i'*eta(N+1:N+M);         % (2:end) as VLM does not have points directly on
root
wd    = psi_id'*eta(1:N);            % Displacements due to every mode.
```

```
% Convergence criterion
if abs(Lover)<mg/100, break, end % End loop condition
end


% Total lift coefficient
switch CTRL.spacing
    case 'uniform'

    CL       = Ltot(1,end)/(0.5*rho*(Vinf^2)*(Sa));                              %% FULL
SPAN
    CL_y     = LiHSC(1:end)./(0.5*rho*(Vinf^2)*(c*(1-
lam*((y_C((Np/2)+1:end)/S))))*DY*S);                  %% HALF SPAN
    CL_sweep = (CL_y/CL)';
    case 'cosine'
    CL       = Ltot(1,end)/(0.5*rho*(Vinf^2)*(Sa));                              %% FULL
SPAN
    for i = 1:length(LiHSC)
    CL_y(1,i) = LiHSC(i)/(0.5*rho*(Vinf^2)*(c*(1-
lam*(y_C((Np/2)+i)/S)))*DY(i)*S);                  %% HALF SPAN
    end
    CL_sweep = (CL_y/CL)';
end

figure; set(gcf,'color','w');
% Elastic Cl (Li calculated within loop)
plot (y_C((Np/2)+1:end)/S,CL_y,'LineWidth',2,'color',[0.8 0 0]); hold on
% Elastic Cl/ Rigid Cl
plot(y_C((Np/2)+1:end)/S,CL_sweep,'--','LineWidth',2,'color',[0 0.8 0]); hold on
% Rigid Cl distributed.
plot (y_C((Np/2)+1:end)/S,CL_y_R(2:(Np/2)+1),'--r','LineWidth',2)
xlabel ('Dimensionless span (y/S)'); ylabel ('Lift coefficeint (C_{L})')
legend('  C_{L,elastic}','  C_{L,elastic}/C_{L,rigid}', '  C_{L,rigid}')
set(gca,'FontName','Times New Roman'); grid minor
title('CL at colocation points')



figure; set(gcf,'color','w');
plot (y_L,alphae*180/pi,'LineWidth',2,'color',[0.8 0 0])
xlabel ('Dimensionless span (y/S)'); ylabel ('Aeroelastic angle (deg)')
set(gca,'FontName','Times New Roman'); grid minor

figure; set(gcf,'color','w');
plot (alpha*180/pi,'LineWidth',2,'color',[0.8 0 0]);
xlabel ('Iteration'); ylabel ('Pitch angle (deg)')
set(gca,'FontName','Times New Roman'); grid minor

figure; set(gcf,'color','w');
plot (y_L,theta*180/pi,'LineWidth',2,'color',[0.8 0 0])
xlabel ('Dimensionless span (y/S)'); ylabel ('Torsion angle (deg)')
set(gca,'FontName','Times New Roman'); grid minor

figure; set(gcf,'color','w');
plot (y_L,wd,'-r','LineWidth',2,'color',[0.8 0 0])
xlabel ('Dimensionless span (y/S)'); ylabel ('Bending (m)')
set(gca,'FontName','Times New Roman'); grid minor


end
```

6.2.4 Function for trimmer without VLM

```
function [CL] = trimmerNOVLM(VAR,CTRL)

% Establishing variables from structured matrix
c      = VAR.Cr;        % Chord
```

```
lam    = VAR.lam;       % Taper ratio
S      = VAR.S;         % Semi span
e      = VAR.e;         % Elastic axis location relative to quarter chord
cLam   = VAR.cLam;      % sweep in degrees
Np     = VAR.Np;        % Number of panels
DY     = VAR.DY/VAR.S;  % Dimensionless panel span
y_L    = VAR.y_L;       % Panel position along one wing
c_Y    = VAR.c_y((VAR.Np/2)+1:end); % chord along the span due to taper
Sa     = VAR.Sa;        % Wing area
AR     = VAR.AR;        % Aspect ratio
mg     = VAR.mg;        % Aircraft weight 10 ton aircraft
Vinf   = VAR.Vinf;      % m/s
rho    = VAR.rho;       % kg/m^3 at 20,000 ft

% Number of bending and torsional model
N      = VAR.N;         % Bending polynomials
M      = VAR.M;         % Torsion polynomials

% going to use VLM instead to get this.
a3=2*pi*AR/(2+sqrt(4+AR^2)); % Lift Cyrve slope 3D approximation
% [] = VLMworks()

% Establishing initial conditions of aircraft without elastic wing (rigid)
trimstep=1;
% Calculating alpha with respect to a rigid wing
alpha(trimstep) =mg/(0.5*rho*(Vinf^2)*Sa*a3);
% Estimating lift due to initial alpha with respect to taper
Li=0.5*rho*Vinf^2*(c*(1-lam*(y_L)))*a3*alpha(trimstep);
% Rigid wing CL along span
CL_y_R=Li./(0.5*rho*Vinf.^2*(c*(1-lam*y_L)));
% Rigid Ltot
Ltot_R = trapz(y_L,Li)*S*2;

% Calculating bending and torsion displacements due to lift and moment
% forces (lift - bending, moment - torsion)
% Mi=e*c_Y.*Li;
Mi=e*c*Li;
for i=1:N
    psi_i(i,:) =(y_L).^(i+1); % ith bending function wing
    psi_id(i,:)=((i+1)/S)*((y_L).^i); % first derivative of the ith bending ufnction of
the wiing
    F(i,:)=trapz(y_L,Li.*(psi_i(i,:)))*S;
end
for j=1:M
    phi_i(j,:)= (y_L).^(j); % ith torsion function wing
    F(j+N,:)=trapz(y_L,Mi.*(phi_i(j,:)))*S;
    % j + N as to index torsion force in same matrix as bending
end

% Determining bend and twist of the wing
eta    = VAR.Kq\F;
% Spanwise twist and bending for the first trim step
theta = phi_i'*eta(N+1:N+M); % Twist
wd     = psi_id'*eta(1:N);    % Bending

% Resolving entire angle of attack due to initial, bending and twist angle
% of attack.
alphae = alpha(trimstep) + theta*cos(cLam) - wd*sin(cLam);

% Recomputing lift accross span for wing with deformation
Li=(0.5*rho*Vinf^2*(c*(1-lam*(y_L)))*a3).*alphae';
Ltot=trapz(y_L,Li)*S*2;        % Calculate total wing lift

% Displaying initial iteration
formatSpec = 'TrimStep %i, Ltot %f, mg %f, Lover %f, alpha %f, alphae(NP) %f\n';
fprintf (formatSpec,trimstep, Ltot, mg, Ltot-mg, alpha, alphae(Np/2))

% Begin iterations for changing of lift due to deformation, which in turn
% changes the deformation.

for trimstep=2:199
    % Elastic angle of attack for one wing ( AoA effective )
```

```
    alphae = alpha(trimstep-1) + theta*cos(cLam) - wd*sin(cLam);
    % lift distribution for elastic wing
    Li=(0.5*rho*Vinf^2*(c*(1-lam*(y_L)))*a3).*alphae';
    Ltot=trapz(y_L,Li)*S*2; %*2 as working with half a wing

    % Lift disparity between desired lift (mg) and actual lift
    Lover  = Ltot - mg;
    % ROOT angle of attack over/under correction
    aover=Lover/(.5*rho*Vinf^2*Sa*a3);

    % Recompute alpha with correction of aover, multiplied by a relaxation
    % variable to aid convergence.
    alpha(trimstep) = alpha(trimstep-1)-aover*CTRL.relaxation;

    % Reprinting iteration
    formatSpec = 'TrimStep %i, Ltot %f, mg %f, Lover %f, aover %f, alpha %f\n';
    fprintf (formatSpec,trimstep, Ltot, mg, Lover, aover, alpha(trimstep))

    % Recompute displacments
%     Mi=e*c_Y.*Li;
    Mi=e*c*Li;
    clear psi_i psi_id phi_i F
    for i=1:N
        psi_i(i,:) =(y_L).^(i+1); % ith bending function wing
        psi_id(i,:)=((i+1)/S)*((y_L).^i); % first derivative of the ith bending ufnction
of the wiing
        F(i,:)=trapz(y_L,Li.*(psi_i(i,:)))*S;
    end
    for i=1:M
        phi_i(i,:)= (y_L).^(i); % ith torsion function wing
        F(i+N,:)=trapz(y_L,Mi.*(phi_i(i,:)))*S;
    end

    % Solve system of equations for deflections "eta"
    eta=VAR.Kq\F;
    % Spanwise bending and twist
    theta=phi_i'*eta(N+1:N+M);
    wd=psi_id'*eta(1:N);

    if abs(Lover)<mg/100, break, end

end

% Total lift coefficient
CL=Ltot/(0.5*rho*(Vinf^2)*Sa);
% spanwise lift coefficient
CL_y=Li./(0.5*rho*(Vinf^2)*(c*(1-lam*(y_L))));
CL_sweep=(CL_y/CL)';


figure
set(gcf,'color','w');
% Elastic Cl (Li calculated within loop)
plot (y_L,CL_y,'LineWidth',2,'color',[0.8 0 0])
hold on
% Elastic Cl/ Rigid Cl
plot(y_L,CL_sweep,'-','LineWidth',2,'color',[0 0.8 0])
hold on
% Rigid Cl distributed.
plot (y_L,CL_y_R(1:(Np/2) +1),'--r','LineWidth',2)
xlabel ('Dimensionless span (y/L)')
ylabel ('Lift coefficeint (CL)')
legend('  C_{L,elastic}','  C_{L,elastic}/C_{L,rigid}', '  C_{L,rigid}')
set(gca,'FontName','Times New Roman');
grid on
grid minor
hold off

figure
set(gcf,'color','w');
plot (y_L,alphae*180/pi,'LineWidth',2,'color',[0.8 0 0])
xlabel ('Dimensionless span (y/L)')
```

```
ylabel ('Aeroelastic angle (deg)')
set(gca,'FontName','Times New Roman');
grid on
grid minor

figure
set(gcf,'color','w');
plot (alpha*180/pi,'LineWidth',2,'color',[0.8 0 0])
xlabel ('Iteration')
ylabel ('Pitch angle (deg)')
set(gca,'FontName','Times New Roman');
grid on
grid minor
hold off

figure
set(gcf,'color','w');
plot (y_L,theta*180/pi,'LineWidth',2,'color',[0.8 0 0])
xlabel ('Dimensionless span (y/L)')
ylabel ('Torsion angle (deg)')
set(gca,'FontName','Times New Roman');
grid on
grid minor
hold off

figure
set(gcf,'color','w');
plot (y_L,wd,'-r','LineWidth',2,'color',[0.8 0 0])
xlabel ('Dimensionless span (y/L)')
ylabel ('Bending (m)')
set(gca,'FontName','Times New Roman');
grid on
grid minor
hold off



end
```

6.2.5 VLM function for trimmer

```
%Vortex lattice method
function [Li,plt,y_C] = VLMfunction(VAR,CTRL)
% NOTE: Uses full wing
%% Establishing variables in function
alpha  = VAR.alphaeVLM;       % AoA in radians                          %% FULL SPAN
Vinf   = VAR.Vinf;
rho    = VAR.rho;
qinf   = VAR.qinf;            % Dynamic head

% Geometric parameters
cLam = VAR.cLam;       % Sweep angle
lam  = VAR.Cr/VAR.Ct;  % Taper ratio
S    = VAR.S;          % Semi span length                                %% HALF SPAN
Sa   = VAR.Sa;         % TOTAL Wing area                                 %% FULL SPAN

% Numerical parameters
Np   = VAR.Np; % number of panels
y    = [-fliplr(VAR.y(2:end)),VAR.y]; % Converting to full span          %% FULL SPAN
c_y  = VAR.c_y;                                                          %% FULL SPAN
DY   = VAR.DY;                                                           %% FULL SPAN

% Establishing displacements in full wingspan form
if length(alpha) ~= 1
theta = [flipud(VAR.theta(2:end));VAR.theta]';                           %% FULL SPAN
wd    = [flipud(VAR.wd(2:end));VAR.wd]';                                 %% FULL SPAN
else
```

```
    theta = VAR.theta;
    wd    = VAR.wd;
end

% Establish 1/4, 3/4, 1/2 chord and LE, TE lines for the wing
% Calculating x position for each
% - Origin on half chord at root chord.
% - Using half chord as reference line
x_hc  = sign(y).*y.*tan(cLam);   % Not multplied by cos(theta) as e is 0.25*c away from
quater chord. E.g. half chord
x_tqc = x_hc + ((c_y./4).*cos(theta));
x_qc  = x_hc - ((c_y./4).*cos(theta));
x_te  = x_hc + ((c_y./2).*cos(theta));
x_le  = x_hc - ((c_y./2).*cos(theta));

z_hc  = wd;  % On EA so no twist
z_tqc = wd + ((c_y./4).*sin(theta));
z_qc  = wd - ((c_y./4).*sin(theta));
z_te  = wd + ((c_y./2).*sin(theta));
z_le  = wd - ((c_y./2).*sin(theta));

%% Colocation points and reference points
% A,B on quater chord.
% C on 3/4 chord (AC).

y_A = y(1:1:end-1); % Start point and every point after.
y_B = y(2:1:end); % Start at second point and every  point after.
y_C = (y_A + y_B)/2; % inbetween A and B

x_A = x_qc(1:1:end-1);    % |
x_B = x_qc(2:1:end);      % |> On Quater chord.
x_C = (x_tqc(1:1:end-1)+ x_tqc(2:1:end))/2; % On 3 quater chord

% --- x_A etc as it is with respect to the origin which is located at the half
% chord.
z_A = z_qc(1:1:end-1);
z_B = z_qc(2:1:end);
z_C = (z_tqc(1:1:end-1)+ z_tqc(2:1:end))/2;

% Matrixes for colocation points
A = [x_A; y_A; z_A];
B = [x_B; y_B; z_B];
C = [x_C; y_C; z_C];

%% Biot-Savert: influence coefficient calculations
% r0,r1,r2
% Need to loop for all points. A(1,1) will be paired with B(1,1), C(1,1).

% Need to loops to calculate each HSV with respect to other HSV. Should get
% n x n matrix of AIC
for j = 1:1:(Np)  % Defining colocation point for calculations
    for k = 1:1:(Np)  % Changing colocation points for colocation pt,i

    r_0 = [(x_B(1,k) - x_A(1,k));(y_B(1,k) - y_A(1,k));(z_B(1,k) - z_A(1,k))];
    r_1 = [(x_C(1,j) - x_A(1,k));(y_C(1,j) - y_A(1,k));(z_C(1,j) - z_A(1,k))];
    r_2 = [(x_C(1,j) - x_B(1,k));(y_C(1,j) - y_B(1,k));(z_C(1,j) - z_B(1,k))];



VAB(j,k,:)=((1/(4*pi))*((cross(r_1,r_2))/((norm(cross(r_1,r_2))).^2))*(dot(r_0,((r_1/(no
rm(r_1)))-(r_2/(norm(r_2))))))');

    VAcomp1 = (1/(4*pi));
    VAcomp2 = (([ 0, (z_C(1,j)-z_A(1,k)) , (y_A(1,k)-y_C(1,j))]));
    VAcomp3 =  1/(((z_C(1,j)-z_A(1,k))^2) + ((y_A(1,k)-y_C(1,j))^2));
    VAcomp4 = (1 + ((x_C(1,j)-x_A(1,k))/(( ((x_C(1,j)-x_A(1,k))^2) + ((y_C(1,j)-
y_A(1,k))^2) + ((z_C(1,j)-z_A(1,k))^2) )^(1/2))));
    VAI(j,k,:) = VAcomp2.*VAcomp1.*VAcomp3.*VAcomp4;

    VBcomp1 = (-1/(4*pi));
    VBcomp2 = (([ 0, (z_C(1,j)-z_B(1,k)) , (y_B(1,k)-y_C(1,j))]));
    VBcomp3 = 1/(((z_C(1,j)-z_B(1,k))^2) + ((y_B(1,k)-y_C(1,j))^2));
```

```
    VBcomp4 = (1 + ((x_C(1,j)-x_B(1,k))/(( ((x_C(1,j)-x_B(1,k))^2) + ((y_C(1,j)-
y_B(1,k))^2) + ((z_C(1,j)-z_B(1,k))^2) )^(1/2))));

    VBI(j,k,:) = VBcomp2.*VBcomp1.*VBcomp3.*VBcomp4;
    end
end
clear VAcomp1 VAcomp2 VAcomp3 VAcomp4 VBcomp1 VBcomp2 VBcomp3 VBcomp4
IC =VAB +VAI + VBI;
% Sorting Influence coefficients into x,y,z components
ICx = squeeze(IC(:,:,1));
ICy = squeeze(IC(:,:,2));
ICz = squeeze(IC(:,:,3));

% Calculate normals for each panel (V.n = 0;)
for i = 1:1:(Np)
    n = cross(A(:,i)-C(:,i),A(:,i)-B(:,i)); % unit normal from plane formed by A,B,C
    nval(:,i) = n/((dot(n,n))^(1/2));
end

% Calculate aerodynamic influence coefficient matrix
for i=1:1:(Np)
    for j=1:1:(Np)
        % Multipling normal vectors with respective AIC direction (eg, x
        % normal component with AICx)
        AIC(i,j)=(((nval(1,j)*ICx(i,j)) + (nval(2,j)*ICy(i,j)) + (nval(3,j)*ICz(i,j))));
    end
end

% Alpha currently for every A and B point.Interpolating for point C.
% Note, if alpha length = 1 then it is the initial value
if length(alpha) ~=1
    for i = 1:length(y)-1
        alphaC(i,1) = (alpha(i,1)+ alpha(i+1,1))/2; % Same as interp between 2 points
    end
else
    alphaC = repmat(alpha,length(y_C),1);
end

% Calculate incident flow velocity and apply V.n=0 boundary condition
% cos(a) = x direction, sin(a) = y direction
Boundcond = -((Vinf*((cos(alphaC').*nval(1,:)) + (sin(alphaC').*nval(3,:))))); %% FULL
SPAN
% ' to convert alpha to Np x 1 to multiply by nval(1,:) which is Np x 1.
% Where alpha is AoAe of every panel and nval is unit normal of every panel.
% ^ gives the resolved velocity components at every panel

% Circulation at colocation points
circulation = AIC\Boundcond';

%% Method 1: From Dr Barakos

switch CTRL.spacing
    case 'uniform'
    qinf       = 0.5*rho*Vinf*Vinf; % dynamic head
    Li         = rho*Vinf.*circulation.*DY; % calculate local
lift                        %% FULL SPAN
    Ltot1      =
sum(Li);                                                          %% FULL SPAN
    Cltot1     = Ltot1/(qinf*Sa); % wing lift coefficient                              %%
FULL SPAN
    Cla_parts  = Cltot1/(sum(alpha)/length(alpha)); % Compute Lift-slope         %% FULL
SPAN
    plt.Cla1   = sum(Cla_parts)/length(Cla_parts);                            %% FULL
SPAN
    plt.Cltot = Cltot1;
    % Calculate local lift coefficients
    for i=1:Np
        localc(i)=(c_y(i)+c_y(i+1))/2.;
        Cl(i)=Li(i)/(qinf*localc(i)*DY);
    end

    case 'cosine'
```

```
        qinf = 0.5*rho*Vinf*Vinf; % Dynamic head
        for i=1:Np
            Li(i,1)     = rho*Vinf*circulation(i)*DY(i); % Calculate local
lift                        %% FULL SPAN
            localc(i)   = (c_y(i)+c_y(i+1))/2.;
            Cl(i)       = Li(i)/(qinf*localc(i)*DY(i));
        end
        Ltot1      = sum(Li);                                                %%
FULL SPAN
        Cltot1     = Ltot1/(qinf*Sa); % Wing lift coefficient
        Cla_parts  = Cltot1/(sum(alpha)/length(alpha)); % Compute Lift-slope      %%
FULL SPAN
        plt.Cla1   = sum(Cla_parts)/length(Cla_parts);                       %%
FULL SPAN
        plt.Cltot  = Cltot1;
end
end
```

6.3 Assignment 3 code

6.3.1 Main code

```
%% ASSIGNMENT 3: K-Method

clc, clear, close all
%% Controls
CTRL.store = '5';       % 5 or 7e (which mass config)
CTRL.n     = 100;        % Number of segmentations along span
CTRL.nk    = 100;        % Number of segmentations of frequency

% Modes
VAR.N = 3;                      % Bending number of assumed shapes
VAR.M = 2;                      % Torsional number of assumed shapes

%% Establishing variables of wing
VAR.mw      = 1.578501;         % wing mass                 (kg) ( Altered)
VAR.c       = 0.2032;           % Chord at 70% span         (m)
VAR.b       = VAR.c/2;          %                           (m)
VAR.S       = 1.2192;           % Semi span                 (m)
VAR.EA      = 0.437;            % Elastic axis              (% of chord)
VAR.IA      = 0.454;            % Inertial axis (CG)        (% of chord)
VAR.EI      = 4.0376e+02;       % Bending Stiffness         (Nm^2)
VAR.GJ      = 198.5813;         % Torsional stiffness       (Nm^2)
VAR.mu      = VAR.mw/VAR.S;     % Mass per length           (kg/m)
VAR.X       = (4.349e-3)/VAR.S;         % MoI per twist about EA    (kgm^2)
VAR.rho     = VAR.mu/(pi*(VAR.b^2)*32.6); %                   (kg/m^3)
VAR.xa      = (VAR.IA - VAR.EA)*VAR.c/VAR.b;  % Distance between Elastic and CG axis
normilized by semichord /VAR.b

% Store
switch CTRL.store
    case '4'
    VAR.ms = 0.636*VAR.mw;          % Mass of store
    VAR.xs = 0.625*VAR.b;           % Distance between store CG and wing EA.  (-ve
infront of elastic axis)
    VAR.Is = 1.91*VAR.X*VAR.S;      % Store intertia
    case '5'
    VAR.ms = 0.636*VAR.mw;          % Mass of store
    VAR.xs = 0.687*VAR.b;           % Distance between store CG and wing EA.  (+ve
behind EA)
    VAR.Is = 2.68*VAR.X*VAR.S;      % Store intertia
    case'7e'
```

```
    VAR.ms = 0.954*VAR.mw;          % Mass of store
    VAR.xs = 0.034*VAR.b;           % Distance between store CG and wing EA.  (+ve
behind EA)
    VAR.Is = 1.56*VAR.X*VAR.S;      % Store intertia
    case 'nomass'
    VAR.ms = 0;
    VAR.xs = 0;
    VAR.Is = 0;

end

VAR.y_L     = 0:    1/CTRL.n:      1;
VAR.SSIND   = 0:    VAR.S/CTRL.n:  VAR.S;     % Store position
VAR.K       = 0.01:  1.49/CTRL.nk:   1.5;       % Reduced frequency k

% Elastic axis position is known. a = EC/b  - 0.5
a = 2*VAR.EA  - 0.5;

for i = 1:length(VAR.SSIND) % Looping over for store positions.


% Store position
VAR.Ss = VAR.SSIND(i);

% MC is structured array of matrix components (DEL, D, B etc)
% Function for mass and stiffness matrix

[Mqs,Mqw,Kq,MC] = MSmatrixfun(VAR,CTRL);

% Debug (Checking if structure matrices are working)
if i == 1
    [Vect,LAM] = eig(Kq,Mqw);
    tomeg = sqrt(diag(LAM));
    freq  = tomeg/(2*pi);
end

            for ii = 1:length(VAR.K)           % Looping over for reduce freq k

            % Theodorsen function calculations
            theo =
besselk(1,(1i*VAR.K(ii)))./(besselk(0,(1i*VAR.K(ii)))+besselk(1,1i*VAR.K(ii)));
            F(ii) = real(theo); % For plot checks
            G(ii) = imag(theo);

            % Calculate Astar, Bstar, Cstar from DEL, D etc.
            Astar =   2*pi*VAR.b*(VAR.K(ii)^2)*([MC.DEL,          a*VAR.b*MC.C;
                                                 a*VAR.b*MC.CT,
(VAR.b^2)*((a^2)+(1/8))*MC.D]);

            Bstar =  -2*pi*(VAR.K(ii))*j*([2*theo*MC.DEL,                -VAR.b*(1 +
2*(0.5 - a)*theo)*MC.C;
                                           2*VAR.b*(0.5 + a)*theo*MC.CT, (VAR.b^2)*(0.5 -
a)*(1- 2*(0.5+a)*theo)*MC.D]);

            Cstar =  -2*pi*VAR.b*([zeros(VAR.N,VAR.N), -2*theo*MC.C;
                                   zeros(VAR.N,VAR.M)', -VAR.b*(1+2*a)*theo*MC.D]);

             A_hat = Astar + Bstar + Cstar;

            [muvect, muval]  = eig(Kq\((Mqw+Mqs) +
(0.5*VAR.rho*((VAR.b^2)/(VAR.K(ii)^2))).*A_hat));
             mu              = diag(muval);

             % Calculation of outputs
             % Data => [rows = modes, cols = reduce freq, extrusion = store positions].
             omega(:,ii) = sqrt(1./real(mu));                % Frequencies from real
part of eigenvalues
             g(:,ii)     = ((omega(:,ii).^2).*imag(mu)/j); % Damping from imaginary part
of eigenvalues
             U(:,ii)     = omega(:,ii).*VAR.b/VAR.K(ii);   % corresponding speed (from
reduced freq equation)
            end
```

```
    ganal(i,:) = -0.5*imag(g(2,:));
    m = 2;  % Mode two
    int = find(-0.5*imag(g(m,:))>0);
    if isempty(int)
        Uf(i) = NaN;
    else
    int = int(end);     % Lowest velocity value
    int = polyfit([U(m,int), U(m,(int+1))], [(-0.5*imag(g(m,int))), (-
0.5*imag(g(m,(int+1))))],1);
    Uf(i) = abs(int(2))/abs(int(1));
    end

    % Minimum flutter speed for corresponding mass position
    if i > 1
        if Uf(i) < Uf(i-1)
           Ufstore = Uf(i);
           omegastore = omega;
           gstore     = g;
           Ustore     = U;
           ind = i;
        end
    end
end

load('flutter_W5.mat')
if flutter_W5(1,1) < 0
    flutter_W5(1,1) =0;
end
load('flutter_W7.mat')
if flutter_W7(1,1) < 0
    flutter_W7(1,1) =0;
end

Ufnorm = Uf/(Uf(1));    % Normilizing by no massed wing e.g. mass at root

figure(1); set(gcf,'color','w')
plot(VAR.SSIND/VAR.S,Ufnorm,'color',[0.8 0 0], 'LineWidth',2)
switch CTRL.store
    case '5'
        hold on; plot(flutter_W5(:,1),flutter_W5(:,2),'-s','color',[0 0.8 0],
'LineWidth',2); hold on
        title('Normalized flutter speed over span for mass 5')
    case '7e'
        hold on; plot(flutter_W7(:,1),flutter_W7(:,2),'-s','color',[0 0.8 0],
'LineWidth',2); hold on
        title('Normalized flutter speed over span for mass 7e')
end
legend('Results','Experimental data')
daspect([1 1 1])
xlabel('Spanwise position (-)'); ylabel('$$U_F/U_{F0}$$','Interpreter','Latex');
set(gca,'FontName','Times New Roman','Fontsize',12); grid minor


switch CTRL.store
    case '5'
% Frequency and damping plots
figure(2); set(gcf,'color','w')
plot(Ustore(1,:),omegastore(1,:)/2*pi,'color',[0 0.8 0],'Linewidth',2); hold on
plot(Ustore(2,:),omegastore(2,:)/2*pi,'color',[0.8 0 0],'Linewidth',2); hold on
plot(Ustore(3,:),omegastore(3,:)/2*pi,'color',[0 0 0.8],'Linewidth',2); hold on
plot(repmat(Ufstore,1,500),1:500,'-.k')
legend('Mode 1','Mode 2','Mode 3','Flutter speed')
xlabel('Corresponding velocity (m/s)'); ylabel('Frequency (Hz)'); title('Frequency at
minimum flutter speed (Mass 5)')
xlim([0 Ufstore+0.1*Ufstore])
set(gca,'FontName','Times New Roman','Fontsize',12); grid minor

figure(3); set(gcf,'color','w')
plot(Ustore(1,:),-0.5*imag(gstore(1,:)),'color',[0 0.8 0],'Linewidth',2); hold on
plot(Ustore(2,:),-0.5*imag(gstore(2,:)),'color',[0.8 0 0],'Linewidth',2); hold on
plot(Ustore(1,:),-0.5*imag(gstore(3,:)),'color',[0 0 0.8],'Linewidth',2); hold on
```

```
plot(Ufstore,0,'s','MarkerSize',7,'MarkerFaceColor',[0.9 0 0],'MarkerEdgecolor',[0.5 0
0])
plot(0:Ufstore+0.1+Ufstore,zeros(1,length(0:Ufstore+0.1+Ufstore)),'--k','Linewidth',0.5)
legend('Mode 1','Mode 2','Mode 3','Flutter speed')
xlabel('Corresponding velocity (m/s)'); ylabel('-0.5 Damping'); title('Damping at
minimum flutter speed (Mass 5)')
xlim([0 Ufstore+0.1*Ufstore])
set(gca,'FontName','Times New Roman','Fontsize',12); grid minor


    case '7e'
figure(2); set(gcf,'color','w')
plot(Ustore(1,:),omegastore(1,:)/2*pi,'color',[0 0.8 0],'Linewidth',2); hold on
plot(Ustore(2,:),omegastore(2,:)/2*pi,'color',[0.8 0 0],'Linewidth',2); hold on
plot(Ustore(3,:),omegastore(3,:)/2*pi,'color',[0 0 0.8],'Linewidth',2); hold on
plot(repmat(Ufstore,1,500),1:500,'-.k')
legend('Mode 1','Mode 2','Mode 3','Flutter speed')
xlabel('Corresponding velocity (m/s)'); ylabel('Frequency (Hz)'); title('Frequency at
minimum flutter speed (Mass 7e)')
xlim([0 Ufstore+0.1*Ufstore])
set(gca,'FontName','Times New Roman','Fontsize',12); grid minor

figure(3); clf; set(gcf,'color','w')
plot(Ustore(1,:),-0.5*imag(gstore(1,:)),'color',[0 0.8 0],'Linewidth',2); hold on
plot(Ustore(2,:),-0.5*imag(gstore(2,:)),'color',[0.8 0 0],'Linewidth',2); hold on
plot(Ustore(1,:),-0.5*imag(gstore(3,:)),'color',[0 0 0.8],'Linewidth',2); hold on
plot(Ufstore,0,'s','MarkerSize',7,'MarkerFaceColor',[0.9 0 0],'MarkerEdgecolor',[0.5 0
0])
plot(0:Ufstore+0.1+Ufstore,zeros(1,length(0:Ufstore+0.1+Ufstore)),'--k','Linewidth',0.5)
legend('Mode 1','Mode 2','Mode 3','Flutter speed')
xlabel('Corresponding velocity (m/s)'); ylabel('-0.5 Damping'); title('Damping at
minimum flutter speed (Mass 7e)')
xlim([0 Ufstore+0.1*Ufstore])
set(gca,'FontName','Times New Roman','Fontsize',12); grid minor


end
```

6.3.2 Mass and stiffness function

```
%% Function for bending, torsional modes
function [Mqs,Mqw,Kq,MC] = MSmatrixfun(VAR,CTRL)


% % Establishing variables
N = VAR.N;                          % Bending number of assumed shapes
M = VAR.M;                          % Torsional number of assumed shapes
Ss = VAR.Ss;
%% Wing matrices
% Bending
for i = 1:N
        WV.PSI      = (VAR.y_L).^(i+1);
        WV.PSIstore = (Ss/VAR.S).^(i+1);
        WV.PSIDD    = ((i*(i+1))/VAR.S.^2)*(VAR.y_L.^(i-1));
        %Bending jth component for matrix multiplication
        for j = 1:N
            WV.PSIj      = (VAR.y_L).^(j+1);
            WV.PSIstorej = (Ss/VAR.S).^(j+1);
            WV.PSIDDj    = ((j*(j+1))/VAR.S.^2)*(VAR.y_L.^(j-1));
            DEL(i,j)     =  VAR.S*trapz  (VAR.y_L,(WV.PSI.*WV.PSIj));   % Wing mass
matrix (DEL): Symbolic integration of matix (dy) Kinetic
            B  (i,j)     =  VAR.S*trapz  (VAR.y_L,(WV.PSIDD.*WV.PSIDDj));   % Wing
stiffness matrix (B): Symbolic integration of matrix (dy) potenial
            DELS(i,j)    = (WV.PSIstore.*WV.PSIstorej);
        end
```

```
        for k = 1:M
            WV.PHIj       = (VAR.y_L).^(k);
            WV.PHIDj      = (k/VAR.S)*(VAR.y_L.^(k-1));
            WV.PHIstorej  = (Ss/VAR.S).^(k);
            C(i,k)        = -VAR.S*trapz(VAR.y_L,(WV.PSI.*WV.PHIj));    %*VAR.b
            Cs(i,k)       = (WV.PSIstore.*WV.PHIstorej);         %/VAR.b
        end
end
clear WV i j k

% Torsional modes
for i = 1:M
WV.PHI      = (VAR.y_L).^(i);
WV.PHIstore = (Ss/VAR.S).^(i);
WV.PHID     = ((i)/VAR.S)*(VAR.y_L.^(i-1));
            for j = 1:M
            WV.PHIj       = (VAR.y_L).^(j);
            WV.PHIstorej  = (Ss/VAR.S).^(j);
            WV.PHIDj      = (j/VAR.S)*(VAR.y_L.^(j-1));
            T (i,j)       = VAR.S*trapz(VAR.y_L,(WV.PHID.*WV.PHIDj));    % Wing stiffness
matrix, T
            D (i,j)       = VAR.S*trapz (VAR.y_L,(WV.PHI.*WV.PHIj));    % Wing mass
matrix, D
            Ds(i,j)       = (WV.PHIstore.*WV.PHIstorej);
            end
end

%% Combining matrices for eigan calculations

% Stiffness matrix => | B   0 |
%                     | 0   T |

Kq  = [VAR.EI*B  , zeros(N,M)  ; zeros(N,M)',  VAR.GJ*T];

% Mass matrix => Atotal = Awing + Astore
%                       = |Del      xa*b*C|  + |DelS    xs(Cs) |
%                         |xa*b*C'     D   |    |xs(Cs')    DS   |

Mqw  = [VAR.mu*DEL,   VAR.mu*VAR.xa*VAR.b*C;   % C*VAR.b
        VAR.mu*VAR.xa*VAR.b*C',     VAR.X*D];  % + mass store

Mqs  = [VAR.ms*DELS,       VAR.ms*VAR.xs*Cs;  VAR.ms*VAR.xs*Cs'     , VAR.Is*Ds];

% Matrix components
MC.DEL  = DEL;
MC.DELS = DELS;
MC.C    = C;
MC.CT   = C';
MC.Cs   = Cs;
MC.CsT  = Cs';
MC.D    = D;
MC.Ds   = Ds;
MC.B    = B;
MC.T    = T;

end
```