Development of a Tethered, High-Altitude, Rigid-Wing Platform

by

Joeleff T. Fitzsimmons

A thesis submitted in conformity with the requirements
for the degree of Master of Applied Science
The Institute for Aerospace Studies
University of Toronto

0-612-29366-1

Title: Development of a Tethered, High-Altitude, Rigid-Wing Platform

Degree: Master of Applied Science, November 1997

Name: Joeleff T. Fitzsimmons

The Institute for Aerospace Studies

University of Toronto

A tethered, high-altitude, rigid-wing, platform (THARWP) was studied as a possible improvement over the helium-filled, tethered aerostats currently in use for positioning payloads at a specified altitude. The possible gains associated with implementing the THARWP were increased operating altitude, improved positioning accuracy, and reduced operating cost. An equilibrium analysis was developed to evaluate the trim-state and the feasibility of the THARWP.

The THARWP was composed of a tether and a flight vehicle. In most research of previous tethered systems, the dynamic motions of the tether were neglected or minimally accounted for in the formulation of the dynamic stability analysis. Unlike these previous analyses, the THARWP stability analysis was based on a discrete-element tether model which allowed the evaluation of the tether motion and its interaction with the flight vehicle motion. The THARWP design was then optimized for the maximum trim-state altitude as well as the most stable dynamic response. Although the optimized THARWP was naturally stable, its stability characteristics were further enhanced by the inclusion of control gains which defined the behaviour of the THARWP's control system. Finally, the implementation of the control system was discussed along with recommendations for how to improve upon the final control system.

ACKNOWLEDGMENTS

I would like to express my appreciation to all the people that provide technical and moral support throughout the duration of my research. Specifically, I would like to thank Dr. J. D. DeLaurier for providing technical guidance when it was most required, and Lauren Knoblauch for unwavering support and tireless editing.

A postgraduate scholarship granted by the Natural Sciences and Engineering Research Council of Canada is also gratefully acknowledged.

TABLE OF CONTENTS

Table Page

The purpose of this work is to assess the feasibility of a new tethered, high-altitude, rigid-wing, platform (THARWP) that will provide a stable means for positioning payloads at high-altitudes. Current high-altitude tethered vehicles have inherent limitations that impact on their overall performance. The most significant limitation is that tethered vehicles become less capable of accurate station-keeping with increasing tether lengths and may require control augmentation to maintain the payload position to within tolerable limits. Two other significant limitations specific to the operation of helium-filled, tethered aerostats are: (a) for a given payload mass, the size and the cost of the aerostat dramatically increases with the increase in the desired altitude; and (b) large aerostat drag forces cause significant payload-position deviations in the presence of wind-speed variations. These limitations provided the motivation for investigating the feasibility of the THARWP to satisfy future high-altitude payload positioning needs. This research addresses these needs and describes the development of the THARWP, an alternative tethered vehicle capable of high-altitude station-keeping missions.

Currently, there are many different tasks that require the elevated positioning of a payload that is either towed behind a moving vehicle or carried aloft by an aerostat (or some other lifting vehicle) tethered to the ground. These tasks include sonar devices towed behind boats or submarines, air-quality monitoring devices towed behind helicopters, and surveillance, telecommunication, or high-altitude atmospheric monitoring devices carried to a desired altitude by aerostats tethered to the ground. Traditional kites are rarely used for payload positioning because of their inefficient flexible single-surface design. The THARWP is similar to other kites in that—assuming there is adequate ambient wind speed—the THARWP derives its vertical supporting force from aerodynamic lift rather than from the buoyancy of helium. Thus, in order to improve upon the capabilities of current tethered aerostats, a simple, yet highly efficient, kite will be designed.

The THARWP (Figure 1) consists of a flexible tether and a rigid-wing lifting body (the "flight vehicle"). The flexible tether is anchored to the ground at the attachment point ("attachment point") and attached to the flight vehicle *via* the confluence point. The position of the confluence point relative to the flight vehicle's center of gravity (CG) is defined by the rigid link, r_{cp}. This rigid link allows the flight vehicle to rotate freely about the confluence point. Thus, the motion of the flight vehicle is characterized by the motion of the confluence point and the rotation of the flight vehicle about the confluence point.

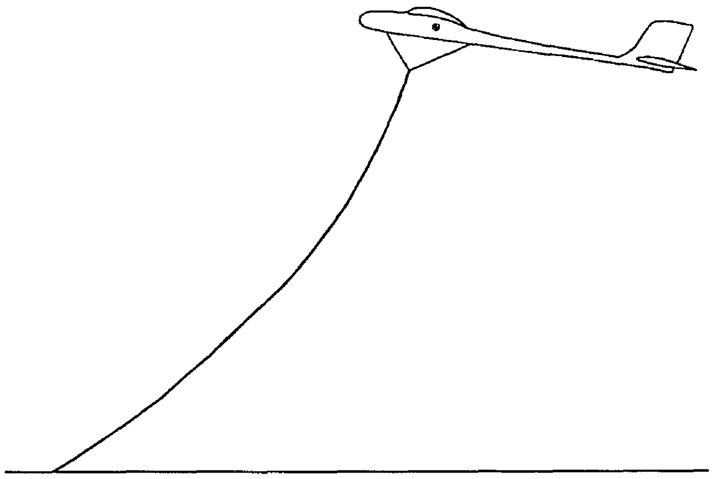


Figure 1. Schematic diagram of a THARWP

The feasibility assessment of the THARWP is based on the performance and stability results calculated using an analytical model developed specifically for the THARWP. In designing any kind of tethered vehicle, it is difficult to determine the combined operational performance and stability of the tether and the flight vehicle. As the number of applications for tethered vehicles has increased over the years, so has the interest in developing mathematical models that accurately describe their dynamic behaviour. As a result, many different analytical techniques have been developed to predict the performance and stability characteristics of towed and tethered vehicles, both lifting and non-lifting.

Initial tethered vehicle research investigated the stability of kites and towed gliders (Ref. 1). The longitudinal and lateral stability equations developed through this research utilized the tether to calculate the instantaneous equilibrium position of the kite while ignoring any oscillations of the tether. While this may be a valid assumption for kites with a short tether, the stability of a high-altitude kite was sure to be influenced by the dynamic motion of the tether. Subsequent research investigated the stability of non-lifting bodies towed beneath aircraft (Ref. 2). The dynamic behaviour of a towed body was similar to that of the THARWP since the gravitational force acting on the mass of a non-lifting body works in a similar manner as the lift force acting on a tethered flight vehicle. However, the stability analysis for the towed body was developed to characterize lateral motion only (Ref. 2). The two underlying assumptions in this research were: (a) the motion of the tether was defined as a rigid-body rotation about the attachment point; and (b) the effects of the aerodynamic and inertial tether forces were approximated through modified external forces acting on the body attached to the tether. These modified external forces appeared in the equations of motion as "cable stability derivatives." While not calculating the actual motions of the tether, this method did attempt to account for the interactions between the tether and the towed body.

Approximately a decade past, a significantly different method was developed to describe the dynamics of tethered body systems (Refs. 3, 4). In this research, equations of motion were formulated for the motion of the tether, while the end and auxiliary conditions for the tether equations were provided by the aerostat attached to the tether. However, the tether was still assumed to rotate in a rigid-body manner about the attachment point. Therefore, this simple tether model did not account for other oscillations that could occur within the tether and which could have an impact on the motion of the aerostat. Despite this shortcoming, this analysis was successful in describing the motion of low-altitude tethered aerostats.

Addressing the need for a more advanced tether model, an analysis technique was developed in Ref. 5 that combined a discrete-element formulation of the tether equations of motion with aerostat linearized equations of motion derived from traditional aircraft equations of motion (Refs. 6, 7). In this discrete-element formulation, a continuous tether was divided into N discrete elements and, through the use of the Lagrange's equations, resulted in a system of second-order, linear differential equations of motion. The longitudinal and lateral components of these equations were independent of one another, as was the case with the aerostat equations of motion. Therefore, two complete and independent sets of equations could be formulated to describe the longitudinal and lateral motions of the tethered aerostat. This method was advantageous in that the discrete-element tether model allowed for the characterization of oscillation frequency and amplitude within the tether. The results of the discrete-element analysis were validated through their comparison to experimental results obtained from wind tunnel tests. However, because the experimental results were obtained solely for the lateral motions, only the lateral equations of motion were validated.

The latter two analysis techniques provided the starting point from which the THARWP stability analysis was developed. This analysis, formulated using the discrete-element tether model, was developed to assess the feasibility of, and to optimize, the THARWP. However, as noted above, the validity of the longitudinal equations of motion for the discrete-element tether (Ref. 5) was not demonstrated. Thus, to validate the THARWP implementation of the longitudinal discrete-element analysis, an additional THARWP stability analysis was formulated using the simple tether model (Refs. 3, 4).

Prior to the development of the stability analysis, the feasibility and performance capabilities of the THARWP were determined. These were ascertained through the development of an equilibrium analysis (the “trim-state analysis”), which evaluated the steady-state balance of aerodynamic and gravitational forces acting on both the tether and the flight vehicle. The first goal of the trim-state analysis was to determine whether or not the THARWP concept was feasible. The initial design of the flight vehicle was based on that of a high-performance glider. High-performance gliders possess characteristics thought to be the most desirable for the flight vehicle, specifically high lift and low drag. Following the feasibility study, the trim-state analysis was used to design a preliminary THARWP with optimal trim-state performance. The goals of this preliminary design were twofold:

1. to maximize the payload and altitude capabilities of the THARWP; and
2. to establish the “baseline” THARWP configuration for use in the validation of the discrete-element stability analysis.

THARWP configuration specified, the validity of the longitudinal discrete-element stability analysis was tested. Recall that the validity of the lateral equations of motion were previously confirmed (Ref. 5). The stability characteristics of the baseline THARWP were calculated using both of the THARWP stability analyses developed: the first was based on the simple tether model (Refs. 3, 4), and the second was based on the discrete-element tether model (Ref. 5). Once these results were compared, and the validity of the discrete-element stability analysis confirmed, the THARWP design was optimized. The optimization of the THARWP entailed modifying various flight-vehicle parameters and assessing their impact on both the THARWP's equilibrium state and dynamic stability. This iterative process continued until an optimum THARWP design was achieved. Finally, through knowledge gained from the dynamic stability analysis, the necessity for additional control augmentation was assessed and possible methods for its implementation were discussed.

The THARWP, similar to a high-altitude tethered aerostat, is a complex dynamic system. The dynamic motion of the tether affects and will be affected by the dynamic motion of the flight vehicle. As in most attempts to model complex dynamic systems, some simplifications are required to formulate solvable equations and still adequately describe the behaviour of the THARWP. However, the risk involved in making these simplifications is that the real-world validity of the results may be compromised. Therefore, all simplifications were carefully considered to determine their impact on the validity of the solutions.

The analysis of the THARWP includes two distinct components: the equilibrium (or static trim-state) analysis and the dynamic stability analysis. Both of these analyses utilize a process by which the tether is divided into equal-length elements (Refs. 3, 4, 5). This "discretization" of the tether facilitates the use of analysis techniques developed specifically to model discrete-element systems. The implementation of these techniques, and the two components of the THARWP analysis, will be discussed in the following sections.

A. THARWP Trim-State Analysis

The THARWP trim-state analysis was developed to assess both the feasibility and the optimum performance of the THARWP. In addition, the trim-state position of the flight vehicle and of each tether element was required to satisfy the initial conditions of the dynamic stability analysis. With the flight vehicle parameters fixed and the atmospheric conditions determined based on an initial estimate of the flight vehicle altitude, the trim-state analysis calculated the position of the tether and the flight vehicle. As stated previously, the key to the THARWP trim-state analysis was the discretization of the tether. The mechanics of the continuous tether were modeled using N straight elements joined together with frictionless pin joints. A schematic diagram of the discretized tether and the forces acting on an individual tether element are shown in Figure 2. The aerodynamic and gravitational forces for each tether element were concentrated at the mid-point of each element. If each tether element was at equilibrium then these forces, when combined with the tension forces acting at each end of the tether element, will balance. In addition, the moments resulting from these forces will also

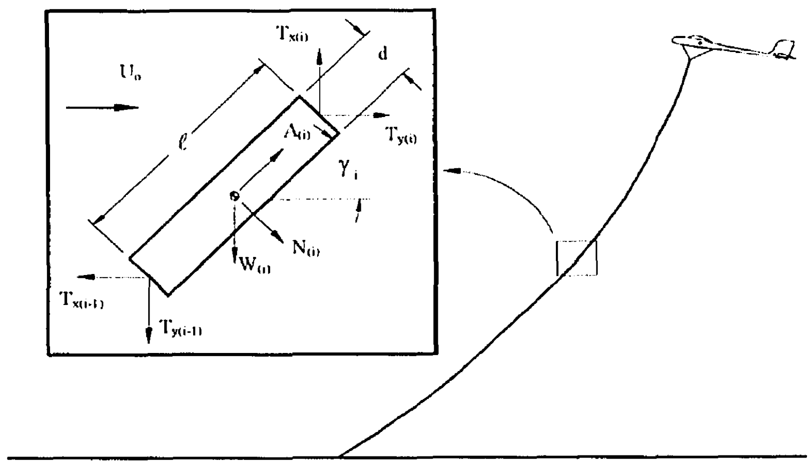


Figure 2. Tether model used for the trim-state analysis

balance. Referring to Figure 2, the balance of forces and moments for the ith tether element are given by the following equations:

$$\sum F_X = 0 = T_{X_i} + A_i \cos\gamma_i + N_i \sin\gamma_i - T_{X_{i-1}} \tag{1a}$$

$$\sum F_Y = 0 = T_{Y_i} + A_i \sin\gamma_i - N_i \cos\gamma_i - W_i - T_{X_{i-1}} \tag{1b}$$

$$\sum M_{cm} = 0 = T_{X_i}\frac{\ell}{2}\sin\gamma_i - T_{Y_i}\frac{\ell}{2}\cos\gamma_i + T_{X_{i-1}}\frac{\ell}{2}\sin\gamma_i - T_{Y_{i-1}}\frac{\ell}{2}\cos\gamma_i \tag{1c}$$

where: A_i is the axial drag force acting on the ith element due to skin friction,

$A_i = C_{d(\text{skin fric})} q_i d_i \ell \cos^2\gamma_i$, and

N_i is the cross-flow drag force acting on the ith element,

$N_i = C_{d(\text{cyl})} q_i d_i \ell \sin^2\gamma_i$ for i = N ... 1

Upon inspection, the three Eqs. (1a-c) appear to be under-determined. However, at the confluence point (i = N), the tension can be determined by calculating the equilibrium attitude of the flight vehicle and the associated drag and lift forces. The drag of the flight vehicle is the horizontal component of the confluence point tension, T_{X_N}, and the lift, in excess of the flight vehicle's weight, is the vertical component of the confluence point tension, T_{Y_N}. Through algebraic manipulation, Eqs. (1a-c) can be converted into the following expressions:

$$2T_{X_i}\sin\gamma_i - 2T_{Y_i}\cos\gamma_i + C_{d(cyl)}q_i d_i \ell \sin^2\gamma_i + W_i\cos\gamma_i = 0 \tag{2a}$$

$$T_{X_{i-1}} = T_{X_i} + A_i\cos\gamma_i + N_i\sin\gamma_i \tag{2b}$$

$$T_{Y_{i-1}} = T_{Y_i} + A_i\sin\gamma_i - N_i\cos\gamma_i - W_i \tag{2c}$$

By using Eq. (2a), which is written in terms of T_{X_i}, T_{Y_i}, and γ_i, the angle of the N^th tether element, γ_N, can be calculated. Subsequently, the tension forces acting on the upper end of the N-1 tether element, $T_{X_{N-1}}$ and $T_{Y_{N-1}}$, can be calculated using Eqs. (2b) and (c). This procedure is then repeated for each element down to the attachment point.

As stated above, in order for this iterative process to begin, the confluence point tension and, thus, the drag and excess lift generated by the flight vehicle must be known. To calculate these forces, all the moments acting on the flight vehicle, taken about the confluence point, are summed together. When the flight vehicle is at equilibrium, the summation of moments taken about the confluence point is equal to zero. The summation of moments is as follows:

$$\sum M_{cp} = 0 = M_{fuse} + M_{ac} + L_{ac}\left(x_{cp} - x_{ac}\right) + L_t\left(x_{cp} - x_t\right) + D_{fuse}\left(z_{cg} - z_{cp}\right)$$
$$+ D_{ac}\left(z_{ac} - z_{cp}\right) + W\left(x_{cg} - x_{cp}\right) \tag{3}$$

where: M_{fuse} and M_{ac} are the fuselage and wing moments;
L_{ac} and L_t are the wing and horizontal stabilizer lifts;
D_{fuse} and D_{ac} are the fuselage and wing drags;
W is the total weight of the flight vehicle; and
x and z are the horizontal and vertical distances, respectively, from the most forward point on the fuselage center line to the location denoted by the subscripts.

The moment, lift, and drag terms in Eq. (3) are then expanded using traditional aerodynamic theory (Refs. 6, 7, 8, 9, 10). Following the expansion of the moment, lift, and drag terms, The solution of the quadratic equation results in two values for α_{fv}, both of which are compared to the allowable operating range of the flight vehicle. The solution within the allowable operating range of the flight vehicle is the equilibrium AOA of the flight vehicle, α_{eq}. Note that each equilibrium AOA calculated is valid only for the altitude for which it was initially specified. Using α_{eq}, the lift and drag produced by the flight vehicle can be calculated. Then the tension in the tether at the confluence point is the result of the vectoral addition of the lift force in excess of the flight vehicle's weight and the drag force.

Following the calculation of the tether tension at the confluence point, the tension and the angle of each tether element is calculated sequentially by applying Eqs. (2a-c) to each tether element starting from the confluence point down to the ground. By stepping down the tether one element at a time, the tether element intersecting the ground is identified as the one that is oriented horizontally with respect to the ground. The results from each iteration through the N tether elements are the total length of the tether and the actual altitude of the flight vehicle (the "calculated altitude"). The calculated altitude is then compared to the altitude used to calculate the drag and the lift of the flight vehicle (the "specified altitude"). The comparison is conducted after each iteration through the N tether elements and is used to modify the specified altitude for the next iteration. This process continues until the difference between the calculated altitude and the specified altitude is less ten percent of the length of one tether element (this allowable error can be modified as desired). The steps of the trim-state analysis are as follows:

1. specify the aerodynamic parameters of the tether and flight vehicle;
2. identify the variation of the atmospheric conditions with changes in altitude;
3. calculate all the terms that remain constant for the entire trim-state analysis;
4. choose an initial specified altitude to begin the trim-state analysis;
5. using the specified altitude, calculate the equilibrium AOA of the flight vehicle [Eq. (3)];

at the confluence point;

7. calculate the tension and angle for each tether element, starting from the confluence point and working downwards;
8. determine the actual tether length and the calculated altitude;
9. evaluate the difference between the specified and calculated altitudes and exit the iteration loop if the desired accuracy has been reached;
10. modify the specified altitude based on the value of the 'calculated' altitude; and
11. go to step 5.

The trim-state analysis was implemented in a computer program called "Trimstat" (see Appendix 14) to facilitate the rapid evaluation of several different tether and flight vehicle configurations. The program was written in the computer programming language Borland C++ 4.52 and was compiled for the Windows95 operating system environment. In addition to allowing rapid modifications of the tether and the flight vehicle parameters, Trimstat also provides two methods for defining the relationship between the wind speed and the altitude. The first method allows the use of up to a 9th order polynomial function to define the wind speed versus altitude relationship. The second method allows tabular data for the wind speeds versus altitude to be read from a regular text file.

The wind speed versus altitude relationship, used throughout the THARWP's trim-state and dynamic stability analyses, was based on statistical data received from the Atmospheric Profiler Research Facility in New Mexico, United States (Ref. 11). The statistical data was comprised of hourly average wind speeds for altitudes between 7.6 and 18500.0 meters. Data for the days between May 10, 1995 and May 20, 1995 were averaged and approximated with a 5th order polynomial curve to give the wind profile shown in Figure 3. Although specific to the New Mexico area in late spring, this wind speed profile provided a good approximation for the typical THARWP operating conditions.

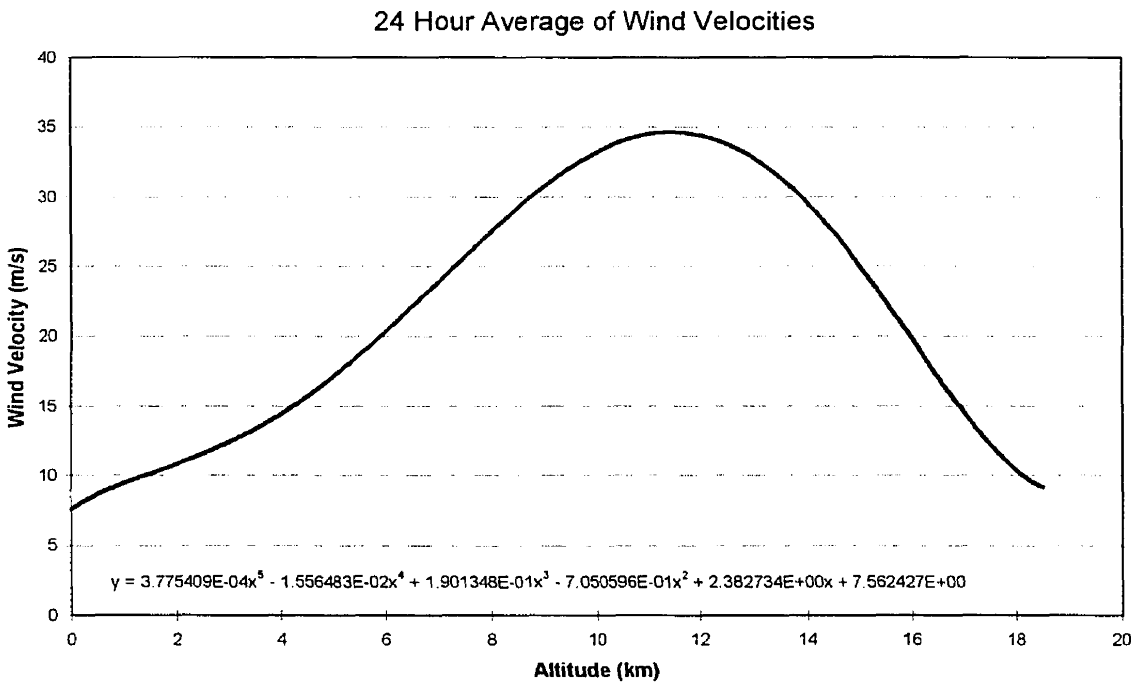


Figure 3. Statistical wind speed *vs.* altitude data

With the static trim-state analysis implemented in Trimstat, the evaluation of various flight vehicle configurations was performed. The initial flight vehicle parameters were set to approximate those for a high-lift glider-type flight vehicle since the initial assumption was that a high-lift, low-drag design would allow the THARWP to achieve the greatest operating altitudes. The wingspan of 15 meters, the wing chord of 1.5 meters, and the flight vehicle mass of 70 kg would remain constant throughout the evaluation to provide a basis for comparison between the different results. The initial tether parameters were set to match those of a 2.5 mm diameter rope made from Kevlar-49 fibers (Ref. 12). The motivation for this initial tether material choice was to allow the flight vehicle to attain greater altitudes by minimizing the tether mass.

Starting with the initial THARWP configuration, the trim-state analysis was performed many times while the THARWP parameters were modified. The goals of this process were to prove the feasibility of the THARWP concept and to improve the trim-state performance of the attainable altitude for a given flight vehicle and payload mass. One key factor in maximizing the flight vehicle altitude was to minimize the trim-state drag of the flight vehicle. This was done by modifying the position of the flight vehicle's CG and the other flight vehicle components (wing, horizontal stabilizer, and fuselage) relative to the confluence point such that the trim-state AOA of the flight vehicle was zero degrees. At this attitude, the drag of the fuselage and horizontal stabilizer were minimized. Using this approach as well as modifying many of the other THARWP parameters, the THARWP configuration that achieved the highest trim-state altitude was found. See Appendix 1 for the specifications for this baseline THARWP.

One of the parameters modified during the THARWP optimization was the tether material. The material was changed from woven Kevlar-49 fibers to spring-tempered, 302 stainless steel wire (Ref. 13). Because the yield strength of this type of steel was much higher than that of the Kevlar-49 (see Table 1 for material properties), the diameter of the steel tether could be reduced to 1 mm. The reduction in the tether diameter lowered the drag forces acting on the tether. Even though the mass-per-unit length of the steel tether was higher than that of the Kevlar-49 tether, the reduction in the tether drag resulted in the increase of the flight vehicle altitude. The flight vehicle altitude and the total tether length corresponding to both tether materials are shown in Table 2.

Table 1. Tether Material Properties

	Density $(\mathrm{kg/m^3})$	Tensile Yield Strength (MPa)	Elastic Modulus (GPa)
Kevlar-49	1,010	370[1]	10
302 SS	7,860	2,450	200

[1] The yield strength of Kevlar-49 was estimated as 67 percent of its ultimate tensile strength for design purposes.

Table 2. THARWP Performance vs. Tether Material

	Flight Vehicle Altitude (m)	Total Tether Length (m)
Kevlar-49 rope	14,370	28,500
302 SS wire	15,175	22,800

The results of the static trim-state analysis are shown graphically in Figures 4 and 5. The tether profile for the baseline THARWP, shown in Figure 4, was calculated using the statistical wind data from Ref. 11. The plot in Figure 5 was generated by varying the flight vehicle payload mass for several different wind speeds. Figure 5 illustrates the effect of payload mass and wind speed on the maximum attainable altitude.

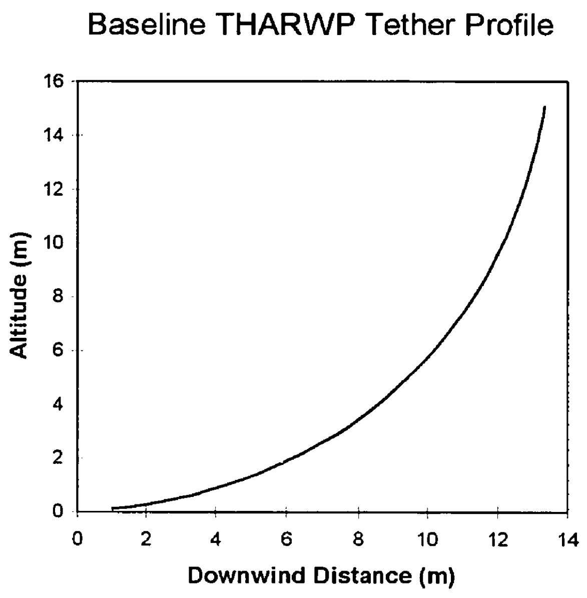


Figure 4. The tether profile for the baseline THARWP design

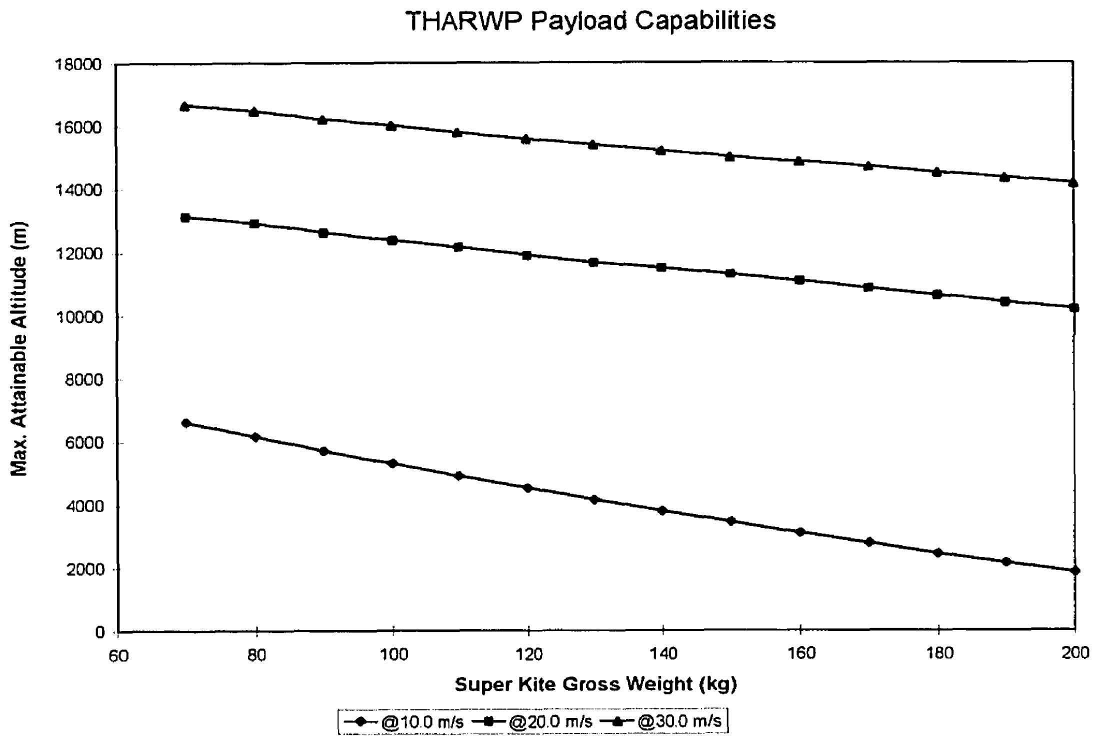


Figure 5. Performance plot for the baseline THARWP design

B. THARWP Equations of Motion: Simplified Tether Model

With the completion of the THARWP trim-state analysis and the specification of a baseline THARWP configuration, the development of the THARWP equations of motion could begin. There have been several methods developed in the past to formulate and solve the equations of motion for tethered systems. These methods have been developed for devices towed behind aircraft and boats, and for aerostats tethered to the ground. Since the THARWP was quite similar to a tethered aerostat, the methods developed for tethered aerostats were adapted for use in the analysis of the THARWP.

There were two distinct approaches used to develop the equations of motion for the THARWP. The first approach modeled the effect of attaching a tether to the flight vehicle while using three models for the tether itself: the "simple" tether model, the "string" tether model, and the modified string tether model. These variations were adapted from methods developed previously for the dynamic analysis of tethered aerostats (Ref. 3, 4). The application and comparison of these approaches will be discussed in the following sections. The second approach, which will be discussed in Section II.C, modeled the entire tether using a discrete-element technique. This latter approach allowed a more accurate evaluation of both the dynamic motions of the tether and their interaction with the dynamic motions of the flight vehicle.

1. Linearized Flight Vehicle Equations of Motion with a Simple Tether (no dynamic tether effects)

The development of the first dynamic analysis of the THARWP was based on a simple tether model developed to analyze the basic motions of a tethered aerostat (Ref. 3). To describe the motions of the flight vehicle, this analysis used the linearized general equations of unsteady motion for an airplane in steady state flight (Refs. 6, 7). The dynamic motions of the tether were not calculated using this analysis. However, the additional external forces acting on the flight vehicle due to the tether attachment were included and the motion of the confluence point was calculated.

A schematic view of the simple tether model and the orientation of the coordinate systems used in the analysis are shown in Figure 6. Traditional stability axes were used for describing the flight vehicle dynamics, which facilitated the use of traditional non-dimensional stability derivatives. This allows different flight vehicle configurations to be compared in the same way that traditional aircraft are compared. The full development of this simple tether model is shown in Appendix 2. This development produced two systems of equations which describe the longitudinal and lateral motions of the flight vehicle's CG and the confluence point. The separation of the longitudinal and the lateral motions was achieved through linearization of the equations (*i.e.* retaining only first-order effects). The longitudinal motion of the THARWP was defined in terms of the flight vehicle pitch angle, θ, and the motion of the confluence point, $\zeta(L_n,t)$ (see Figure 6). The lateral motion of the THARWP was defined in terms of the flight vehicle roll and yaw angles, and the lateral motion of the confluence point, $\tilde{Y}(L_n,t)$.

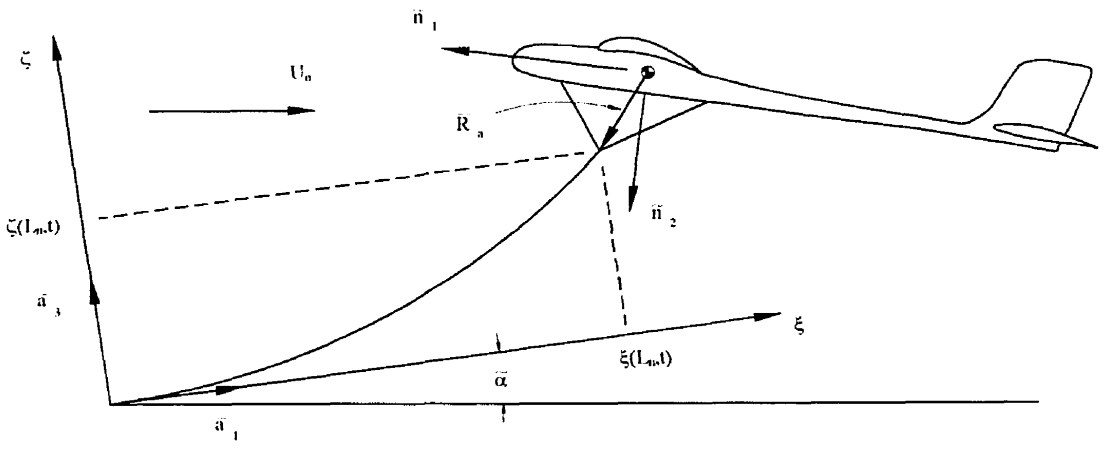


Figure 6. Simple tether model

2. <u>Linearized Flight Vehicle Equations of Motion with a String Tether (less the effects of drag and inertia)</u>

The theoretical model of the THARWP was modified in an attempt to account for simple motions of the tether. The tether was modeled as a rigid link not affected by inertial or aerodynamic forces. Figure 7 shows a schematic view of the rigid link tether model and the associated variables used. Note that these equations, as well as the equations for the third tether model variation, were only developed for the longitudinal motion. Only the longitudinal equations were developed since only the longitudinal equations for the discrete-element THARWP analysis required validation. The rigid-link tether was perfectly straight and assumed to rotate about the attachment point. The angle between the THARWP tether and the ground was given by $\tilde{\alpha} = \tilde{\alpha}_o + \Delta\tilde{\alpha}$ (where $\tilde{\alpha}_o$ was the equilibrium angle and $\Delta\tilde{\alpha}$ was the small perturbation angle from the equilibrium position). Now, by writing a similar expression for the tether tension, $T = T_o + \Delta T$, the forces acting on the flight vehicle through the confluence point were determined. Then the complete longitudinal equations of motion for the THARWP were formulated in terms of $\Delta\alpha$, θ, and ΔT. The complete development of the THARWP equations of motion using the string tether model is given in Appendix 3.

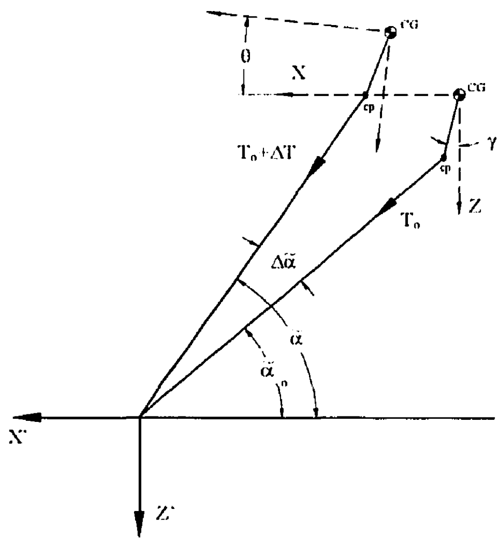


Figure 7. Rigid "String" tether model

3. <u>Linearized Flight Vehicle Equations of Motion with a Modified String Tether (including the effects of drag and inertia)</u>

The simple tether model for the THARWP was modified once more to include the inertial and aerodynamic forces acting on the tether. In addition, the tether profile was assumed to have a general curvature as a result of these aerodynamic forces. The tether profile and the associated variables used in this analysis are shown in Figure 8. As with the string tether model, the angle of the tether at the attachment point and the tether tension at the confluence point were explicitly defined. The additional tether forces were integrated over the length of the tether. The results of this integration were used to calculate the additional forces acting on the confluence point, which were then combined with the equations developed for the string tether. The final differential equations of motion, along with their development, are presented in Appendix 4.

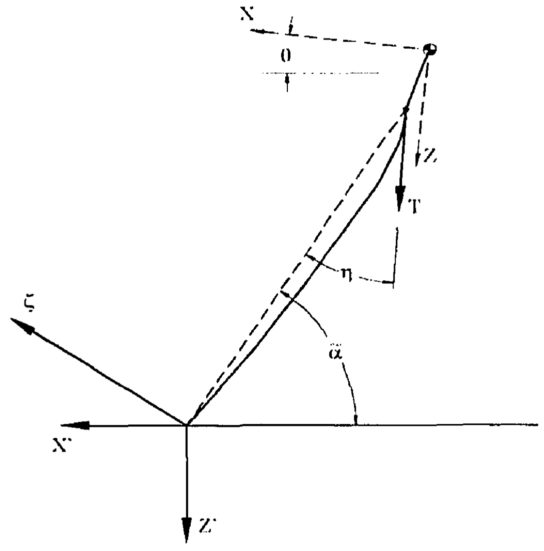


Figure 8. Modified Rigid "String" tether model

4. Longitudinal Results Compared

The three techniques developed above describe the motion of the THARWP by using three different tether models combined with the linearized equations of motion of the flight vehicle. The three tether models define the tether in the following ways:

1. a flight vehicle constraint which defines the position of both the confluence point and the flight vehicle's CG relative to the attachment point;
2. a rigid "string" element that rotates dynamically about the attachment point but is not affected by aerodynamic or inertial forces; and
3. a curved, rigid "string" element that rotates dynamically about the attachment point and which includes an approximation for the aerodynamic and inertial forces acting on the tether.

These different techniques were fully developed in Appendices 2, 3, and 4, and were implemented in the computer programs jdd_meth.cpp, jdd_strg.cpp, and jdd_mass.cpp, listed in Appendices 11, 12, and 13, respectively. These programs calculate the longitudinal stability were based on the perturbations of the flight vehicle position in the X and Z directions and on the perturbations of the flight vehicle pitch angle, θ, from the equilibrium pitch angle, θ_o. This data was then used to generate root-locus plots for each of the three longitudinal control gains.

The primary purpose for developing these three preliminary THARWP analyses was to generate a database of analytical results which could then be compared to the results from the discrete-element THARWP analysis. The discrete-element THARWP analysis, which will be developed in Section II.C, was previously used to analyze the stability characteristics of tethered aerostats (Ref. 5). The research conducted in Ref. 5 was primarily concerned with the lateral stability of the aerostat. As a result, only the lateral analysis was verified with experimental results. However, since the lateral and longitudinal equations of motion in this research were developed using the identical methodology, one could argue that the validation of the lateral equations implies the validity of the longitudinal equations. While the above assumption was considered correct, the additional comparison with the results from the three simple tether models was also performed.

For a valid comparison of the three simple tether models, the baseline THARWP configuration (see Appendix 1), defined using the trim state analysis, was used for all three analyses. For each of the three analyses, the X and Z position control gains were varied from -50 to 50 N/m using a step size of 5 N/m, and the pitch angle control gain was varied from -500 to 500 $\text{N}\cdot\text{m/rad}$ using a step size of 50 $\text{N}\cdot\text{m/rad}$. Each of these control gains was varied separately to assess their impact on the stability of the THARWP. A larger variation was used for the pitch angle control gain since the units of the pitch angle control gain were very small (*e.g.* the control gain of 500 $\text{N}\cdot\text{m/rad}$ was equivalent to 8.73 $\text{N}\cdot\text{m/deg}$).

The root-locus plots resulting from the above control gain variations for each of the three tether model variations are presented in Appendix 5. By observing these root-locus plots, the agreement among the results from the string and modified string tether models is evident except for small discrepancies between the various stability roots. The results from two tether models. These discrepancies can be attributed to the differences in the dynamic systems that are mathematically defined by the three tether models. Therefore, the results from the modified string tether model will be used to validate the discrete-element THARWP analysis.

The root-locus plots show three distinct stability roots except for larger variations of the X control gain. Two of the stability roots are aperiodic (one highly damped and the other lightly damped) and the third is oscillatory (lightly damped). The oscillatory stability root has a frequency ranging from 0.6 rad/s to 1.1 rad/s depending on the tether model considered. For comparison, the frequency of the inverted pendulum motion (Ref. 14), which is characteristic of tethered and towed systems, can be calculated using the following equation:

$$\omega_p = \left[\frac{L - mg}{m\ell}\right]^{\frac{1}{2}}$$

where: L is the total lift produced by the flight vehicle;
m is the mass of the flight vehicle; and
ℓ is the total length of the tether.

Note: The theory from which this calculation of the pendulum frequency was derived is only valid for low wind velocities.

Using the baseline THARWP design ($m = 100\text{ kg}$ and $\ell = 18000\text{ m}$), the resulting pendulum frequency is $\omega_p = 0.0279$ rad/s. When compared to the expected pendulum frequency, the oscillatory stability root has a much higher frequency (more characteristic of a 20 meter long tether). Therefore, the oscillatory stability root likely characterizes the interaction between the oscillations of the confluence point and the pitching oscillations of the flight vehicle about the confluence point. The lightly damped, aperiodic stability root describes the pendulum type motion of the THARWP and the highly damped, aperiodic stability root characterizes the pitching motion of the flight vehicle about the confluence point.

C. THARWP Equations of Motion: Discrete-Element Tether Model

The three formulations of the THARWP equations of motion developed above lacked the ability to accurately describe the tether's dynamic motion. For very long tethers (greater than 1000 m), the dynamic motions of the tether can significantly impact the dynamic motions of the flight vehicle attached to the end of the tether. For this reason a different approach was taken to developing the equations of motion for the THARWP. The continuous tether was replaced by N discrete tether elements, which facilitated the calculation of each element's dynamic motion as well as the interaction between the dynamic motions of the tether and the flight vehicle. This discrete-element technique (Ref. 5) was used to develop the equations of motion for the THARWP. These equations were implemented in Trimstat in order to readily evaluate the stability roots for many different THARWP configurations. In addition, the equations of motion were integrated in a time-step analysis to aid the evaluation of the overall performance of the THARWP.

1. Tether Equations of Motion

The discrete-element technique is the method by which a continuous system is broken into many small elements. These elements are then connected together using assumptions that will approximate the continuous system. As the number of elements becomes large, the behaviour of the discrete-element system will closely model the behaviour of the continuous system. In this manner, the continuous tether of the THARWP was divided into N elements of equal length, $\bar{\ell}$. The tether mass was approximated by positioning concentrated masses at the ends of each element. The elastic modulus, E_c, and the damping, η_c, properties were constant for all of the tether elements. Each tether element was assumed to remain straight but was allowed to pivot freely about their ends as if they were pin-jointed together. Figure 9 shows a schematic diagram of the discrete-element tether model. Despite the obvious physical differences between the continuous tether and the discrete-element tether, the continuous tether was adequately modeled by the discrete-element tether as the number of tether elements was increased (Ref. 5).

The entire tether was made of the same material and, as a result, the tether density, ρ_c, was constant for all tether elements. However, assuming a circular cross-section, the diameter

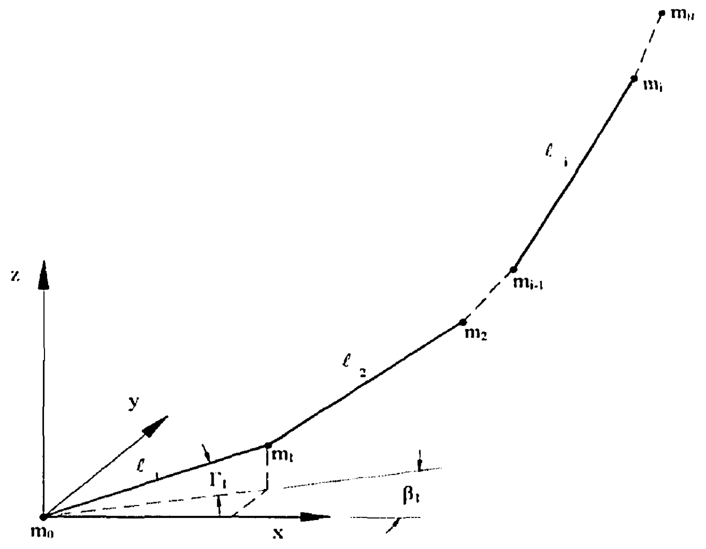


Figure 9. Discrete-element tether model

was allowed to vary from the ground to the confluence point. This feature was included in the tether model to investigate whether the tether diameter could be optimized to increase the efficiency of the THARWP. The tether diameter, denoted by d_i for i=0...N, was assumed to vary linearly from the attachment point to the confluence point. The lumped masses were then distributed by concentrating half of the mass of each tether element at either end of that element. The resulting lumped masses are given by:

$$m_i = \frac{\pi\rho_c\ell}{4}d_i^2 \quad \text{for } i = 1 \ldots \text{N-1} \tag{1a}$$

$$m_N = \frac{\pi\rho_c\ell}{8}d_N^2 \tag{1b}$$

noting that m_0 is rigidly fixed at the attachment point and, therefore, is not a factor in the stability analysis. Both the stiffness and damping within each tether element are modeled using an ideal spring and an ideal dashpot in parallel as shown in Figure 10. The stiffness of each element is given by:

$$k_i = \frac{E_c A_i}{\ell} = \frac{\pi E_c}{8\ell}\left(d_i^2 + d_{i-1}^2\right) \tag{2}$$

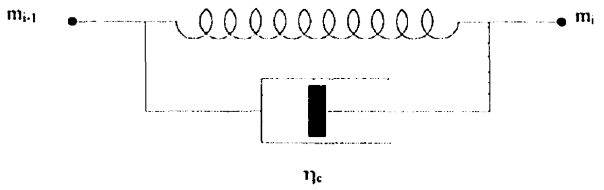


Figure 10. Physical model for a discrete tether element

With the basic tether model defined above, the Lagrange's equations (Ref. 14) were used to formulate the second-order equations of motion for the discrete-element tether model. Of the many forms that the Lagrange's equations can take, the matrix form of the Lagrange's equations, shown below, was used to develop the THARWP tether analysis.

$$\frac{d}{dt}\left(\frac{\partial T}{\partial \dot{q}_i}\right) - \frac{\partial T}{\partial q_i} + \frac{\partial V}{\partial q_i} = Q_{q_i} \qquad \text{for } i = 1 \ldots N \tag{3}$$

where: T is the kinetic energy of the tether system;

V is the potential energy of the tether system;

q_i is the set of generalized coordinates that uniquely describe the position of the ith tether element; and

Q_{q_i} is the generalized forces associated with the ith generalized coordinate, also given by:

$$Q_{q_i} = \sum_{j=1}^{N}\left(F_{xj}\frac{\partial x_j}{\partial q_i} + F_{yj}\frac{\partial y_j}{\partial q_i} + F_{zj}\frac{\partial z_j}{\partial q_i}\right) \tag{4}$$

The forces, F_{xj}, F_{yj}, and F_{zj} in the expression for Q_{q_i} are the components of the total force acting on the ith lumped tether mass. These forces, expressed in terms of the earth-fixed axes, (x, y, and z), excluded forces derivable from a potential energy function, V. The next step in the development of the discrete-element tether stability analysis was to develop the kinetic and potential energy expressions for the discrete-element tether.

[illegible]

In order to develop the kinetic energy expression for the discrete-element tether, the position and velocity of all N lumped masses had to be determined with respect to the attachment point. The position of the lumped masses are uniquely defined by the generalized coordinates ℓ_i, Γ_i, and β_i (see Figure 9). The exact expression for the position of the kth lumped mass relative to the attachment point is given by the following:

$$x_k = \sum_{i=1}^{k} \ell_i \cos\Gamma_i \cos\beta_i \tag{5a}$$

$$y_k = \sum_{i=1}^{k} \ell_i \cos\Gamma_i \sin\beta_i \tag{5b}$$

$$z_k = \sum_{i=1}^{k} \ell_i \sin\Gamma_i \tag{5c}$$

and the velocity of the kth lumped mass are given by the time derivative of Eqs. (5a-c):

$$\dot{x}_k = \sum_{i=1}^{k}\left(\dot{\ell}_i \cos\Gamma_i \cos\beta_i - \dot{\Gamma}_i\ell_i \sin\Gamma_i \cos\beta_i - \dot{\beta}_i\ell_i \cos\Gamma_i \sin\beta_i\right) \tag{6a}$$

$$\dot{y}_k = \sum_{i=1}^{k}\left(\dot{\ell}_i \cos\Gamma_i \sin\beta_i - \dot{\Gamma}_i\ell_i \sin\Gamma_i \sin\beta_i + \dot{\beta}_i\ell_i \cos\Gamma_i \cos\beta_i\right) \tag{6b}$$

$$\dot{z}_k = \sum_{i=1}^{k}\left(\dot{\ell}_i \sin\Gamma_i + \dot{\Gamma}_i\ell_i \cos\Gamma_i\right) \tag{6c}$$

The expression for the total kinetic energy is the summation of the kinetic energies of all N lumped masses and is written as follows:

$$T = \tfrac{1}{2}\sum_{k=1}^{N} m_k\left(\dot{x}_k^2 + \dot{y}_k^2 + \dot{z}_k^2\right) \tag{7}$$

Before the velocities defined in Eqs. (6a-c) are substituted into the kinetic energy expression, Eq. (7), all quantities in Eqs. (6a-c) are written in terms of the equilibrium and the small perturbation components. First, the generalized coordinates and their first time derivatives are written as:

$$\ell_i = \bar{\ell} + \delta\ell_i \qquad \Gamma_i = \bar{\Gamma}_i + \gamma_i \qquad \bar{\beta}_i = 0$$

where the $\overline{(\ )}$ terms are the equilibrium values. Also, due to the small perturbation assumption, the following simplifications can be made:

$$\frac{\delta\ell_i}{\ell_i}, \frac{\gamma_i}{\Gamma_i}, \beta_i, \delta\dot{\ell}_i, \dot{\gamma}_i, \dot{\beta}_i << 1 \qquad \cos\beta_i \cong 1 \qquad \sin\beta_i \cong \beta_i$$

$$\cos\Gamma_i = \cos\left(\overline{\Gamma}_i + \gamma_i\right) \cong \cos\overline{\Gamma}_i - \gamma_i \sin\overline{\Gamma}_i \equiv \overline{c}_i - \gamma_i \overline{s}_i$$

$$\sin\Gamma_i = \sin\left(\overline{\Gamma}_i + \gamma_i\right) \cong \sin\overline{\Gamma}_i + \gamma_i \cos\overline{\Gamma}_i \equiv \overline{s}_i + \gamma_i \overline{c}_i$$

Using the above assumptions and retaining only the first-order terms, Eqs. (6a-c) are written as follows:

$$\dot{x}_k = \sum_{i=1}^{k}\left(\delta\dot{\ell}_i \overline{c}_i - \dot{\gamma}_i \overline{\ell}\, \overline{s}_i\right) \tag{8a}$$

$$\dot{y}_k = \sum_{i=1}^{k} \dot{\beta}_i \overline{\ell}\, \overline{c}_i \tag{8b}$$

$$\dot{z}_k = \sum_{i=1}^{k}\left(\delta\dot{\ell}_i \overline{s}_i + \dot{\gamma}_i \overline{\ell}\, \overline{c}_i\right) \tag{8c}$$

From inspection of Eqs. (8a-c), the tether motion is seen to be comprised of independent longitudinal and lateral components. The longitudinal motion, $\overline{x}_k$ and $\overline{z}_k$, are dependent only on the small perturbations of the longitudinal generalized coordinates, $\delta\ell_i$ and γ_i. Similarly, the lateral motion, $\overline{y}_k$, is only dependent on the small perturbation of the lateral generalized coordinate, β_i. As a result, the kinetic energy can be said to be the combination of a longitudinal and a lateral component. This can be written as follows:

$$T = T_{lg} + T_{lt} = \tfrac{1}{2}\sum_{k=1}^{N} m_k\left(\dot{x}_k^2 + \dot{z}_k^2\right) + \tfrac{1}{2}\sum_{k=1}^{N} m_k \dot{y}_k^2 \tag{9}$$

and substituting Eqs. (8a-c) into Eq. (9) gives the following expression for the longitudinal and lateral kinetic energy of the tether:

$$T_{lg} = \tfrac{1}{2}\sum_{k=1}^{N} m_k \sum_{i=1}^{k}\sum_{j=1}^{k}\left[\delta\dot{\ell}_i \delta\dot{\ell}_j \overline{c}_{ij} + \dot{\gamma}_i \dot{\gamma}_j \overline{\ell}^2 \overline{c}_{ij} + \left(\delta\dot{\ell}_i \dot{\gamma}_j \overline{\ell} - \delta\dot{\ell}_j \dot{\gamma}_i \overline{\ell}\right)\overline{s}_{ij}\right] \tag{10a}$$

where: $\bar{c}_{ij} = \cos(\bar{\Gamma}_i - \bar{\Gamma}_j) = \bar{c}_i\bar{c}_j + \bar{s}_i\bar{s}_j$, and

$$\bar{s}_{ij} = \sin(\bar{\Gamma}_i - \bar{\Gamma}_j) = \bar{s}_i\bar{c}_j - \bar{s}_j\bar{c}_i$$

$$T_{lt} = \tfrac{1}{2}\sum_{k=1}^{N} m_k \sum_{i=1}^{k}\sum_{j=1}^{k}\left[\dot{\beta}_i\dot{\beta}_j\bar{\ell}^2\bar{c}_i\bar{c}_j\right] \tag{10b}$$

Through close observation of Eqs. (10a) and (10b), one can see that, for arbitrary i, j, and k, the term $m_k[\ ]_{ij}$ is present in the summation only if $k \geq \max(i, j)$. This simplification can be explained physically as follows:

- the kinetic energy of the kth element is dependent on the velocity of the kth element, and
- the velocity of the kth element is dependent only on the velocity of the elements below the kth element.

Therefore, the velocity of any element above the kth element does not contribute to the kinetic energy of the kth element. Thus, Eqs. (10a) and (10b) can be written with the mass summation inside the other two summations as follows:

$$T_{lg} = \tfrac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\left[\sum_{k=\max(i,j)}^{N} m_k\right]\left[\delta\dot{\ell}_i\delta\dot{\ell}_j\bar{c}_{ij} + \dot{\gamma}_i\dot{\gamma}_j\bar{\ell}^2\bar{c}_{ij} + \left(\delta\dot{\ell}_i\dot{\gamma}_j\bar{\ell} - \delta\dot{\ell}_j\dot{\gamma}_i\bar{\ell}\right)\bar{s}_{ij}\right] \tag{11a}$$

$$T_{lt} = \tfrac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\left[\sum_{k=\max(i,j)}^{N} m_k\right]\left[\dot{\beta}_i\dot{\beta}_j\bar{\ell}^2\bar{c}_i\bar{c}_j\right] \tag{11b}$$

To facilitate their application to the Lagrange's equations, Eqs. (11a) and (11b) are transformed from their quadratic form into a compact matrix form.

$$T_{lg} = \tfrac{1}{2}\underline{\dot{q}}_{lg}^{T}\underline{M}_{lg}\underline{\dot{q}}_{lg} \tag{12a}$$

$$T_{lt} = \tfrac{1}{2}\underline{\dot{q}}_{lt}^{T}\underline{M}_{lt}\underline{\dot{q}}_{lt} \tag{12b}$$

where: $\underline{q}_{lg}^{T} = [\delta\ell_1, \gamma_1, \delta\ell_2, \gamma_2, \cdots \delta\ell_N, \gamma_N]$, and $\underline{q}_{lt}^{T} = [\beta_1, \beta_2, \cdots \beta_N]$ are the longitudinal and lateral generalized coordinate vectors, respectively. Note that $\underline{q}_{lg}^{T}$ has 2N elements and $\underline{q}_{lt}^{T}$ has N elements. The longitudinal mass matrix, $\underline{M}_{lg}$, is a 2N x 2N symmetrical matrix given by:

$$(M_{lg})_{2i-1,2j-1} = c_{ij}\left(\sum_{k=\max(i,j)} m_k\right) = \bar{c}_{ij} I_{ij} \qquad (13a)$$

$$(M_{lg})_{2i,2j} = \bar{\ell}^2 \bar{c}_{ij} I_{ij} \qquad (13b)$$

$$(M_{lg})_{2i-1,2j} = \bar{\ell}\, \bar{s}_{ij} I_{ij} \qquad (13c)$$

$$(M_{lg})_{2i-1,2j} = -\bar{\ell}\, \bar{s}_{ij} I_{ij} \qquad (13d)$$

and the lateral mass matrix, $\underline{M}_{lt}$, is an N x N symmetrical matrix given by:

$$(M_{lt})_{i,j} = \bar{\ell}^2 \bar{c}_i \bar{c}_j I_{ij} \qquad (14)$$

b. Potential Energy Formulation

The second set of expressions required for the use of the Lagrange's equations are the potential energy expressions of the discrete-element tether. The total potential energy is composed of the gravitational and elastic potential energy components of the tether and can be written as follows:

$$V = \sum_{k=1}^{N} m_k g z_k + \tfrac{1}{2} \sum_{i=1}^{N} k_i \left(\ell_i - \bar{\ell}_i\right)$$

where: g is the acceleration due to gravity;

$z_k = \sum_{i=1}^{k} \ell_i \sin \Gamma_i$ is the altitude of the kth lumped mass; and

k_i is the stiffness of the ith tether element (defined previously).

Observation of this equation reveals the fact that the potential energy is solely dependent on the longitudinal variables, ℓ and Γ and, therefore, has no effect on the lateral motion of the tether. Now, substituting the expression for z_k into the potential energy expression gives:

$$V = \sum_{k=1}^{N} m_k g \left(\sum_{i=1}^{k} \ell_i \sin \Gamma_i \right) + \tfrac{1}{2} \sum_{i=1}^{N} k_i \left(\ell_i - \bar{\ell}_i\right) \qquad (15)$$

$\sum_{i=1}^{k} \ell_i \sin\Gamma_i$ term defines the altitude of the kth element, the term $m_k g \ell_i \sin\Gamma_i$ is present in the summation for arbitrary i and k only if k ≥ i. This allows the potential energy expression to be rewritten as follows:

$$V = g\sum_{i=1}^{N} \ell_i \sin\Gamma_i I_{ii} + \tfrac{1}{2}\sum_{i=1}^{N} k_i\left(\ell_i - \bar{\ell}_i\right) \tag{16}$$

where: $I_{ii} = \sum_{k=i}^{N} m_k$

In this form, the potential energy equation cannot be put into the compact matrix form as were the kinetic energy equations. Thus, Eq. (16) must be expanded about the equilibrium state $\underline{q}_o = \underline{0}$ using a Taylor series expansion.

$$V = V(0) + \sum_{i=1}^{2N}\left(\frac{\partial V}{\partial q_{lg_i}}\right)_o q_i + \sum_{i=1}^{2N}\sum_{j=1}^{2N}\left(\frac{\partial^2 V}{\partial q_{lg_i}\partial q_{lg_j}}\right)_o q_{lg_i} q_{lg_j} + \cdots \tag{17}$$

The first term in Eq. (17) is the potential energy of the tether at equilibrium and can be arbitrarily set to zero. The second term is also zero since $\left(\frac{\partial V}{\partial q_{lg_i}}\right)_o$ cancels out the equilibrium part of the generalized force vector, Q_{q_i}. This leaves the third term as the most significant term of the Taylor expansion, which can be expressed in compact matrix form as:

$$V_{lg} = \underline{q}_{lg}^T \underline{N} \underline{q}_{lg} \tag{18}$$

where: $\underline{N}$ is a 2N x 2N symmetric matrix given by

$$\underline{N}_{ij} = \left(\frac{\partial^2 V}{\partial q_{lg_i}\partial q_{lg_j}}\right)_o$$

As stated above, $\underline{q}_{lg}^T = [\delta\ell_1, \gamma_1, \delta\ell_2, \gamma_2, \cdots \delta\ell_N, \gamma_N]$ can also be given as $\underline{q}_{lg_{2i-1}} = \delta\ell_i$ and $\underline{q}_{lg_{2i}} = \gamma_i$. Therefore, the terms of $\underline{N}$ are given by the following:

$$\underline{N}_{2i-1,2j} = \underline{N}_{2i,2j-1} = g\,\bar{c}_i I_{ii}\delta_{ij} \tag{19b}$$

$$\underline{N}_{2i,2j} = -g\,\bar{\ell}\,\bar{s}_i I_{ii}\delta_{ij} \tag{19c}$$

where: $\delta_{ij} = 1$ for $i = j$, and

$\delta_{ij} = 0$ for $i \neq j$

c. <u>Formulation of the Generalized Forces</u>

The final expression required for the use of the Lagrange's equations is that describing the generalized forces acting on the N tether elements, Q_{q_i}. The forces contributing to Q_{q_i} are the aerodynamic forces (the drag), internal damping forces, and the cable tension at the confluence point caused by the attachment of the flight vehicle. All of these forces must be expressed in terms of the generalized coordinates to use them in the Lagrange's equations. Expressions for each of the three generalized forces will be developed and then combined together to form the expression for Q_{q_i}.

i. <u>Aerodynamic Forces</u>

In order to facilitate the development of the aerodynamic force equations the following assumptions were specified:

1. each tether element is assumed to have a uniform, circular cross-section;
2. the prevailing wind is assumed to be a uniform steady flow;
3. the Reynolds Number is sub-critical for the flow normal to the tether;
4. the axial force on all tether elements due to aerodynamic drag is assumed negligible; and
5. the distribution of the aerodynamic force over a tether element is assumed to be concentrated into a single force acting at the midpoint of the tether element.

Since the aerodynamic forces act on every tether element differently, a new coordinate system fixed to the mid-point of every tether element will be used to facilitate the definition of the aerodynamic forces in terms of the generalized coordinates. Figure 11 shows this new elemental coordinate system, which is composed of the three independent orthogonal unit vectors, $\underline{e}_1$, $\underline{e}_2$, and $\underline{e}_3$. As shown for the ith tether element, the direction of $\underline{e}_1$ is parallel to the long axis of the tether element pointing towards the i+1 element, and the direction of $\underline{e}_3$ is such that the steady-state wind vector is within the plane of $\underline{e}_1$ and $\underline{e}_3$. Now $\underline{e}_1$, $\underline{e}_2$, and $\underline{e}_3$ can be related to the coordinate system at the attachment point, $\underline{i}$, $\underline{j}$, and $\underline{k}$, through the following transformation matrix:

$$\begin{bmatrix} \underline{e}_1 \\ \underline{e}_2 \\ \underline{e}_3 \end{bmatrix} = \underline{C}_k \begin{bmatrix} \underline{i} \\ \underline{j} \\ \underline{k} \end{bmatrix} \tag{20}$$

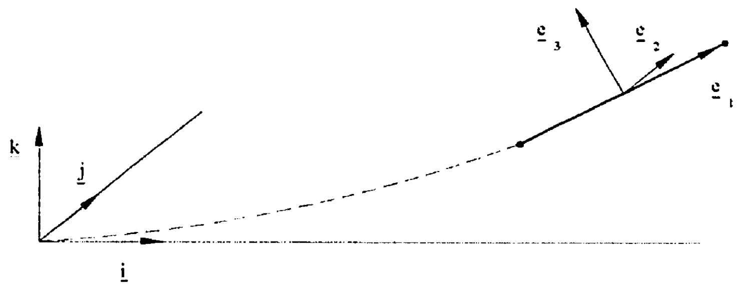


Figure 11. Tether element coordinate system

The transformation matrix, $\underline{C}_k$, is generated through the combination of single-angle transformations which describe axis rotations about $\underline{k}$ and $\underline{j}$ (Ref. 7). Following implementation of the small perturbation assumptions, the transformation matrix becomes:

$$\underline{C}_k = \begin{bmatrix} c_k & c_k\beta_k & s_k \\ -\beta_k & 1 & 0 \\ -s_k & -s_k\beta_k & c_k \end{bmatrix} \tag{21}$$

and the inverse transformation is:

$$\begin{bmatrix} \underline{i} \\ \underline{j} \\ \underline{k} \end{bmatrix} = \begin{bmatrix} c_k & -\beta_k & -s_k \\ c_k\beta_k & 1 & -s_k\beta_k \\ s_k & 0 & c_k \end{bmatrix} \begin{bmatrix} \underline{e}_1 \\ \underline{e}_2 \\ \underline{e}_3 \end{bmatrix} \tag{22}$$

The aerodynamic force acting on the kth tether element is caused by drag due to air moving around the cylindrical tether element. In order to calculate the drag of the kth tether element, its velocity must be expressed relative to the steady-state wind, U_o, and the component of this velocity normal to $\underline{e}_1$ must be calculated. This normal velocity vector, $\underline{V}_N$, will then be used to calculate the drag force acting on that tether element. First, the velocity of the kth tether element, relative to the attachment point, is written in terms of the tether element coordinates as follows:

$$\underline{w}_k = w_{k_1}\underline{e}_1 + w_{k_2}\underline{e}_2 + w_{k_3}\underline{e}_3 \tag{23}$$

Then, the velocity relative to the steady-state wind is given by

$$\underline{V} = -U_o\underline{i} + \underline{w}_k \tag{24}$$

and Eq. (22) is used to transform Eq. (24) into:

$$\underline{V} = \left(-U_o c_k + w_{k_1}\right)\underline{e}_1 + \left(U_o\beta_k + w_{k_2}\right)\underline{e}_2 + \left(U_o s_k + w_{k_3}\right)\underline{e}_3 \tag{25}$$

The normal velocity vector, $\underline{V}_N$, is defined as the velocity normal to $\underline{e}_1$ and in the plane defined by $\underline{V}$ and $\underline{e}_1$. This is written as:

$\underline{V}_N = \left(\underline{e}_1 \times V\right) \times \underline{e}_1$ and, given Eq. (25), becomes:

$$\underline{V}_N = \left(U_o\beta_k + w_{k_2}\right)\underline{e}_2 + \left(U_o s_k + w_{k_3}\right)\underline{e}_3 \tag{26}$$

In addition, the velocity of the kth tether element can be written using the attachment point coordinate system in the following manner:

$$\underline{w}_k = \tfrac{1}{2}\left[\left(\dot{x}_{k-1} + \dot{x}_k\right)\underline{i} + \left(\dot{y}_{k-1} + \dot{y}_k\right)\underline{j} + \left(\dot{z}_{k-1} + \dot{z}_k\right)\underline{k}\right] \tag{27}$$

If Eqs. (6a-c) and the coordinate transformation given in Eq. (22) are combined with Eq. (27), and the results are compared with Eq. (23), the coefficients of Eq. (23) are given by:

$$w_{k_1} = \sum_{i=1}^{k}\left(\delta\dot{\ell}_i\bar{c}_{ki} + \dot{\gamma}_i\bar{\ell}\,\bar{s}_{ki}\right)\varepsilon_{ik} \tag{28a}$$

$$w_{k_2} = \sum_{i=1}^{k}\dot{\beta}_i\bar{\ell}\,\bar{c}_i\varepsilon_{ik} \tag{28b}$$

$$w_{k_3} = \sum_{i=1} \left(-\delta\ell_i s_{ki} + \gamma_i \ell \bar{c}_{ki} \right) \varepsilon_{ik} \tag{28c}$$

where: $\varepsilon_{ik} = 1$ for $i \neq k$, and

$\varepsilon_{ik} = \frac{1}{2}$ for $i = k$

Now, considering assumptions 1 through 4 listed earlier, the aerodynamic force acting on kth tether element can be written as:

$$\underline{D}_k = -\tfrac{1}{2}\rho_k d_k C_D \ell_k \left| \underline{V}_N \right| \underline{V}_N \tag{29}$$

where: ρ_k is the air density at the altitude of the kth tether element;

d_k is the average diameter of the kth tether element; and

C_D is the cross-flow coefficient of drag for the tether material.

The aerodynamic force given in Eq. (29) is assumed to have an equilibrium component and a small perturbation component, which are given by:

$$\overline{\underline{D}}_k = -\tfrac{1}{2}\rho_k d_k C_D \bar{\ell}\, U_o^2 \bar{s}_k^2 \left(-\bar{s}_k \underline{i} + \bar{c}_k \underline{k} \right) \tag{30a}$$

$$\delta \underline{D}_k = -\tfrac{1}{2}\rho_k d_k C_D \left(\delta\ell_k \left| \overline{\underline{V}}_N \right| \overline{\underline{V}}_N + \delta\left| \underline{V}_N \right| \bar{\ell}\, \overline{\underline{V}}_N + \delta \underline{V}_N \bar{\ell} \left| \overline{\underline{V}}_N \right| \right) \tag{30b}$$

Now, Eq. (26) combined with the assumption of small perturbations give:

$$\underline{e}_2 = \overline{\underline{e}}_2 + \delta \underline{e}_2 \qquad\qquad \underline{e}_3 = \overline{\underline{e}}_3 + \delta \underline{e}_3$$

$$\left| \overline{\underline{V}}_N \right| = U_o \bar{s}_k \qquad \delta\left| \underline{V}_N \right| = U_o \bar{c}_k \gamma_k + w_{k_3} \qquad \overline{\underline{V}}_N = U_o \bar{s}_k \overline{\underline{e}}_3 \tag{31a,b,c}$$

$$\delta \underline{V}_N = \left(U_o \beta_k + w_{k_2} \right) \overline{\underline{e}}_2 + \left(U_o \gamma_k \bar{c}_k + w_{k_3} \right) \overline{\underline{e}}_3 + U_o \bar{s}_k \,\delta \underline{e}_3 \tag{31d}$$

and, using the coordinate transform in Eq. (20), $\overline{\underline{e}}_2$, $\overline{\underline{e}}_3$, and $\delta\underline{e}_3$ become:

$$\overline{\underline{e}}_2 = \underline{j} \qquad\qquad \overline{\underline{e}}_3 = -\bar{s}_k \underline{i} + \bar{c}_k \underline{k} \tag{32a,b}$$

$$\delta \underline{e}_3 = -\gamma_k \bar{c}_k \underline{i} - \bar{s}_k \beta_k \underline{j} - \bar{s}_k \gamma_k \underline{k} \tag{32c}$$

Substituting Eqs. (31a-d) and Eqs. (32a-c) into Eq. (30b) gives the following expression for the small perturbation aerodynamic force acting on the kth tether element.

$$\delta \underline{D}_k = -\left(\delta D_{k_3} \bar{s}_k + \overline{D}_{k_3} \gamma_k \bar{c}_k \right) \underline{i} + \left(\delta D_{k_2} + \overline{D}_{k_3} \bar{s}_k \beta_k \right) \underline{j}$$

$$+\left(\delta D_{k_3}c_k - \overline{D}_{k_3}s_k\gamma_k\right)\underline{k} \qquad (33)$$

where: $\delta D_{k_2} = -\frac{1}{2}\rho_k d_k C_D \overline{\ell}\, U_o \overline{s}_k\left(U_o\beta_k + w_{k_2}\right)$,

$\delta D_{k_3} = -\frac{1}{2}\rho_k d_k C_D U_o \overline{s}_k\left(\delta\ell_k U_o \overline{s}_k + 2\gamma_k \overline{\ell}\,\overline{c}_k U_o + 2w_{k_3}\overline{\ell}\right)$, and

$\overline{D}_{k_3} = -\frac{1}{2}\rho_k d_k C_D \overline{\ell}\, U_o^2 \overline{s}_k^2 = \left|\underline{\overline{D}}_{k_3}\right|$

Next, through manipulation of Eq. (33), the small perturbation aerodynamic force is substituted into the expression for the generalized force, given by Eq. (4). This substitution results in the following expressions for the aerodynamic forces acting on the tether:

$$D_{\ell_i} = \sum_{k=1}^{N}\left(D_{x_k}\frac{\partial x_k}{\partial \ell_i} + D_{y_k}\frac{\partial y_k}{\partial \ell_i} + D_{z_k}\frac{\partial z_k}{\partial \ell_i}\right) \qquad (34a)$$

$$D_{\gamma_i} = \sum_{k=1}^{N}\left(D_{x_k}\frac{\partial x_k}{\partial \gamma_i} + D_{y_k}\frac{\partial y_k}{\partial \gamma_i} + D_{z_k}\frac{\partial z_k}{\partial \gamma_i}\right) \qquad (34b)$$

$$D_{\beta_i} = \sum_{k=1}^{N}\left(D_{x_k}\frac{\partial x_k}{\partial \beta_i} + D_{y_k}\frac{\partial y_k}{\partial \beta_i} + D_{z_k}\frac{\partial z_k}{\partial \beta_i}\right) \qquad (34c)$$

where D_{x_k}, D_{y_k}, and D_{z_k} are the $\underline{i}$, $\underline{j}$, and $\underline{k}$ components of the aerodynamic force defined in Eqs. (30a) and (33). The above partial derivatives can be evaluated using Eqs. (5a-c) while noting that, since the aerodynamic forces are averaged over the length of the element, a factor of $\frac{1}{2}$ should be added for i = k. Also, since the velocity of the kth tether element is dependent only on the motion of other tether elements below, the partial derivatives for i > k are all zero. Therefore, again using the small perturbation assumption, the partial derivatives of Eqs. (34a-c) are as follows:

$$\frac{\partial x_k}{\partial \ell_i} = \left(\frac{\partial x_k}{\partial \ell_i}\right)_o + \delta\left(\frac{\partial x_k}{\partial \ell_i}\right) = \left(\overline{c}_i - \gamma_i \overline{s}_i\right)\varepsilon_{ik} \qquad (35a)$$

$$\frac{\partial y_k}{\partial \ell_i} = \delta\left(\frac{\partial y_k}{\partial \ell_i}\right) = \beta_i \overline{c}_i \varepsilon_{ik} \qquad (35b)$$

$$\frac{\partial z_k}{\partial \ell_i} = \left(\frac{\partial z_k}{\partial \ell_i}\right)_o + \delta\left(\frac{\partial z_k}{\partial \ell_i}\right) = \left(\overline{s}_i + \gamma_i \overline{c}_i\right)\varepsilon_{ik} \qquad (35c)$$

$$\frac{\quad}{\partial\beta_i} = \delta\left(\frac{\quad}{\partial\beta_i}\right) = -\beta_i \ell c_i \varepsilon_{ik} \tag{35d}$$

$$\frac{\partial y_k}{\partial\beta_i} = \left(\frac{\partial y_k}{\partial\beta_i}\right)_o + \delta\left(\frac{\partial y_k}{\partial\beta_i}\right) = \left(\bar{\ell}\,\bar{c}_i - \gamma_i\bar{\ell}\,\bar{s}_i + \delta\ell_i\bar{c}_i\right)\varepsilon_{ik} \tag{35e}$$

$$\frac{\partial z_k}{\partial\ell_i} = 0 \tag{35f}$$

$$\frac{\partial x_k}{\partial\gamma_i} = \left(\frac{\partial x_k}{\partial\gamma_i}\right)_o + \delta\left(\frac{\partial x_k}{\partial\gamma_i}\right) = \left(-\bar{\ell}\,\bar{s}_i - \gamma_i\bar{\ell}\,\bar{c}_i - \delta\ell_i\bar{s}_i\right)\varepsilon_{ik} \tag{35g}$$

$$\frac{\partial y_k}{\partial\gamma_i} = \delta\left(\frac{\partial y_k}{\partial\gamma_i}\right) = -\beta_i\bar{\ell}\,\bar{s}_i\varepsilon_{ik} \tag{35h}$$

$$\frac{\partial z_k}{\partial\gamma_i} = \left(\frac{\partial z_k}{\partial\gamma_i}\right)_o + \delta\left(\frac{\partial z_k}{\partial\gamma_i}\right) = \left(\bar{\ell}\,\bar{c}_i - \gamma_i\bar{s}_i\bar{\ell} + \delta\ell_i\bar{c}_i\right)\varepsilon_{ik} \tag{35i}$$

By substituting Eqs. (35a-i) into Eqs. (34a-c), and by subtracting out the equilibrium quantities, the small perturbation components of equations (34a-c) are:

$$\delta D_{\ell_i} = \sum_{k=1}^{N}\left[\delta\left(\frac{\partial x_k}{\partial\ell_i}\right)\bar{D}_{x_k} + \delta D_{x_k}\left(\frac{\partial x_k}{\partial\ell_i}\right)_o + \delta\left(\frac{\partial z_k}{\partial\ell_i}\right)\bar{D}_{z_k} + \delta D_{z_k}\left(\frac{\partial z_k}{\partial\ell_i}\right)_o\right]\varepsilon_{ik} \tag{36a}$$

$$\delta D_{\gamma_i} = \sum_{k=1}^{N}\left[\delta\left(\frac{\partial x_k}{\partial\gamma_i}\right)\bar{D}_{x_k} + \delta D_{x_k}\left(\frac{\partial x_k}{\partial\gamma_i}\right)_o + \delta\left(\frac{\partial z_k}{\partial\gamma_i}\right)\bar{D}_{z_k} + \delta D_{z_k}\left(\frac{\partial z_k}{\partial\gamma_i}\right)_o\right]\varepsilon_{ik} \tag{36b}$$

$$\delta D_{\beta_i} = \sum_{k=1}^{N}\left[\delta\left(\frac{\partial x_k}{\partial\beta_i}\right)\bar{D}_{x_k} + \delta D_{y_k}\left(\frac{\partial y_k}{\partial\beta_i}\right)_o\right]\varepsilon_{ik} \tag{36c}$$

Finally, Eqs. (36a-c) combined with Eqs. (35a-i), (30a), (28a-c), and (33) give the perturbation component of the generalized aerodynamic force in the form required for use in the Lagrange's equations. Note that the equilibrium quantities of the aerodynamic force are not required in the Lagrange's equations. The perturbation aerodynamic quantities resulting from the above substitutions are given as follows:

$$\delta D_{\ell_i} = -\tfrac{1}{2}\gamma_i C_D U_o^2\bar{\ell}\sum_{k=i+1}^{N}\rho_k d_k \bar{s}_k^2\bar{c}_{ik} - \tfrac{1}{2}C_D U_o^2\sum_{k=i+1}^{N}\rho_k d_k\left[\gamma_k\bar{\ell}\,\bar{s}_k\left(2\bar{c}_k\bar{s}_{ik} - \bar{s}_k\bar{c}_{ik}\right)\right.$$

$$+\bar{s}_k^2\delta\ell_k\bar{s}_{ik}\Big] - C_D U_o \ell \sum_{k=1} \sum_{j=\max(i+1,k)} \rho_j d_j \bar{s}_j \bar{s}_{ij}\left(\dot{\gamma}_k \bar{\ell}\,\bar{c}_{jk} - \delta\dot{\ell}_k \bar{s}_{jk}\right)\varepsilon_{kj} \tag{37a}$$

$$\delta D_{\gamma_i} = -\tfrac{1}{2}\delta\ell_i C_D U_o^2 \bar{\ell} \sum_{k=i}^{N} \rho_k d_k \bar{s}_k^2 \bar{c}_{ik}\varepsilon_{ik} + \tfrac{1}{2}\gamma_i \bar{\ell}^2 C_D U_o^2 \sum_{k=i+1}^{N} \rho_k d_k \bar{s}_k^2 \bar{s}_{ik}$$

$$-\tfrac{1}{2}C_D U_o^2 \bar{\ell} \sum_{k=i}^{N} \rho_k d_k \bar{s}_k \varepsilon_{ik}\left[\gamma_k \bar{\ell}\left(\bar{s}_k\bar{s}_{ik} + 2\bar{c}_k\bar{c}_{ik}\right) + \bar{s}_k\bar{c}_{ik}\delta\ell_k\right]$$

$$-C_D U_o \bar{\ell}^2 \sum_{k=1}^{N} \sum_{j=\max(i,k)}^{N} \rho_j d_j \bar{s}_j \bar{c}_{ij}\varepsilon_{ij}\varepsilon_{kj}\left(\dot{\gamma}_k \bar{\ell}\,\bar{c}_{jk} - \delta\dot{\ell}_k \bar{s}_{jk}\right) \tag{37b}$$

$$\delta D_{\beta_i} = -\tfrac{1}{2}\beta_i C_D U_o^2 \bar{\ell}^2 \bar{c}_i \sum_{k=i}^{N} \rho_k d_k \bar{s}_k^3 \varepsilon_{ik} + \tfrac{1}{2}\bar{\ell}^2 \bar{c}_i C_D U_o^2 \sum_{k=i}^{N} \beta_k \rho_k d_k \bar{s}_k \bar{c}_k^2 \varepsilon_{ik}$$

$$-\tfrac{1}{2}C_D U_o \bar{\ell}^3 \bar{c}_i \sum_{k=1}^{N} \sum_{j=\max(i,k)}^{N} \rho_j d_j \bar{s}_j \dot{\beta}_k \bar{c}_k \varepsilon_{ij}\varepsilon_{kj} \tag{37c}$$

ii. <u>Internal Damping Forces</u>

As stated earlier, η_c is the internal damping factor for the tether material and is constant for all tether elements. Since the damping force is proportional to the rate of change of the length of each tether element and acts in the direction parallel to the tether element length, the generalized damping forces are given by:

$$d_{\ell_i} = -\eta_c \delta\dot{\ell}_i \tag{38a}$$

$$d_{\gamma_i} = d_{\gamma_i} = 0 \tag{38b}$$

iii. <u>Tension Forces</u>

The tension force acts throughout the entire tether. However, only the confluence point tension force contributes to the generalized forces, Q_{q_i}, since all other tension forces within the tether are workless internal forces. Using Eq. (4), the generalized tension forces are given by:

$$T_{\ell_i} = T_x \frac{\partial x_N}{\partial \ell_i} + T_y \frac{\partial y_N}{\partial \ell_i} + T_z \frac{\partial z_N}{\partial \ell_i} \tag{39a}$$

$$T_{\gamma_i} = T_x \frac{\partial x_N}{\partial \gamma_i} + T_y \frac{\partial y_N}{\partial \gamma_i} + T_z \frac{\partial z_N}{\partial \gamma_i} \tag{39b}$$

$$T_{\beta_i} = T_x \frac{\partial x_N}{\partial \beta_i} + T_y \frac{\partial y_N}{\partial \beta_i} + T_z \frac{\partial z_N}{\partial \beta_i} \tag{39c}$$

where: T_x, T_y, and T_z are the orthogonal components of the tether tension at the confluence point.

The direction of the tension force at the confluence point is defined by a longitudinal and lateral angle, Γ_B and β_B, respectively (see Figure 12). Therefore, the three components of the tension force acting at the confluence point are:

$$T_x = T \cos\Gamma_B \cos\beta_B \tag{40a}$$

$$T_y = T \cos\Gamma_B \sin\beta_B \tag{40b}$$

$$T_z = T \sin\Gamma_B \tag{40c}$$

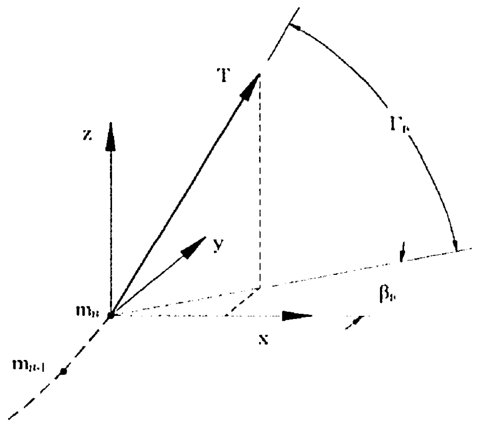


Figure 12. Tether tension at the confluence point

$$\overline{T}_x = \overline{T}\cos\overline{\Gamma}_B \tag{41a}$$

$$\overline{T}_y = 0 \tag{41b}$$

$$\overline{T}_z = \overline{T}\sin\overline{\Gamma}_B \tag{41c}$$

$$\delta T_x = \delta T\cos\overline{\Gamma}_B - \gamma_B\overline{T}\sin\overline{\Gamma}_B \tag{41d}$$

$$\delta T_y = \beta_B\overline{T}\cos\overline{\Gamma}_B \tag{41e}$$

$$\delta T_z = \delta T\sin\overline{\Gamma}_B + \gamma_B\overline{T}\cos\overline{\Gamma}_B \tag{41f}$$

Using the same method that was used for the generalized aerodynamic forces, Eqs. (39a-c) are written in small perturbation form and Eqs. (35a-i) and (41a-f) are then substituted into Eqs. (39a-c) to give the following generalized tension forces:

$$\delta T_{\ell_i} = \delta T\cos\left(\overline{\Gamma}_B - \overline{\Gamma}_i\right) + \left(\gamma_i - \gamma_B\right)\overline{T}\sin\left(\overline{\Gamma}_B - \overline{\Gamma}_i\right) \tag{42a}$$

$$\delta T_{\gamma_i} = \delta T\overline{\ell}\sin\left(\overline{\Gamma}_B - \overline{\Gamma}_i\right) + \delta\ell_i\overline{T}\sin\left(\overline{\Gamma}_B - \overline{\Gamma}_i\right) + \left(\gamma_B - \gamma_i\right)\overline{T}\,\overline{\ell}\cos\left(\overline{\Gamma}_B - \overline{\Gamma}_i\right) \tag{42b}$$

$$\delta T_{\beta_i} = \left(\beta_B - \beta_i\right)\overline{\ell}\,\overline{c}_i\overline{T}\cos\overline{\Gamma}_B \tag{42c}$$

This completes the development of the three types of generalized forces, all of which have distinct longitudinal and lateral components. Thus, the total expressions for the longitudinal and lateral generalized forces can be discussed separately.

iv. Longitudinal Generalized Forces Combined

The longitudinal generalized forces, expressed below, are given by the summation of the aerodynamic forces, the damping forces, and the tension forces.

$$\delta Q_{\ell_i} = \delta D_{\ell_i} + \delta T_{\ell_i} + d_{\ell_i} \tag{43a}$$

$$\delta Q_{\gamma_i} = \delta D_{\gamma_i} + \delta T_{\gamma_i} + d_{\gamma_i} \tag{43b}$$

By substituting Eqs. (37a), (38a), and (42a) into Eq. (43a), and Eqs. (37b), (38b), and (42b) into Eq. (43b), the longitudinal generalized forces are expanded to give the following:

$$\delta Q_{\ell_i} = -\tfrac{1}{2}\gamma_i C_D U_o^2 \bar{\ell} \sum_{k=i+1} \rho_k d_k \bar{s}_k^2 \bar{c}_{ik} - \tfrac{1}{2} C_D U_o^2 \sum_{k=i+1} \rho_k d_k \Big[\gamma_k \bar{\ell}\, \bar{s}_k \left(2\bar{c}_k \bar{s}_{ik} - \bar{s}_k \bar{c}_{ik}\right)$$

$$+ \bar{s}_k^2 \bar{s}_{ik} \delta\ell_k\Big] + C_D U_o \sum_{k=1}^{N}\left(\sum_{j=\max(i+1,k+1)}^{N} \rho_j d_j \bar{s}_j \bar{s}_{ij} \bar{\ell}\, \bar{s}_{jk} - \frac{\eta_c \delta_{ik}}{C_D U_o}\right)\delta\dot{\ell}_k$$

$$- C_D U_o \bar{\ell}^2 \sum_{k=1}^{N}\left(\sum_{j=\max(i+1,k)}^{N} \rho_j d_j \bar{s}_j \bar{s}_{ij} \bar{c}_{jk} \varepsilon_{jk}\right)\dot{\gamma}_k + \delta T \cos\left(\bar{\Gamma}_B - \bar{\Gamma}_i\right)$$

$$+ \left(\gamma_i - \gamma_B\right)\bar{T} \sin\left(\bar{\Gamma}_B - \bar{\Gamma}_i\right) \tag{44a}$$

$$\delta Q_{\gamma_i} = -\tfrac{1}{2}\delta\ell_i C_D U_o^2 \bar{\ell} \sum_{k=i}^{N} \rho_k d_k \bar{s}_k^2 \bar{c}_{ik} \varepsilon_{ik} + \tfrac{1}{2}\gamma_i \bar{\ell}^2 C_D U_o^2 \sum_{k=i+1}^{N} \rho_k d_k \bar{s}_k^2 \bar{s}_{ik}$$

$$- \tfrac{1}{2} C_D U_o^2 \bar{\ell} \sum_{k=i}^{N} \rho_k d_k \bar{s}_k \varepsilon_{ik} \left[\gamma_k \bar{\ell}\left(\bar{s}_k \bar{s}_{ik} + 2\bar{c}_k \bar{c}_{ik}\right) + \bar{s}_k \bar{c}_{ik} \delta\ell_k\right]$$

$$- C_D U_o \bar{\ell}^3 \sum_{k=1}^{N}\left(\sum_{j=\max(i,k)}^{N} \rho_j d_j \bar{s}_j \bar{c}_{ij} \varepsilon_{ij} \varepsilon_{kj} \bar{c}_{jk}\right)\dot{\gamma}_k + C_D U_o \bar{\ell}^2 \sum_{k=1}^{N}\left(\sum_{j=\max(i,k+1)}^{N} \rho_j d_j \bar{s}_j \bar{c}_{ij} \varepsilon_{ij} \bar{s}_{jk}\right)\delta\dot{\ell}_k$$

$$+ \delta T \bar{\ell} \sin\left(\bar{\Gamma}_B - \bar{\Gamma}_i\right) + \delta\ell_i \bar{T} \sin\left(\bar{\Gamma}_B - \bar{\Gamma}_i\right) + \left(\gamma_B - \gamma_i\right)\overline{T\ell} \cos\left(\bar{\Gamma}_B - \bar{\Gamma}_i\right) \tag{44b}$$

Equations (44a-b) can now be expressed in matrix form , just as the kinetic and potential energy expressions were. Assuming the generalized forces can be expressed in the following form:

$$\delta \underline{Q}_q^T = \left[\delta Q_{\ell_1}, \delta Q_{\gamma_1}, \delta Q_{\ell_2}, \delta Q_{\gamma_2}, \cdots, \delta Q_{\ell_{N-1}}, \delta Q_{\gamma_{N-1}}, \delta Q_{\ell_N}, \delta Q_{\gamma_N}\right]$$

they then can also be written as follows:

$$\delta \underline{Q}_q = -\underline{E}_{lg}\, \underline{q} - \underline{F}_{lg}\, \dot{\underline{q}} + \delta \underline{P}_{lg} \qquad \text{for i, k} = 1 \ldots N \tag{45}$$

where: $\underline{q}^T = \left[\ell_1, \gamma_1, \ell_2, \gamma_2, \cdots \ell_{N-1}, \gamma_{N-1}, \ell_N, \gamma_N\right]$

The elements of the $\underline{E}_{lg}$, $\underline{F}_{lg}$, and $\delta\underline{P}_{lg}$ matrices can be determined from examination of Eqs. (44a-b), and are given as follows:

for $i \geq k$ $\quad \left(E_{lg}\right)_{2i-1,2k-1} = 0$

for $i < k$ $\quad \left(E_{lg}\right)_{2i-1,2k-1} = \tfrac{1}{2} C_D U_o^2 \rho_k d_k \bar{s}_k^2 \bar{s}_{ik}$ (46a) for i > k $\quad \left(E_{lg}\right)_{2i-1,2k} = 0$

for i = k $\quad \left(E_{lg}\right)_{2i-1,2k} = \tfrac{1}{2} C_D U_o^2 \bar{\ell} \sum_{j=i+1}^{N} \rho_j d_j \bar{s}_j^2 \bar{c}_{ij} - \bar{T} \sin\left(\bar{\Gamma}_B - \bar{\Gamma}_i\right)$

for i < k $\quad \left(E_{lg}\right)_{2i-1,2k} = \tfrac{1}{2} C_D U_o^2 \rho_k d_k \bar{\ell} \bar{s}_k \left(2\bar{c}_k \bar{s}_{ik} - \bar{s}_k \bar{c}_{ik}\right)$ (46b)

for i > k $\quad \left(E_{lg}\right)_{2i,2k-1} = 0$

for i = k $\quad \left(E_{lg}\right)_{2i,2k-1} = \tfrac{1}{2} C_D U_o^2 \bar{\ell} \sum_{j=i}^{N} \rho_j d_j \bar{s}_j^2 \bar{c}_{ij} - \bar{T} \sin\left(\bar{\Gamma}_B - \bar{\Gamma}_i\right)$

for i < k $\quad \left(E_{lg}\right)_{2i,2k-1} = \tfrac{1}{2} C_D U_o^2 \rho_k d_k \bar{\ell} \bar{s}_k^2 \bar{c}_{ik}$ (46c)

for i > k $\quad \left(E_{lg}\right)_{2i,2k} = 0$

for i = k $\quad \left(E_{lg}\right)_{2i,2k} = -\tfrac{1}{2} C_D U_o^2 \bar{\ell}^2 \sum_{j=i+1}^{N} \rho_j d_j \bar{s}_j^2 \bar{s}_{ij} + \tfrac{1}{2} C_D U_o^2 \bar{\ell}^2 \rho_i d_i \bar{s}_i \bar{c}_i + \bar{T} \bar{\ell} \cos\left(\bar{\Gamma}_B - \bar{\Gamma}_i\right)$

for i < k $\quad \left(E_{lg}\right)_{2i,2k} = \tfrac{1}{2} C_D U_o^2 \bar{\ell}^2 \rho_k d_k \bar{s}_k \left(\bar{s}_k \bar{s}_{ik} + 2\bar{c}_k \bar{c}_{ik}\right)$ (46d)

for i ≥ k $\quad \left(F_{lg}\right)_{2i-1,2k-1} = -C_D U_o \bar{\ell} \sum_{j=i+1}^{N} \rho_j d_j \bar{s}_j \bar{s}_{ij} \bar{s}_{jk} + \eta_c \delta_{ik}$

for i < k $\quad \left(F_{lg}\right)_{2i-1,2k-1} = -C_D U_o \bar{\ell} \sum_{j=k+1}^{N} \rho_j d_j \bar{s}_j \bar{s}_{ij} \bar{s}_{jk}$ (47a)

for i ≥ k $\quad \left(F_{lg}\right)_{2i-1,2k} = C_D U_o \bar{\ell}^2 \sum_{j=i+1}^{N} \rho_j d_j \bar{s}_j \bar{s}_{ij} \bar{c}_{jk}$

for i < k $\quad \left(F_{lg}\right)_{2i-1,2k} = C_D U_o \bar{\ell}^2 \sum_{j=k}^{N} \rho_j d_j \bar{s}_j \bar{s}_{ij} \bar{c}_{jk} \varepsilon_{jk}$ (47b)

for i > k $\quad \left(F_{lg}\right)_{2i,2k-1} = -C_D U_o \bar{\ell}^2 \sum_{j=i}^{N} \rho_j d_j \bar{s}_j \bar{c}_{ij} \bar{s}_{jk} \varepsilon_{ij}$

for i ≤ k $\quad \left(F_{lg}\right)_{2i,2k-1} = -C_D U_o \bar{\ell}^2 \sum_{j=k+1}^{N} \rho_j d_j \bar{s}_j \bar{c}_{ij} \bar{s}_{jk}$ (47c)

for i > k $\quad \left(F_{lg}\right)_{2i,2k} = C_D U_o \bar{\ell}^3 \sum_{j=i}^{N} \rho_j d_j \bar{s}_j \bar{c}_{ij} \bar{c}_{jk} \varepsilon_{ij}$

for $i \geq k$ $$\left(F_{lg}\right)_{2i,2k} = C_D U_o \ell \sum_{j=k} \rho_j d_j s_j c_{ij} c_{jk} \varepsilon_{ij} \varepsilon_{kj} \qquad (47d)$$

$$\left(\delta P_{lg}\right)_{2i-1} = \delta T \cos\left(\overline{\Gamma}_B - \overline{\Gamma}_i\right) - \gamma_B \overline{T} \sin\left(\overline{\Gamma}_B - \overline{\Gamma}_i\right) \qquad (48a)$$

$$\left(\delta P_{lg}\right)_{2i} = \delta T \overline{\ell} \sin\left(\overline{\Gamma}_B - \overline{\Gamma}_i\right) + \gamma_B \overline{T} \overline{\ell} \cos\left(\overline{\Gamma}_B - \overline{\Gamma}_i\right) \qquad (48b)$$

v. Lateral Generalized Forces Combined

The lateral generalized forces, expressed in Eq. (49) below, are given by the summation of the aerodynamic forces and the tension forces.

$$\delta Q_{\beta_i} = \delta D_{\beta_i} + \delta T_{\beta_i} \qquad (49)$$

By substituting Eqs. (37c) and (42c) into Eq. (49), the lateral generalized forces are expanded to give the following:

$$\delta Q_{\beta_i} = -\tfrac{1}{2} \beta_i C_D U_o^2 \overline{\ell}^2 \overline{c}_i \sum_{k=i}^{N} \rho_k d_k \overline{s}_k^3 \varepsilon_{ik} + \tfrac{1}{2} \overline{\ell}^2 \overline{c}_i C_D U_o^2 \sum_{k=i}^{N} \beta_k \rho_k d_k \overline{s}_k \overline{c}_k^2 \varepsilon_{ik}$$
$$-\tfrac{1}{2} C_D U_o \overline{\ell}^3 \overline{c}_i \sum_{k=1}^{N} \left[\sum_{j=\max(i,k)}^{N} \rho_j d_j \overline{s}_j \varepsilon_{ij} \varepsilon_{kj} \right] \dot{\beta}_k \overline{c}_k + \left(\beta_B - \beta_i\right) \overline{\ell}\, \overline{c}_i \overline{T} \cos \overline{\Gamma}_B \qquad (50)$$

Again, as with the longitudinal generalized forces, Eq. (50) can now be expressed in matrix form. Assuming the lateral generalized forces can be expressed as follows:

$$\delta \underline{Q}_q^T = \left[\delta Q_{\beta_1}, \delta Q_{\beta_2}, \cdots, \delta Q_{\beta_{N-1}}, \delta Q_{\beta_N}\right]$$

then they can also be written in the following matrix form:

$$\delta \underline{Q}_q = -\underline{E}_{lt} \underline{q} - \underline{F}_{lt} \dot{\underline{q}} + \delta \underline{P}_{lt} \qquad \text{for } i = 1 \ldots N \qquad (51)$$

where: $\underline{q}^T = \left[\beta_1, \beta_2, \cdots \beta_{N-1}, \beta_N\right]$

The elements of the $\underline{E}_{lt}$, $\underline{F}_{lt}$, and $\delta \underline{P}_{lt}$ matrices can be determined from examination of Eq. (50), and are given by:

$$\text{for } i = k \quad \left(E_{lt}\right)_{i,k} = \tfrac{1}{2} C_D U_o^2 \bar{\ell}^2 \bar{c}_i \sum_{j=i}^{N} \rho_j d_j \bar{s}_j^3 \varepsilon_{ij} + \bar{\ell}\,\bar{c}_i \bar{T} \cos \bar{\Gamma}_B - \tfrac{1}{4} C_D U_o^2 \bar{\ell}^2 \bar{c}_i^3 \rho_i d_i \bar{s}_i$$

$$\text{for } i < k \quad \left(E_{lt}\right)_{i,k} = -\tfrac{1}{2} C_D U_o^2 \bar{\ell}^2 \bar{c}_i \rho_k d_k \bar{s}_k \bar{c}_k^2 \tag{52a}$$

$$\text{for } i > k \quad \left(F_{lt}\right)_{i,k} = \tfrac{1}{2} C_D U_o \bar{\ell}^3 \bar{c}_i \bar{c}_k \sum_{j=i}^{N} \rho_j d_j \bar{s}_j \varepsilon_{ij}$$

$$\text{for } i \leq k \quad \left(F_{lt}\right)_{i,k} = \tfrac{1}{2} C_D U_o \bar{\ell}^3 \bar{c}_i \bar{c}_k \sum_{j=k}^{N} \rho_j d_j \bar{s}_j \varepsilon_{ij} \varepsilon_{kj} \tag{52b}$$

$$\left(\delta P_{lt}\right)_i = \beta_B \bar{\ell}\,\bar{c}_i \bar{T} \cos \bar{\Gamma}_B \tag{53}$$

d. <u>Complete Tether Equations of Motion</u>

With the kinetic energy, the potential energy, and the generalized force expressions developed, the complete equations of motion, which describe both the longitudinal and lateral motions of the tether, can be written. Referring to Eq. (3), the longitudinal version of the Lagrange's Equations are as follows:

$$\frac{d}{dt}\left(\frac{\partial T_{lg}}{\partial \dot{\underline{q}}}\right) - \frac{\partial T_{lg}}{\partial \underline{q}} + \frac{\partial V_{lg}}{\partial \underline{q}} = \delta \underline{Q}_q \tag{54}$$

where: $\dfrac{\partial}{\partial \dot{\underline{q}}} = \begin{bmatrix} \dfrac{\partial}{\partial \dot{q}_1} \\ \vdots \\ \dfrac{\partial}{\partial \dot{q}_{2N}} \end{bmatrix}$ and $\dfrac{\partial}{\partial \underline{q}} = \begin{bmatrix} \dfrac{\partial}{\partial q_1} \\ \vdots \\ \dfrac{\partial}{\partial q_{2N}} \end{bmatrix}$

Recalling Eqs. (12a) and (18), the expressions for T_{lg} and V_{lg} is:

$$T_{lg} = \tfrac{1}{2} \dot{\underline{q}}_{lg}^T \underline{M}_{lg} \dot{\underline{q}}_{lg} \quad \text{and}$$

$$V_{lg} = \underline{q}_{lg}^T \underline{N} \underline{q}_{lg}$$

written in the following form:

$$\frac{\partial T_{lg}}{\partial \underline{\dot{q}}} = \tfrac{1}{2}\left(\underline{M}_{lg} + \underline{M}_{lg}^{T}\right)\underline{\dot{q}} = \underline{M}_{lg}\ \underline{\dot{q}} \tag{55a}$$

$$\frac{\partial V_{lg}}{\partial \underline{q}} = \tfrac{1}{2}\left(\underline{N} + \underline{N}^{T}\right)\underline{q} = \underline{N}\,\underline{q} \tag{55b}$$

$$\frac{\partial T_{lg}}{\partial \underline{q}} = \underline{0} \tag{55c}$$

Substituting Eqs. (55a-c) and (45) into Eq. (54) results in the set of 2N second-order linear differential equations. These equations are written in matrix form as follows:

$$\underline{M}_{lg}\,\underline{\ddot{q}} + \underline{F}_{lg}\,\underline{\dot{q}} + \left(\underline{N}_{lg} + \underline{E}_{lg}\right)\underline{q} = \delta\underline{P}_{lg} \tag{56}$$

Equation (56) is then transformed into 4N first-order differential equations by introducing a new set of variables as follows:

$$\underline{h}_{lg} = \underline{\dot{q}}$$

The resulting transformation of Eq. (56) gives:

$$\left[\begin{array}{c|c} \underline{M}_{lg} & \underline{0} \\ \hline \underline{0} & -\underline{1} \end{array}\right]\left[\begin{array}{c} \underline{\dot{h}}_{lg} \\ \hline \underline{\dot{q}} \end{array}\right] + \left[\begin{array}{c|c} \underline{F}_{lg} & \underline{N}_{lg} + \underline{E}_{lg} \\ \hline \underline{1} & \underline{0} \end{array}\right]\left[\begin{array}{c} \underline{h}_{lg} \\ \hline \underline{q} \end{array}\right] = \left[\begin{array}{c} \delta\underline{P}_{lg} \\ \hline \underline{0} \end{array}\right] \tag{57}$$

where: $\underline{1}$ is a 2N x 2N identity matrix and

$\underline{0}$ is a 2N x 2N zero matrix.

This completes the development of the longitudinal equations of motion for the discrete-element tether. The lateral equations are completed in the same manner. Again, referring to Eq. (3), the lateral version of the Lagrange's Equations are as follows:

$$\frac{d}{dt}\left(\frac{\partial T_{lt}}{\partial \underline{\dot{q}}}\right) - \frac{\partial T_{lt}}{\partial \underline{q}} = \delta\underline{Q}_{q} \tag{58}$$

where: $\dfrac{\partial}{\partial \underline{\dot{q}}} = \begin{bmatrix} \dfrac{\partial}{\partial \dot{q}_1} \\ \vdots \\ \dfrac{\partial}{\partial \dot{q}_N} \end{bmatrix}$ and $\dfrac{\partial}{\partial \underline{q}} = \begin{bmatrix} \dfrac{\partial}{\partial q_1} \\ \vdots \\ \dfrac{\partial}{\partial q_N} \end{bmatrix}$

Recalling Eq. (12b), the expression for T_{lt} is

$$T_{lt} = \tfrac{1}{2}\underline{\dot{q}}_{lt}^{T}\underline{M}_{lt}\underline{\dot{q}}_{lt}$$

and, since $\underline{M}_{lt}$ is a symmetric matrix, the partial derivatives in Eq. (58) can be written as follows:

$$\frac{\partial T_{lt}}{\partial \underline{\dot{q}}} = \tfrac{1}{2}\left(\underline{M}_{lt} + \underline{M}_{lt}^{T}\right)\underline{\dot{q}} = \underline{M}_{lt}\,\underline{\dot{q}} \tag{59a}$$

$$\frac{\partial T_{lt}}{\partial \underline{q}} = \underline{0} \tag{59b}$$

By substituting Eqs. (59a-b) and (51) into Eq. (58), the set of N second-order linear differential equations is formed and given by:

$$\underline{M}_{lt}\underline{\ddot{q}} + \underline{F}_{lt}\underline{\dot{q}} + \underline{E}_{lt}\,\underline{q} = \delta\underline{P}_{lt} \tag{60}$$

Finally, as with the longitudinal equations of motion, a new set of variables are introduced which allow Eq. (60) to be transformed into 2N first-order linear differential equations. Using the new set of variables, $\underline{h}_{lt} = \underline{\dot{q}}$, Eq. (60) is transformed into:

$$\left[\begin{array}{c|c} \underline{M}_{lt} & \underline{0} \\ \hline \underline{0} & -\underline{I} \end{array}\right]\left[\begin{array}{c} \underline{\dot{h}}_{lt} \\ \hline \underline{\dot{q}} \end{array}\right] + \left[\begin{array}{c|c} \underline{F}_{lt} & \underline{E}_{lt} \\ \hline \underline{I} & \underline{0} \end{array}\right]\left[\begin{array}{c} \underline{h}_{lt} \\ \hline \underline{q} \end{array}\right] = \left[\begin{array}{c} \delta\underline{P}_{lt} \\ \hline \underline{0} \end{array}\right] \tag{61}$$

The complete longitudinal and lateral motions of the discrete-element tether are now described by Eqs. (57) and (61), respectively. Both of these expressions are sets of first-order linear differential equations with constant coefficient matrices. These equations can then be combined with both the flight vehicle equations of motion and those that describe the kinematic relationship between the flight vehicle and the tether.

The THARWP equations of motion were developed by combining the equations of motion for both the tether and the flight vehicle with the equations that describe the kinematic relationship between the flight vehicle and the tether. In the previous section, the equations describing the longitudinal and lateral dynamic motions of the tether were developed. Below, the flight vehicle equations of motion will be developed, and then the equations that describe the kinematic relationship between the flight vehicle and the tether will be developed.

a. Flight Vehicle First-Order Dynamic Equations of Motion

The flight vehicle equations of motion are based on the dimensional first-order linearized equations of motion developed for an aircraft in steady-state flight (Refs. 6, 7). The longitudinal and lateral motions of the flight vehicle are each described by an independent set of equations:

Longitudinal

$$m\dot{u} - X_u u - X_w w + mg\theta\cos\theta_o = \Delta X_c \tag{62a}$$

$$\left(m - Z_{\dot{w}}\right)\dot{w} - Z_u u - Z_w w - \left(mU_o + Z_q\right)q + mg\theta\sin\theta_o = \Delta Z_c \tag{62b}$$

$$I_{yy}\dot{q} - \left(M_u + \frac{M_{\dot{w}}Z_u}{m - Z_{\dot{w}}}\right)u - \left(M_w + \frac{M_{\dot{w}}Z_w}{m - Z_{\dot{w}}}\right)w - \left(M_q + \frac{M_{\dot{w}}Z_q}{m - Z_{\dot{w}}}\right)q$$
$$\frac{M_{\dot{w}}mg\theta\sin\theta_o}{m - Z_{\dot{w}}} = \Delta M_c + \frac{M_{\dot{w}}\Delta Z_c}{m - Z_{\dot{w}}} \tag{62c}$$

$$\dot{\theta} - q = 0 \tag{62d}$$

Lateral

$$m\dot{v} - Y_v v - Y_p p - \left(Y_r - mU_o\right)r - mg\phi\cos\theta_o = \Delta Y_c \tag{62e}$$

$$\dot{p} - \left[\frac{L_v}{I'_x} + I'_{zx}N_v\right]v - \left[\frac{L_p}{I'_x} + I'_{zx}N_p\right]p - \left[\frac{L_r}{I'_x} + I'_{zx}N_r\right]r = \left[\frac{\Delta L_c}{I'_x} + I'_{zx}\Delta N_c\right] \tag{62f}$$

$$\dot{r} - \left[\frac{N_v}{I'_z} + I'_{zx}L_v\right]v - \left[\frac{N_p}{I'_z} + I'_{zx}L_p\right]p - \left[\frac{N_r}{I'_z} + I'_{zx}L_r\right]r$$

$$= \left[\frac{\Delta N_c}{I'_z} + I'_{zx}\Delta L_c\right] \tag{62g}$$

$$\dot{\phi} = p - r\tan\theta_o \tag{62h}$$

$$\dot{\psi} = r\sec\theta_o \tag{62i}$$

where: $I'_x = \left(I_{xx}I_{zz} - I_{zx}^2\right)/I_{zz}$

$I'_z = \left(I_{xx}I_{zz} - I_{zx}^2\right)/I_{xx}$

$I'_{zx} = I_{zx}/\left(I_{xx}I_{zz} - I_{xz}^2\right)$

Equations (62a-g) are formulated using the dimensional stability derivatives, and describe the forces and moments applied to the flight vehicle due to changes in its velocities and accelerations (both translational and rotational). The dimensional stability derivatives are the X, Y, Z, L, M, and N terms in Eqs. (62a-g). Their subscripts denote which flight vehicle motion the particular force or moment is caused by. The only exceptions to this rule are the terms with subscript c, which denote the forces and moments due to control inputs.

Each dimensional stability derivative is a function of the corresponding non-dimensional stability derivative. The relationship between the non-dimensional and the dimensional stability derivatives is consistent for all applications (Refs. 6, 7). However, the derivation of the non-dimensional stability derivatives is more specific to the particular aircraft or flight vehicle and must be obtained in the manner most suitable for the application. The non-dimensional stability derivatives for the THARWP flight vehicle were derived by referencing several related bodies of work (Ref. 6, 7, 15, 16): see Appendix 6.

The six control terms in Eqs. (62a-c) and (62e-g), ΔX_c, ΔY_c, ΔZ_c, ΔL_c, ΔM_c, and ΔN_c, account for the external forces and moments acting on the flight vehicle and originate from two sources: the link attaching the confluence point to the flight vehicle and the flight vehicle control surface deflections. Those forces and moments acting on the flight vehicle through the tether tension perturbation at the confluence point are δT, γ_B, and β_B. The control forces and moments will be defined in terms of these three variables.

First, a transformation matrix is derived to relate the attachment point coordinate system to the flight vehicle stability axes coordinate system. This transformation matrix is derived by successive rotations of Ψ, Θ, and Φ and by reversing the directions of the vertical and horizontal axes. With the implementation of the small perturbation assumptions, the coordinate transformation is given by:

$$\underline{C}_B = \begin{bmatrix} -1 & \psi & \theta \\ \psi & 1 & -\phi \\ -\theta & -\phi & -1 \end{bmatrix} \qquad \underline{C}_B^{-1} = \underline{C}_B^{T} = \begin{bmatrix} -1 & \psi & -\theta \\ \psi & 1 & -\phi \\ \theta & -\phi & -1 \end{bmatrix} \tag{63a,b}$$

The tension force acting on the end of the tether is expressed in Eqs. (40a-c). Thus, by reversing the directions of these forces, the forces acting on the flight vehicle can be written as follows:

$$F_{xc} = -T\cos\Gamma_B \tag{64a}$$

$$F_{yc} = -T\beta_B\cos\Gamma_B \tag{64b}$$

$$F_{zc} = -T\sin\Gamma_B \tag{64c}$$

The forces in Eqs. (64a-c) are expressed in terms of the attachment point coordinate system. Through the transformation, $\underline{C}_B$, these forces can be expressed in terms of the flight vehicle stability axes coordinate system:

$$F_{Xc} = T(\cos\Gamma_B - \theta\sin\Gamma_B) \tag{65a}$$

$$F_{Yc} = T(-\psi\cos\Gamma_B - \beta_B\cos\Gamma_B + \phi\sin\Gamma_B) \tag{65b}$$

$$F_{Zc} = T(\theta\cos\Gamma_B + \sin\Gamma_B) \tag{65c}$$

Finally, if the small perturbation assumption is implemented in Eqs. (65a-c), the tether tension contributions to the control terms in Eqs. (62a,b, & e) are given by:

$$\Delta X_{c_T} = \delta T\cos\bar{\Gamma}_B - (\gamma_B + \theta)\bar{T}\sin\bar{\Gamma}_B \tag{66a}$$

$$\Delta Y_{c_T} = -(\psi + \beta_B)\bar{T}\cos\bar{\Gamma}_B + \phi\bar{T}\sin\bar{\Gamma}_B \tag{66b}$$

$$\Delta Z_{c_T} = (\theta + \gamma_B)\bar{T}\cos\bar{\Gamma}_B + \delta T\sin\bar{\Gamma}_B \tag{66c}$$

The moments about the flight vehicle caused by the tether tension at the confluence point are expressed as follows:

$$\begin{bmatrix} M_{Lc} \\ M_{Mc} \\ M_{Nc} \end{bmatrix} = \underline{r}_{cp} \times \begin{bmatrix} F_{Xc} \\ F_{Yc} \\ F_{Zc} \end{bmatrix}$$

where: $\underline{r}_{cp}$ is the position vector of the confluence point relative to the flight vehicle center of gravity (see Figure 13).

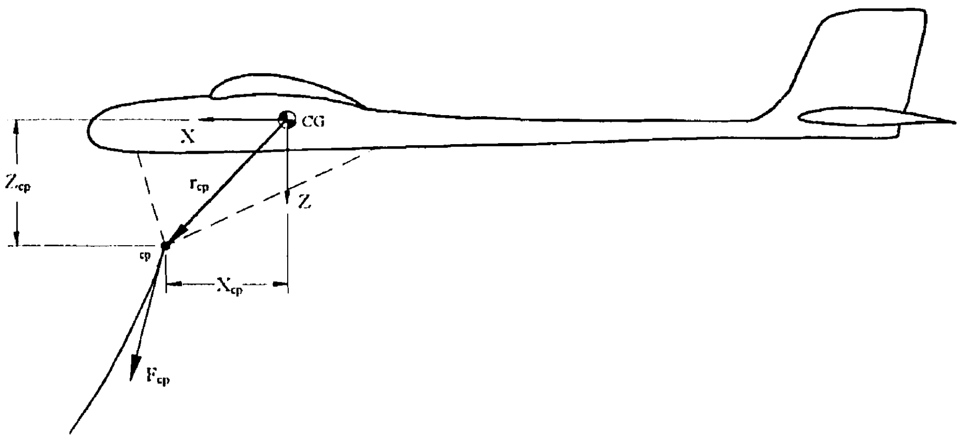


Figure 13. Position of the flight vehicle relative to the confluence point

contributions to the control terms given by:

$$\Delta L_{c_T} = Z_{cp}\overline{T}\left[(\beta_B + \psi)\cos\overline{\Gamma}_B - \phi\sin\overline{\Gamma}_B\right] \tag{66d}$$

$$\Delta M_{c_T} = -\overline{T}(\gamma_B + \theta)\left(X_{cp}\cos\overline{\Gamma}_B + Z_{cp}\sin\overline{\Gamma}_B\right) + \delta T\left(Z_{cp}\cos\overline{\Gamma}_B - X_{cp}\sin\overline{\Gamma}_B\right) \tag{66e}$$

$$\Delta N_{c_T} = -X_{cp}\overline{T}\left[(\beta_B + \psi)\cos\overline{\Gamma}_B - \phi\sin\overline{\Gamma}_B\right] \tag{66f}$$

The second component of the six control terms in Eqs. (62a-c) and (62e-g) results from the deflection of control surfaces mounted on the flight vehicle. Instead of calculating the forces and moments caused by a particular control surface deflection, six control gains were defined. These control gains specify the magnitudes of the control forces and moments as a function of the flight vehicle's CG deviation from its equilibrium position and attitude. Therefore, the perturbations of the flight vehicle's CG, both in translation and in rotation, must be determined. The rotational perturbations of the flight vehicle's CG were the Euler angles, ϕ, ψ, and θ. The total translational perturbation of the flight vehicle's CG was the sum of the confluence point perturbation and the flight vehicle's CG perturbation about the confluence point. From Eqs. (5a-c), the confluence point perturbation relative to the attachment point is given by:

$$\Delta x_{cp} = \sum_{i=1}^{N}\left(\delta\ell_i\overline{c}_i - \gamma_i\overline{\ell}\overline{s}_i\right) \tag{67a}$$

$$\Delta y_{cp} = \sum_{i=1}^{N}\beta_i\overline{\ell}\overline{c}_i \tag{67b}$$

$$\Delta z_{cp} = \sum_{i=1}^{N}\left(\delta\ell_i\overline{s}_i + \gamma_i\overline{\ell}\overline{c}_i\right) \tag{67c}$$

Next, the rotation of the flight vehicle's CG relative to the confluence point is derived. Using the coordinate transformation in Eq. (63b), the vector $\underline{r}_{cp}$ (shown in Figure 13) is transformed into the attachment point coordinate system as follows:

$$\underline{r}_{cp} = X_{cp}\underline{X} + Z_{cp}\underline{Z}$$

Combining Eqs. (67a-c) and the perturbation quantities of Eq. (68) result in the complete expressions for the flight vehicle's CG perturbation relative to the attachment point.

$$\Delta x_{cm} = -Z_{cp}\theta + \sum_{i=1}^{N}\left(\delta\ell_i \bar{c}_i - \gamma_i \bar{\ell} \bar{s}_i\right) \tag{69a}$$

$$\Delta y_{cm} = -Z_{cp}\phi + X_{cp}\psi + \sum_{i=1}^{N}\beta_i \bar{\ell} \bar{c}_i \tag{69b}$$

$$\Delta z_{cm} = X_{cp}\theta + \sum_{i=1}^{N}\left(\delta\ell_i \bar{s}_i + \gamma_i \bar{\ell} \bar{c}_i\right) \tag{69c}$$

Given the above expressions for the flight vehicle's CG perturbation, the control force vector can be written as follows:

$$\underline{\tilde{F}}^{E}_{cm} = C_{cx}\Delta x_{cm}\,\underline{x} + C_{cy}\Delta y_{cm}\,\underline{y} + C_{cz}\Delta z_{cm}\,\underline{z}$$

The terms C_{cx}, C_{cy}, and C_{cz} are the force control gains for the THARWP. Now, $\underline{\tilde{F}}^{E}_{cm}$ must be transformed back into the flight vehicle stability axes coordinate system to facilitate its substitution into Eqs. (62a, b, e). Note that, since the control force acts through the flight vehicle's CG, no moments are produced. The final control forces are given as follows:

$$\underline{\tilde{F}}^{B}_{cm} = \underline{C}_{B}\,\underline{\tilde{F}}^{E}_{cm} = \begin{bmatrix} -1 & \psi & \theta \\ \psi & 1 & -\phi \\ -\theta & -\phi & -1 \end{bmatrix}\begin{bmatrix} C_{cx}\Delta x_{cm} \\ C_{cy}\Delta y_{cm} \\ C_{cz}\Delta z_{cm} \end{bmatrix}$$

$$\Delta X_{c_c} = C_{cx}Z_{cp}\theta - C_{cx}\sum_{i=1}^{N}\left(\delta\ell_i \bar{c}_i - \gamma_i \bar{\ell} \bar{s}_i\right) \tag{70a}$$

$$\Delta Y_{c_c} = -C_{cy}Z_{cp}\phi + C_{cy}X_{cp}\psi + C_{cy}\sum_{i=1}^{N}\beta_i \bar{\ell} \bar{c}_i \tag{70b}$$

$$\Delta Z_{c_c} = -C_{cz}X_{cp}\theta - C_{cz}\sum_{i=1}^{N}\left(\delta\ell_i \bar{s}_i + \gamma_i \bar{\ell} \bar{c}_i\right) \tag{70c}$$

The final control moments are simply written as:

$$\Delta L_{c_c} = C_{cL}\phi \tag{70d}$$

$$\Delta N_{c_c} = C_{cN}\psi \tag{70f}$$

where: C_{cL}, C_{cM}, and C_{cN} are the moment control gain coefficients for the THARWP

b. Tether and Flight Vehicle Kinematic Relations

The final set of equations, which must be developed in order to complete the THARWP equations of motion, are the kinematic relations. The kinematic relations define the velocity of the flight vehicle's CG relative to the steady-state wind. These velocities are dependent on the velocities of all N tether elements and the flight vehicle's rotation rates, p, q, and r, about the confluence point. In vector form, these velocities are:

$$\underline{V}_{cm} = -U_o\underline{i} + \underline{V}_{cp} + \underline{\omega} \times \left(-\underline{r}_{cp}\right) \tag{71}$$

where: $\underline{V}_{cp} = \underline{C}_B \begin{bmatrix} \dot{x}_N \\ \dot{y}_N \\ \dot{z}_N \end{bmatrix}$

$\underline{\omega} = \begin{bmatrix} p \\ q \\ r \end{bmatrix}$ is the angular velocity of the flight vehicle,

$\underline{r}_{cp} = \begin{bmatrix} X_{cp} \\ 0 \\ Z_{cp} \end{bmatrix}$, and $\underline{V}_{cm} = \begin{bmatrix} u \\ v \\ w \end{bmatrix}$

By expanding Eq. (71), the flight vehicle velocity vector becomes:

$$\begin{bmatrix} u \\ v \\ w \end{bmatrix} = \underline{C}_B \begin{bmatrix} \dot{x}_N - U_o \\ \dot{y}_N \\ \dot{z}_N \end{bmatrix} + \begin{bmatrix} -qZ_{cp} \\ pZ_{cp} - rX_{cp} \\ qX_{cp} \end{bmatrix} \tag{71}$$

Further, by substituting Eqs. (8a-c) and (63a) in place of $\dot{x}_N$, $\dot{y}_N$, $\dot{z}_N$, and $\underline{C}_B$, and by retaining only the perturbation quantities, the final kinematic relations can be written as:

$$u = -\sum_{j=1}\left(\delta\ell_j\bar{c}_j - \dot{\gamma}_j\bar{\ell}\bar{s}_j\right) - qZ_{cp} \tag{72a}$$

$$v = -\psi U_o + \sum_{j=1}^{N}\dot{\beta}_j\bar{\ell}\bar{c}_j + pZ_{cp} - rX_{cp} \tag{72b}$$

$$w = \theta U_o - \sum_{j=1}^{N}\left(\delta\dot{\ell}_j\bar{s}_j + \dot{\gamma}_j\bar{\ell}\bar{c}_j\right) + qX_{cp} \tag{72c}$$

c. Final Formulation of the THARWP Equations of Motion

The final step in the development of the discrete-element THARWP stability analysis is the combination of all the equations developed in previous sections. All throughout the equation development, the longitudinal and lateral components were expressed independently. As a result, the longitudinal and lateral motions of the THARWP can also be solved independently. The longitudinal first-order linear differential equations of motion for the THARWP are constructed by combining Eqs. (57), (62a-d), (66a, c, e), (70a, c, e), and (72a, c). Let the matrix form of the THARWP longitudinal equations of motion be written as follows:

$$\underline{A}\,\underline{\zeta} = \underline{B}\,\dot{\underline{\zeta}} \tag{73}$$

where: $\underline{\zeta}^T = \begin{bmatrix} u & w & q & \theta & \delta T & \gamma_B & \underline{h}_{lg}^T & \underline{q}^T \end{bmatrix}$ is the 4N+6 element state vector

The contribution of the coefficient in the matrices, $\underline{A}$ and $\underline{B}$, are shown schematically in Eq. (74):

$$\left[\begin{array}{l:c:c:c}
\multicolumn{1}{l}{\text{Coefficients in 4 equations:}} & \multicolumn{3}{:c}{} \\
\multicolumn{1}{l}{\text{(62a – d) after the substitution of}} & \multicolumn{3}{:c}{\underline{0}} \\
\multicolumn{1}{l}{\text{equations (66a, c, e) and (70a, c, e)}} & \multicolumn{3}{:c}{} \\
\hdashline
\multicolumn{4}{l}{\text{2 kinematic relations: equations (72a \& c)}} \\
\hdashline
\underline{0} & \delta\underline{P}_{lg} & -\underline{F}_{lg} & -\underline{N}_{lg} - \underline{E}_{lg} \\
\hdashline
 & \underline{0} & \underline{1} & \underline{0}
\end{array}\right]
\left[\begin{array}{c} u \\ w \\ q \\ \theta \\ \hdashline \delta T \\ \gamma_B \\ \hdashline \underline{h}_{lg} \\ \hdashline \underline{q} \end{array}\right]$$

$$= \left[\begin{array}{l:c} \begin{array}{l}\text{Coefficients in 4 equations:}\\ \text{(62a – d) after the substitution}\\ \text{of equations (66a, c, e) and}\\ \text{(70a, c, e)}\end{array} & \underline{0} \\ \hdashline \multicolumn{2}{l}{\text{2 kinematic relations: equations (72a \& c)}} \\ \hdashline \underline{0} & \begin{array}{c:c} \underline{M}_{lg} & \underline{0} \\ \hdashline \underline{0} & \underline{I} \end{array} \end{array}\right] \left[\begin{array}{c} \dot{u} \\ \dot{w} \\ \dot{q} \\ \dot{\theta} \\ \hdashline \delta\dot{T} \\ \dot{\gamma}_B \\ \hdashline \dot{\underline{h}}_{lg} \\ \hdashline \dot{\underline{q}} \end{array}\right] \qquad (74)$$

Upon completion of the substitutions needed to generate matrices $\underline{A}$ and $\underline{B}$, it is important to note that there are only 4N+2 roots for this system of 4N+6 equations. The reasons for this are: (a) Eqs. (72a, c) can be rewritten in strictly algebraic form using the relation $\underline{h}_{lg} = \dot{\underline{q}}$ developed earlier (*i.e.* Eqs. (72a, c) are first-order equations only); and (b) the time derivatives of δT and γ_B are not present in Eq. (74). As a result, $\underline{B}$ contains two zero rows corresponding to Eqs. (72a, c) and two zero columns corresponding to the absence of the δT and γ_B time derivatives. Since $\underline{B}$ contains zero columns and rows (such a matrix is called a "generalized real" matrix), the inverse of $\underline{B}$ does not exist, thus rendering traditional eigensystem solvers useless. After considering the use of several commercially available computer programs, a search of the Internet revealed the availability of a public-domain package of eigensystem solver subroutines called EISPACK (Ref. 17). EISPACK, which is programmed using FORTRAN, can compute eigenvalues and eigenvectors for nine classes of matrices including generalized real matrices. The particular EISPACK subroutines required to solve Eq. (74) were converted from FORTRAN to Borland C++ and incorporated as part of the discrete-element THARWP stability analysis implemented in Trimstat. These subroutines are contained in the C++ file rggpool.cpp, which is part of the Trimstat source code listed in Appendix 14. The stability root analysis is implemented as an additional calculation option in Trimstat.

The solutions to Eq. (74) are the 4N+2 eigenvalues, also referred to as stability roots. The stability roots are either real or complex conjugate values. The real part of each stability root is the damping factor, σ, and the imaginary part of each root (if a complex stability root) the THARWP is considered dynamically stable for small perturbations. When examining stability roots, the two calculations of primary interest are the period of oscillation and the time required for the motion to reach half the initial magnitude (or double the initial magnitude if the stability root is unstable). These quantities are calculated as follows:

$$\tau = \frac{2\pi}{\omega} = \text{period of oscillation (seconds), and}$$

$$t_{\text{half/double}} = \frac{\ln 2}{\sigma} = \text{time to half/double amplitude (seconds).}$$

The shape of the motion associated with any particular stability root, whether stable or unstable, can be characterized by calculating the mode shape. The mode shape is calculated by normalizing the associated eigenvector with respect to a specified state variable. In the case of the longitudinal analysis, the normalizing variable is the pitch angle, θ.

The lateral first-order linear differential equations of motion for the THARWP are constructed in the same manner as the longitudinal case. Equations (61), (62e-i), (66b, d, f), (70b, d, f), and (72b) are combined in matrix form to give the THARWP lateral equations of motion as follows:

$$\underline{A}\,\underline{\eta} = \underline{B}\,\dot{\underline{\eta}} \qquad (75)$$

where: $\underline{\eta}^{T} = \begin{bmatrix} v & p & r & \phi & \psi & \beta_{B} & \underline{h}_{lt}^{T} & \underline{q}^{T} \end{bmatrix}$ is the 2N+6 element state vector

The coefficients of Eq. (75) are shown schematically in Eq. (76).

$$\left[\begin{array}{cc:cc}
\multicolumn{2}{l:}{\text{Coefficients in 5 equations:}} & & \\
\multicolumn{2}{l:}{\text{(62e} - \text{i) after the substitution of}} & \multicolumn{2}{c}{\underline{0}} \\
\multicolumn{2}{l:}{\text{equations (66b, d, f) and (70b, d, f)}} & & \\
\hdashline
\multicolumn{4}{l}{\text{kinematic relation from equation (72b)}} \\
\hdashline
\underline{0} & \delta\underline{P}_{lt} & -\underline{E}_{lt} & -\underline{E}_{lt} \\
\hdashline
 & \underline{0} & \underline{1} & \underline{0}
\end{array}\right]
\left[\begin{array}{c} v \\ p \\ r \\ \phi \\ \psi \\ \hdashline \beta_{B} \\ \hdashline \underline{h}_{lt} \\ \hdashline \underline{q} \end{array}\right]$$

$$= \left[\begin{array}{c:c} \begin{array}{l}\text{Coefficients in 5 equations:}\\ \text{(62e} - \text{i) after the substitution}\\ \text{of equations (66b, d, f) and}\\ \text{(70b, d, f)}\end{array} & \underline{0} \\ \hdashline \multicolumn{2}{l}{\text{kinematic relation from equations (72b)}} \\ \hdashline \underline{0} & \begin{array}{c:c} \underline{M}_{lt} & \underline{0} \\ \hdashline \underline{0} & \underline{1} \end{array} \end{array}\right] \left[\begin{array}{c} \dot{v}\\ \dot{p}\\ \dot{r}\\ \dot{\phi}\\ \dot{\psi}\\ \hdashline \dot{\beta}_B \\ \hdashline \dot{\underline{h}}_{lt} \\ \hdashline \dot{\underline{q}} \end{array}\right] \qquad (76)$$

As with the longitudinal equations, after performing all of the substitutions needed to generate matrices $\underline{A}$ and $\underline{B}$, it should be noted that the $\underline{B}$ matrix has a zero row and a zero column. The zero row occurs because Eq. (72b) is a first-order equation and the zero column occurs because of the absence of any β_B time derivative terms. As a result, the solution of the 2N+6 system of equations only produces 2N+4 stability roots. The same solution techniques used to solve the longitudinal equations of motion are used to calculate the lateral stability roots and mode shapes. These solution techniques were implemented in Trimstat along with the calculation of the longitudinal stability roots.

3. Time-Step Analysis

The solution of Eqs. (74) and (76) produced the stability roots for the THARWP and the corresponding mode shapes. The real and imaginary parts of the stability roots characterize the damping and the oscillation frequencies of the different modes of THARWP motion while the mode shapes represent the individual types of motion that the different modes contribute to the total motion of the THARWP. While the stability roots and mode shapes completely quantify the motion of the THARWP, the combined motion of all the modes may be useful to visualize the complete behavior of the tether and the flight vehicle. For this reason, a time-step analysis was completed by rewriting Eqs. (74) and (76) into second-order linear differential equations. These equations were then integrated over a specified period of time using the Newmark method (Ref. 18).

accelerations over an integration time-step are constant. The displacements and velocities are then calculated for each time step using initial conditions and the constant-acceleration assumption. The full development of the time-integration analysis is given in Appendix 7. The results from this analysis will be discussed in the following section, along with the stability roots and the mode shapes from Eqs. (74) and (76).

D. Comparison of the Discrete-Element THARWP Stability Analysis with the Three Previously Developed THARWP Stability Analyses

As stated earlier, the discrete-element THARWP stability analysis was based on a similar analysis developed for tethered aerostats (Ref. 5). In this previous work, the solutions were validated through their comparison to experimental measurements of a tethered aerostat. However, the lateral motions were the primary focus of this previous work and, as such, only the lateral equations of motion were validated by the experimental results. In the analysis of the THARWP, both the lateral and the longitudinal motions are of equal importance. Therefore, the THARWP longitudinal equations of motion must be validated before they can be used to optimize the THARWP design. Although the longitudinal equations of motion were developed using the same methodology as the THARWP lateral equations of motion, this similarity alone is not sufficient evidence that the longitudinal equations are valid. For this reason, the simplified tether models were used to develop alternate analyses for the dynamic stability of the THARWP. By using the baseline THARWP configuration in both the simplified analysis and the discrete-element analysis, the longitudinal stability results were compared. To facilitate this comparison, the baseline THARWP configuration, given in Appendix 1, was modified for input into Trimstat in the following way:

1. the internal tether damping was set to zero;
2. the tether stiffness was increased by six orders of magnitude to simulate a rigid tether; and
3. the tether was divided into two elements.

the THARWP that was analyzed in the modified string tether analysis discussed earlier.

The root-locus plots resulting from the discrete-element analysis of the baseline THARWP are given in Appendix 8. The control gain variations used to produce the simple tether model results in Appendix 5 were also used in Trimstat to produce the results in Appendix 8. By comparing the two sets of root-locus plots, distinct similarities become evident. The lightly damped aperiodic pendulum modes are virtually identical. The flight vehicle pitching mode has maintained essentially the same high damping value but has become periodic with an oscillation frequency equal to $\omega = 0.427$ rad/s. Finally, the mode of motion characterizing the interaction between the confluence point motion and the flight vehicle pitching motion has shifted slightly and has become unstable. In addition to the similarity of the damping and frequency of the three stability roots, the variations of the stability roots with respect to the modification of the control gains are very similar. These similarities strongly indicate that the longitudinal analysis, using the discrete-element technique, is valid.

A combination of the trim-state analysis and the discrete-element stability analysis, both implemented in Trimstat, were used to evaluate effects on the THARWP stability and performance due to variations in the flight-vehicle parameters. The trim-state analysis calculated the equilibrium position and AOA of the flight vehicle, as well as the longitudinal tether profile. From the lateral perspective, the tether and flight vehicle were assumed to be in a perfectly vertical plane at equilibrium. The results from the trim-state calculation were then used in the stability analysis, which calculated the stability roots and the corresponding mode shapes for the THARWP.

The stability roots provide information about the damping, oscillation, and stability associated with each mode of the THARWP motion. The stability roots are either real or complex with a real and imaginary part. In either case, the real part of the stability root is the damping of the mode of motion corresponding to the stability root of interest. If the real part is less than zero, the mode of motion is stable and will diminish over time (convergent). Alternatively, if the real part is greater than zero, the mode of motion is unstable and, over time, the motion will exceed the physical limits of the THARWP (divergent). A complex stability root represents an oscillatory mode of motion. For both stable and unstable modes of motion (determined by the real part of the stability root) the damped natural frequency (in rad/s) of an oscillation is given by the magnitude of the imaginary part of the stability root.

In addition to the stability of the different modes of motion, two other parameters are important for characterizing the THARWP modes of motion: the period of oscillation (only for oscillatory modes of motion) and the time-to-half or time-to-double amplitude. These parameters can be calculated directly, as shown below, using the real and imaginary parts of the stability roots because the equations of motion developed for the THARWP, given in Chapter II, are in dimensional form. The stability roots and the two parameters of interest are given by the following:

$$\lambda = \lambda_r \pm i\,\lambda_i$$

Period of oscillation: Time-to-half or -double amplitude:

$$T = \frac{2\pi}{\lambda_i} \text{ seconds} \qquad t_{1/2} = t_2 = \frac{\ln 2}{|\lambda_r|} \text{ seconds}$$

The time-to-half amplitude is especially useful since it indicates how quickly the THARWP will converge to its equilibrium position following an atmospheric disturbance (assuming the mode of motion is stable).

A. Formulation of the Optimization Framework

The process of optimizing the THARWP was largely one of experimentation. Many different flight-vehicle parameters were modified to gain a sense of each one's impact on the overall performance and stability of the THARWP. Before beginning, the primary goals driving the THARWP optimization were defined as follows:

1. maximize the flight vehicle equilibrium altitude;
2. minimize the drag of the flight vehicle (which will result in higher altitudes);
3. ensure that all of the possible modes of motion are dynamically stable;
4. minimize the time required for the THARWP to return to the trim state following an atmospheric disturbance; and
5. minimize the flight vehicle design complexity, which will reduce the cost and the empty weight of the flight vehicle.

Through an initial trim-state investigation, it was discovered that the value of many of the flight vehicle parameters greatly affected the equilibrium attitude of the THARWP, specifically the flight vehicle AOA. If, while varying one flight vehicle parameter, the AOA exceeds the allowable limits for the given flight vehicle configuration, then other parameters must be modified to restore the AOA to an acceptable value. With multiple parameters being modified simultaneously, the effect of any two parameter modifications would be difficult to compare. However, if the AOA is maintained at a specified value by modifying one parameter for all other parameter variations, then the results could be reasonably compared. The parameter chosen to compensate for AOA variations was the horizontal position of the confluence point because the flight vehicle AOA was most sensitive to variations of this parameter. Thus, using the fore-aft position (horizontal position) of the confluence point, the flight vehicle AOA was

AOA of zero degrees was chosen since this AOA will minimize fuselage drag. Note that the horizontal position of the confluence point, as well as the horizontal position of all other flight vehicle components, were measured relative to a reference point located at the nose of the flight vehicle (the "reference point"). A measurement aft of the reference point is positive.

Additional constraints that were specified to define the THARWP optimization process included the following assumptions:

1. the variation of the wind speed versus altitude was determined by data acquired from Ref. 11 (see Figure 3);
2. the tether was divided into six equal-length tether elements for the optimization process and then increased to twenty elements to ensure the stability of the higher order modes of motion;
3. the flight vehicle wingspan was fixed at 15 meters; and
4. the flight vehicle mass was fixed at 100 kg.

A flight vehicle with a 15 meter wingspan was estimated to have a mass of approximately 75 kg. Therefore, a 25 kg payload resulted in a total flight vehicle mass of 100 kg. These final constraints provided the framework within which the flight vehicle was optimized.

B. Discussion of the THARWP Parameter Variations

After a preliminary investigation of all of the possible flight-vehicle parameter variations, the parameters that had the most significant effect on the performance and stability of the THARWP were identified for the in-depth THARWP optimization. These parameters were as follows:

- the flight vehicle moments of inertia, I_{xx}, I_{yy}, and I_{zz};
- the tether material properties;
- the minimum angle of the tether at the attachment point;
- the dihedral of the wing;

- the horizontal location of the flight vehicle's CG;
- the horizontal location of the wing;
- the horizontal location of the vertical stabilizer;
- the horizontal location of a second vertical stabilizer, while the original vertical stabilizer remained in the position designated by the baseline THARWP configuration;
- the horizontal location of the horizontal stabilizer;
- the span of the horizontal stabilizer;
- the horizontal location of a second horizontal stabilizer, while the original horizontal stabilizer remained in the position designated by the baseline THARWP configuration; and
- the six control gains: C_{cx}, C_{cy}, C_{cz}, C_{cL}, C_{cM}, and C_{cN}.

For tethered vehicles, the tether length is a significant factor in determining stability. However, for the THARWP stability analysis, the tether length was not included as a variable parameter because it was calculated as part of the THARWP trim-state analysis and could not be varied directly.

In a linearized dynamic stability analysis, such as the one developed for the THARWP and implemented in Trimstat, the longitudinal and lateral motions can be calculated independently. Consequently, the variations of certain flight vehicle parameters will affect only the longitudinal or the lateral motion. The stability roots affected by the THARWP parameter variations, listed above, are discussed in the sections below and shown graphically in Appendix 9. All stability roots not discussed are highly-damped stable roots associated with the internal vibrations of the tether.

1. <u>Moments of Inertia, I_{xx}, I_{yy}, and I_{zz}</u>

Since the aerodynamics of the flight vehicle are not affected by variations of the flight vehicle's moments of inertia, the trim state of the THARWP remains constant for all moment of inertia variations. Variations of the moments of inertia about the X and Z flight vehicle axes, I_{xx} and I_{zz}, affect only the lateral stability of the THARWP. As shown in Figure A9.1, varying I_{xx} results in changes to two lateral stability roots: one aperiodic and one oscillatory. The vehicle rapidly converging back to the trim-state position after a small perturbation. This mode of motion remains highly damped and stable for the entire variation of I_{xx}. The oscillatory stability root characterizes the moderately damped lateral motion of the flight vehicle about the confluence point (independent of the tether motion). The damping of this motion, which has a time-to-half amplitude of approximately two seconds, slightly decreases with increasing I_{xx}, but remains stable for all values examined.

Observation of the root-locus plot in Figure A9.2 shows that the oscillatory lateral motion of the flight vehicle about the confluence point is the only mode of motion affected by variations of I_{zz}. As the magnitude of I_{zz} is increased, the frequency of vibration decreases while the damping remains relatively constant. By referring to Figures A9.1 and A9.2, it is clear that the baseline THARWP configuration has one unstable, aperiodic, lateral stability root which is unaffected by the variations of both I_{xx} and I_{zz}.

Of the three moments of inertia investigated, only the moment of inertia about the Y axis of the flight vehicle, I_{yy}, affects the longitudinal motions of the THARWP. The results from the variation of I_{yy} are presented in Figure A9.3. Two oscillatory stability roots, one stable and one unstable, vary steadily with I_{yy}. The stable stability root characterizes a highly damped pitching motion of the flight vehicle about the confluence point (analogous to short period motion in traditional aircraft). For the baseline THARWP, the period and time-to-half amplitude of this mode of motion is:

$T = 29.0$ seconds $\qquad\qquad t_{½} = 0.7$ seconds

As would be expected for this type of oscillation, as I_{yy} is increased, the oscillation frequency decreases. The second stability root affected by the variation of I_{yy} characterizes an unstable oscillatory motion in which the flight vehicle horizontal- and vertical-velocity variations are coupled with vibrations of the tether. This unstable motion has the following characteristics for the baseline THARWP configuration:

$T = 7.6$ seconds $\qquad\qquad t_2 = 2.8$ seconds fairly rapidly. As I_{yy} is increased, the time-to-double amplitude increases from 2.43 seconds to 3.78 seconds, and the period increases from 6.74 seconds to 9.44 seconds. These results indicate that increasing the flight vehicle's moment of inertia about the Y axis may aid in producing a stable THARWP design.

2. Tether Material

Two different tether materials were investigated for use in the THARWP: high-strength, 302 stainless-steel wire (Ref. 13) and light-weight Kevlar-49 string (Ref. 12). Kevlar-49 is a composite material known for its high strength-to-weight ratio and its high energy-absorbing characteristics. An investigation of these two tether materials was completed previously in the feasibility study. The preliminary results of this study indicated that the steel tether would allow the THARWP to achieve higher trim-state altitudes while requiring a shorter tether. To ensure that this tether material selection was also optimum for the dynamic performance of the THARWP, the stability characteristics of the THARWP were evaluated for the two tether materials.

Initially, the tether of the baseline THARWP configuration was specified as 1.0 mm diameter, high-strength, 302 stainless-steel wire. After evaluating the trim-state and stability performance of the baseline THARWP, the steel tether was replaced with a tether made of 5.0 mm diameter Kevlar-49 string. Note that this tether diameter is approximately the smallest practical size for a composite string of this kind. Upon evaluating the trim-state and the stability performance of the THARWP using the Kevlar-49 tether it was found that, despite the fact that the density of 302 stainless-steel is almost eight times greater than the density of Kevlar-49, the smaller diameter stainless-steel tether has a much lower associated drag force resulting in a better performing THARWP. For comparison, the trim-state tether profile for the THARWP, with both the stainless-steel tether and the Kevlar-49 tether, are shown in Figure A9.4. The longitudinal and lateral stability roots were calculated for both tether types and no material for the THARWP is 302 stainless-steel.

3. Tether Angle at the Attachment Point

The tether angle at the attachment point is specified as part of the initial conditions for the trim-state analysis. When this angle is set larger than 0°, the trim-state altitude is calculated by iterating through successive tether elements until the tether element is found with the specified angle. In practice, if the tether angle at the attachment point is 0°, a small length of the tether, downwind of the attachment point, will lay flat on the ground and the tether will begin to curve up to the flight vehicle. Then, if the tether was drawn in using a winch or a similar device, the total tether length would decrease and the angle of the tether at the attachment point would increase. In this study, the tether angle at the attachment point was varied from 0° to 70° in 10° increments. In addition to the usual modification of the horizontal confluence point location, needed to maintain a constant flight-vehicle AOA, the tether element length was also varied for each tether angle setting. The tether element length was varied to allow the total tether length to shorten while maintaining six tether elements.

The significant changes in the trim-state results, due to the variation of the tether angle at the attachment point, are shown in Figure A9.5. One important observation is that the minimum tether angle at the attachment point can be increased from 0° to 30° without either a large decrease in the maximum achievable altitude or a large increase in the maximum tether tension (the confluence point tension). An additional consequence of increasing the minimum tether angle to 30° is that the total length of the tether decreases by approximately 2500 m, thus reducing the cost of the tether. The variations of the tether angle at the attachment point also effect the stability of the entire THARWP. The effects on the longitudinal and lateral stability roots are shown in Figures A9.6a and A9.6b, respectively. In all cases, the magnitudes of the damping and the frequency increase as the tether angle at the attachment point is increased. This will improve the stability of those modes of motion that are already stable, but will also cause any unstable modes of motion to become more unstable. Therefore, THARWP design only if all the stability roots are already stable.

4. Flight Vehicle Wing Dihedral

The dihedral variation of the flight vehicle's wing affected only the lateral stability of the THARWP. The dihedral was varied from -10° to 40° in 5° increments. The resulting affect on the lateral stability roots is shown in Figure A9.7. The three modes of motion that were discussed previously (with reference to the moments of inertia, I_{xx} and I_{zz}) were those most affected by the dihedral variation. The highly damped stable mode of motion became even more heavily damped with a new time-to-half amplitude of $t_{½} = 0.26$ seconds. At a dihedral angle of 40°, the moderately divergent, unstable mode of motion nearly became stable with a new time-to-double amplitude of $t_2 = 9.97$ seconds. The third mode of lateral motion affected by the dihedral angle variation was the moderately-damped oscillatory motion of the flight vehicle about the confluence point. The frequency of this oscillatory motion grew steadily higher with the increase of the dihedral angle while the damping decreased at first and then began to increase again for dihedral angles above 25°. These results, shown graphically in Figure A9.7, indicate that the dihedral variation of the flight vehicle's wing is an important variable in the design of an optimal THARWP design.

5. Horizontal Position of the Flight Vehicle CG

The horizontal position of the flight vehicle's CG is a critical factor in determining the stability characteristics of the THARWP and will, naturally, be an important factor in the design of an optimal THARWP. Therefore, the variation of the CG position affects both the longitudinal and lateral motions of the THARWP. To assess the impact on the longitudinal and lateral stability, the CG position was varied from 1.40 meters to 2.00 meters aft of the reference point. The stability roots that were significantly affected by the variation of the horizontal position of the CG were plotted in root-locus plots for the longitudinal and lateral modes of motion (Figures A9.8a and A9.8b, respectively). The longitudinal root-locus plot root became stable. The opposite effect occurred with lateral stability roots. As the flight vehicle's CG was moved aft, the unstable aperiodic lateral stability root became increasingly unstable and the stable aperiodic lateral stability root became less stable. In terms of the lateral stability, the only positive effect resulting from moving the flight vehicle's CG aft was the oscillatory stability root, which characterizes the lateral oscillations of the flight vehicle about the confluence point, becoming more stable.

6. Horizontal Position of the Wing

The horizontal position of the wing, as measured from the reference point to the aerodynamic center of the wing, is also critical in determining both the longitudinal and lateral stability characteristics of the THARWP. The wing position was varied from 1.60 meters to 0.60 meters aft of the reference point, and the resulting longitudinal and lateral stability roots are presented in Figures A9.9a and A9.9b, respectively. The results presented in the longitudinal root-locus plot show that, as the wing was moved forward on the flight vehicle, the unstable longitudinal stability root became stable. Therefore, by moving the wing in the opposite direction that the flight vehicle CG was moved, the same stability results are achieved. The lateral stability results for the wing position variation were also similar to those for the CG variations. As the wing was moved forward on the flight vehicle, the two lateral aperiodic modes of motion, one stable and the other unstable, both became increasingly unstable, while the oscillatory mode of motion became increasingly stable.

7. Horizontal Position of the Vertical Stabilizer

Using the span and mean chord of the vertical stabilizer specified in the baseline THARWP design, the position of the vertical stabilizer was varied from 2.0 meters in front of, to 6.0 meters aft of, the reference point. This variation has no effect on the trim-state or the longitudinal stability characteristics of the THARWP. However, the horizontal position of the vertical stabilizer does significantly affect the lateral stability characteristics of the THARWP as shown in Figure A9.10. As the vertical stabilizer was moved forward on the flight vehicle, the unstable aperiodic mode of motion became more stable. In addition, the frequency of the lateral oscillatory motion decreased as the vertical stabilizer was moved forward. These results are corroborated by previously completed work (Ref. 1), which demonstrated the stability enhancing properties of a forward mounted vertical stabilizer on rigid-wing kites.

8. Horizontal Position of a Second Vertical Stabilizer

With the benefit of a forward mounted vertical stabilizer shown, consideration was given to adding a second vertical stabilizer to the flight vehicle. The horizontal position of a second vertical stabilizer was varied, while the original vertical stabilizer, as specified by the baseline THARWP configuration, was retained. The specifications of the second vertical stabilizer were identical to those of the baseline vertical stabilizer. The position of the second vertical stabilizer was varied from 2.0 meters in front of, to 6.0 meters aft of, the reference point. These variations affected only the lateral stability characteristics of the THARWP. The results, shown in Figure A9.11, are very similar to the results for the variation of the position of the original vertical stabilizer: see Figure A9.10. In this case, however, the stabilizing effect is not as great due to the presence of the aft vertical stabilizer.

9. Horizontal Position of the Horizontal Stabilizer

The position of the horizontal stabilizer was varied to determine its affect on the longitudinal stability of the THARWP. Using the horizontal stabilizer span and chord defined by the baseline THARWP, the position of the horizontal stabilizer was varied from the baseline position, 6.0 meters aft of the reference point, forward to the reference point. Unlike the examination of the vertical stabilizer, a change in the position of the horizontal stabilizer caused the trim-state AOA to change. Therefore, with every variation of the horizontal stabilizer position, the horizontal position of the confluence point was modified to maintain a trim-state AOA equal to 0°. Despite the resulting confluence point variations, the lateral stability characteristics were unaffected by the horizontal position variation of the horizontal stabilizer. As expected, the longitudinal stability characteristics were greatly affected by these variations (see Figure A9.12). Upon observation of the results in Figure A9.12, the most significant result was that the unstable oscillatory mode of motion became stable as the position of the horizontal stabilizer neared the nose of the flight vehicle. By comparison, the other stable root shown was affected only slightly. The position of the horizontal stabilizer will be very important in the optimization of the THARWP's dynamic performance.

10. The Span of the Horizontal Stabilizer

Also of interest was the importance of the horizontal stabilizer's span in determining the longitudinal stability characteristics. Using the baseline THARWP configuration, the horizontal stabilizer span was varied from 2.0 to 5.0 meters. The results, shown in Figure A9.13a, were surprising since they indicate that the unstable oscillatory pitching motion of the flight vehicle will become more unstable as the area of the horizontal stabilizer is increased. This result may indicate that the moments generated by the aerodynamic forces acting on the horizontal stabilizer are critical in causing the flight vehicle's pitching motion to become unstable. This is consistent with work completed in Ref. 1 where, by moving the horizontal stabilizer forward on the flight vehicle, the unstable oscillatory pitching motion became stable. To investigate the effects of horizontal stabilizer span variations further, the horizontal stabilizer was positioned at the nose of the flight vehicle and then the span was varied the same as before. The results indicate that, once the oscillatory pitching motion is stable, increasing the horizontal stabilizer span improves the stability of the THARWP (see Figure A9.13b).

11. Horizontal Position of a Second Horizontal Stabilizer

As was done in the analysis of the vertical stabilizer, the effects on the longitudinal THARWP stability characteristics were calculated for a second horizontal stabilizer. The position of the second horizontal stabilizer was varied from 6.0 meters aft of, forward to 0.5 meters in front of, the reference point. By comparing the results of this parameter variation, A9.12, the longitudinal stability in both cases is affected in the same manner. However, due to the presence of the original horizontal stabilizer, the magnitude of the effects on the longitudinal stability are smaller.

12. Flight Vehicle Control Gains

The flight vehicle control gains are the coefficients that relate the magnitude of the control forces and moments to the translational and rotational perturbations of the flight vehicle's CG (relative to the attachment point). The three control gains, C_{cx}, C_{cy}, and C_{cz}, are based on the translational perturbations, x, y, and z, respectively. Similarly, the three control gains, C_{cL}, C_{cM}, and C_{cN}, are based on the rotational perturbations, ϕ, θ, and ψ, respectively. The control forces and moments generated as a result of these control gains are idealized. That is, the exact forces and moments are specified to act on the flight vehicle's CG without any consideration as to how these forces and moments may be generated in practice. The longitudinal control gains are C_{cx}, C_{cz}, and C_{cM}, and the lateral control gains are C_{cy}, C_{cL}, and C_{cN}.

The motivation for including control gains in the THARWP stability analysis was the anticipation of instabilities that are inherent in high-altitude, tethered vehicles such as the THARWP. Although the goal of the THARWP optimization was to design a stable THARWP, it is possible that the final THARWP design may still have unstable modes of motion during deployment, normal operation, and recovery. In these cases, additional control forces and moments, implemented through an onboard control system, would be required to stabilize the THARWP. In order to obtain a sense for how each control gain will affect the THARWP stability, a general variation of each control gain was performed and the results presented in root-locus plots. The results from the variation of the longitudinal control gains are shown in Figures A9.15a, A9.15b, and A9.15c, while the results for the variation of the lateral control gains are shown in Figures A9.16a, A9.16b, and A9.16c. Many of the modes of motion that were affected by the previously-discussed parameter variations were also affected by the unstable modes of motion, the control gains will be used in an attempt to force the unstable modes of motion to become stable. However, as shown in Figures A9.15a-c and A9.16a-c, the control gains can not be used solely to stabilize the THARWP.

C. Airframe Modifications Discussed

The motivation for the study of the parameter variations discussed above was to assess the behaviour of the stability roots while different THARWP parameters were varied. In the final step of the optimization process, the knowledge gained above was used to selectively modify different THARWP parameters. The goal of these modifications was to improve the THARWP's stability characteristics over those of the baseline THARWP. Many different THARWP configurations were analyzed during the optimization process. As the THARWP optimization progressed, the parameters that were most useful in improving the stability of the THARWP were as follows:

- the position of the second horizontal and vertical stabilizers;
- the area of both the original, and the second horizontal and vertical stabilizers;
- the position of the wing;
- the dihedral of the wing; and
- the position of the confluence point.

As a result of these parameter iterations, the final optimized THARWP design was stable in all modes of motion without the need for any control gains. However, there were several modes of motion, in both the longitudinal and lateral sense, that were very nearly unstable. In addition, during the launch and recovery phases of THARWP operation, the tether will be shorter than it was optimized for and may cause the THARWP to become unstable (Ref. 1). For these reasons, several control gain combinations were analyzed culminating in the following control gain settings:

$C_{cx} = -10.0$ N/m	$C_{cz} = -10.0$ N/m	$C_{cM} = -50.0$ N-m/rad
$C_{cy} = 0$ N/m	$C_{cL} = -150.0$ N-m/rad	$C_{cN} = 0$ N-m/rad

in conjunction with the stability-root analysis to better visualize the type of oscillations the THARWP would experience when disturbed. The time-step analysis requires a set of initial disturbance forces which cause the THARWP to oscillate. The initial disturbance forces were calculated to simulate an increased wind speed of 5.0 m/s, in both the longitudinal and the lateral directions, for a time duration of 0.5 seconds. After 0.5 seconds the wind speeds returned to their equilibrium values (*i.e.*, the disturbance forces return to zero). The longitudinal and lateral responses following the initial disturbance are shown in Figures A9.17a and A9.17b. These figures illustrate how the heavily damped motions diminish rapidly, while the long-term motions are defined by the lightest damped and the lowest frequency motions.

The final optimized THARWP design differs from the baseline THARWP design in the following ways:

- the span of the aft horizontal stabilizer was increased to 4.0 meters;
- an additional horizontal stabilizer, with a span of 3.0 meters and a chord of 1.0 meters, was positioned 1.0 meter in front of the reference point;
- a second vertical stabilizer, identical in span and mean chord to the original one, was also positioned 1.0 meter in front of the reference point;
- the flight vehicle CG was moved aft to a new position of 1.53 meters aft of the reference point;
- the wing dihedral was increased to 15°; and
- the position of the confluence point was moved forward to 1.391 meters aft of the reference point.

The complete specifications for the final optimized THARWP configuration, including the control gain settings, are listed in Appendix 10. The final THARWP configuration is shown schematically in Figure A10.1.

The linearized, discrete-element stability analysis developed above is limited to describing the THARWP's small-perturbation characteristics. When stability analyses are developed for traditional aircraft, the small-perturbation stability characteristics are generally considered to be a good indication of the overall stability characteristics. Although the extension of this assumption to the THARWP may be valid for constant operating conditions, inevitably there will be situations when the atmospheric conditions will deviate from those specified for the THARWP's equilibrium. The atmospheric condition most likely to vary significantly is the prevailing wind speed. Where wind-speed variation is experienced, the forces acting on the THARWP would either force it to move to a new equilibrium position or cause the THARWP to become non-linearly unstable. Evaluation of the impact that these atmospheric variations may have on both the trim-state and stability of the THARWP is not possible with the linearized stability analysis developed in this research. Only a non-linear stability analysis developed for the THARWP would allow the evaluation of these types of motions. Therefore, the control system developed by using the linearized discrete-element stability analysis is for enhancing the stability of the THARWP with respect to small perturbations only.

The THARWP optimization process, discussed in Chapter III, resulted in a THARWP design that is dynamically stable without the need for an additional control system. However, upon closer investigation, several longitudinal and lateral modes of motion were nearly unstable. For this reason, the control gains were optimized to improve the THARWP's dynamic stability. The control gains that were finally chosen (see Appendix 10) determine the required behaviour of the THARWP control system.

As discussed in Section III, the control gains define the magnitude of the control forces and moments acting on the flight vehicle's CG for a given flight-vehicle perturbation. These control forces and moments may be generated by deflecting control surfaces located on the wing and on the vertical and horizontal stabilizers, both fore and aft. The algorithm governing the deflection of these control surfaces must account for the aerodynamic characteristics of the flight vehicle. This is crucial since generating a particular force or moment without also generating other unwanted forces or moments will require simultaneous deflection of a combination of the available control surfaces. The final component required to implement the control system is a method for accurately measuring, in real-time, the absolute position and attitude of the flight vehicle. To accomplish this, one of two systems could be used:

1. a Global Positioning System (GPS) receiver would monitor the absolute position of the flight vehicle while rotational accelerometers would provide flight vehicle attitude changes; and
2. both translational and rotational accelerometers would provide the flight vehicle position and attitude changes.

Due to advances in the electronics industry, the components required for either of these systems are very small and lightweight. Implementation of either control system may be suitable for the THARWP.

The goal of this research was to show that the THARWP could provide a simple, inexpensive, and stable alternative to tethered aerostats for positioning payloads at high altitudes. Helium-filled, tethered aerostats are currently used for positioning payloads at higher altitudes. However, these tethered aerostats are very complex and expensive, and are sensitive to atmospheric variations which may result in large variations in the payload position.

Previous research on tethered vehicles has generally focused on tethered aerostats or objects towed behind aircraft or boats. The analyses developed in these previous works have, for the most part, drastically simplified or omitted altogether the contribution of the tether to the dynamics of the total system. However, one of these previous studies focused specifically on the development of an accurate tether model, where the tether was modeled by using a discrete-element technique that allowed for the evaluation of the dynamic tether motions. Since the tether dynamics are an important component of the total THARWP motion, the discrete-element tether model was implemented in the THARWP stability analysis.

The feasibility of the THARWP concept was completed by developing a trim-state analysis which calculated the equilibrium position of the tether and the flight vehicle. Through this analysis, it was shown that an example THARWP is capable of positioning a payload mass of 25 kg at approximately 15 km altitude. Subsequent to the trim-state analysis, the THARWP stability analysis was developed using the discrete-element tether model. The research through which this model was originally developed focused primarily on the lateral motions of a tethered aerostat. As a result, only the lateral implementation of the discrete-element tether model was validated. Thus, before the THARWP longitudinal equations of motion were used to optimize the THARWP design, their validity was confirmed by comparing their results with results from a THARWP stability analysis based on a simple tether model.

The optimization of the THARWP design demonstrated that a high performance flight vehicle could be made stable . Although most high-altitude tethered vehicles are inherently unstable, a stable THARWP was achieved by manipulating many of the design parameters. The parameters found to be most influential in determining the stability characteristics of the vehicle's CG. The resulting THARWP design was stable without the need for any control gains, at its operational altitude. However, the margin of stability was very small for some modes of motion. Therefore, control gains were used to improve the margin of stability for the optimized THARWP design.

The THARWP stability analysis was developed using several assumptions. The most significant of these was that the tether and the flight vehicle experience small motions only. One of the implications of this assumption was that the results from the stability analysis, as well as the control-system design, were valid only for small perturbations of the THARWP. Although this was a significant limitation, the atmospheric conditions can be assumed reasonably constant over a limited period of time. If, at some future time, the characterization of the THARWP's performance and stability for larger perturbations are desired, then a non-linear stability analysis of the THARWP will be required.

[1] Bryant L.W. Brown W.S. and Sweeting N.W. 1942. Collected Researches on the Stability of Kites and Towed Gliders. ARC R.&M. No. 2303.

[2] Etkin B. and Mackworth J.C. 1963. Aerodynamic Instability of Non-lifting Bodies Towed Beneath an Aircraft. Tech. Note 65. University of Toronto, Institute for Aerospace Studies.

[3] DeLaurier J.D. 1976. Unpublished Notes. "The Cable-Body Equations". University of Toronto, Institute for Aerospace Studies.

[4] DeLaurier J.D. 1976. Unpublished Notes. "Longitudinal Forces and Moments Due to a Massless, Dragless 'String' Cable". University of Toronto, Institute for Aerospace Studies.

[5] Wong K. 1986. A Stability Analysis of an Aerostat with a Discrete-Element Dynamic Tether. M.A.Sc. diss., University of Toronto, Institute for Aerospace Studies.

[6] Etkin B. 1959. Dynamics of Fight, Stability and Control. New York: John Wiley & Sons, Inc.

[7] Etkin B. and Reid L.D. 1996. Dynamics of Flight, Stability and Control. 3rd ed. New York: John Wiley & Sons, Inc.

[8] Shevell R.S. 1989. Fundamentals of Flight. 2nd ed. New Jersey: Prentice-Hall, Inc.

[9] Abbott I.H. and Von Doenhoff A.E. 1959. Theory of Wing Sections. New York: Dover Publications, Inc.

[10] Raymer D.P. 1989. Aircraft Design: A Conceptual Approach. Washington: AIAA, Inc.

[11] Larry Misquez. "Wind Velocity Profiles - May 10, 1995 through May 20, 1995". TECOM, White Sands Missile Range.

[12] Marshall A.D. 1989. Composite Basics. 2nd ed. Marshall Consulting.

[13] Product Specification for 302 Stainless Steel Wire (Precision Brand <http://www.precisionbrand.com/catalog/wire.htm>)

Wesley Publishing Company.

[15] Campbell J.P. and McKinney M.O. 1953. Summary of Methods for Calculating Dynamic Lateral Stability and Response, and for Estimating Lateral Stability Derivatives. NACA Report TN 1098. Langley Aeronautical Laboratory, Langley Field, Va.

[16] DeLaurier J.D. 1996. Unpublished Notes. "Advanced Flight Dynamics". University of Toronto, Institute for Aerospace Studies.

[17] EISPACK Matrix Eigensystem Routines (NATS Project at Argonne National Laboratory <dongarra@cs.utk.edu>)

[18] Huebner K.H. and Thornton E.A. 1986. The Finite Element Method for Engineers. 2nd ed. New York: John Wiley & Sons, Inc.

[19] Flannery B.P., Press W.H., Teukolsky S.A., and Vetterling W.T. 1995. Numerical Recipes in C: the art of scientific computing. 2nd ed. New York: Cambridge University Press.

General Specifications

Flight vehicle mass (includes payload)	100.0 kg
Horizontal position of the flight vehicle's CG	1.5 m
Vertical position of the flight vehicle's CG	0.05 m
Horizontal position of the confluence point	1.5 m
Vertical position of the confluence point	-0.5 m
I_{xx} moment of inertia	1500 kg m^2
I_{yy} moment of inertia	1700 kg m^2
I_{zz} moment of inertia	2000 kg m^2
I_{xz} moment of inertia	10 kg m^2
Tether material density	7860 kg/m^3
Tether elastic modulus	200 GPa
Tether diameter	0.001 m
Tether cross-flow coefficient of drag	0.8
Tether axial-flow coefficient of drag	0.1
Internal tether damping	0.0 N s/m
Minimum tether angle at the attachment point	0°

Flight Vehicle Wing

Span	15.0 m
Root chord	1.5 m
Tip chord	1.5 m
Lift-curve slope	5.0 /rad
Maximum coefficient of lift, C_{Lmax}	2.0
Coefficient of moment, C_M	0.1
Dihedral	5°
Leading edge sweep	0°
¼ line sweep	0°
Incidence angle	8°
Zero lift angle	13°
Horizontal position of the aerodynamic center (AC)	1.5 m
Vertical position of the AC	0.2 m
Position of the AC aft of the leading edge (given as a percentage of the mean chord length)	0.3
Zero lift downwash angle	1°

Horizontal Stabilizer #1

Span	3.0 m
Mean chord	0.5 m
Lift-curve slope	4.0 /rad
Horizontal position of the AC	6.0 m
Vertical position of the AC	0.1 m
Incidence angle	0°
Zero lift angle	0°

Vertical Stabilizer #1

Span	1.25 m
Mean chord	0.5 m
Lift-curve slope	4.0 /rad
Horizontal position of the AC	6.0 m
Vertical position of the AC	0.6 m

Horizontal Stabilizer #2

Span	0.0 m
Mean chord	0.0 m
Lift-curve slope	0.0 /rad
Horizontal position of the AC	0.0 m
Vertical position of the AC	0.0 m
Incidence angle	0°
Zero lift angle	0°

Vertical Stabilizer #2

Span	0.0 m
Mean chord	0.0 m
Lift-curve slope	0.0 /rad
Horizontal position of the AC	0.0 m
Vertical position of the AC	0.0 m

Fuselage

Total enclosed volume	1.0 m^3
Maximum width	0.25 m
Maximum height	0.5 m
Width at ¼ fuselage length	0.22 m
Height at ¼ fuselage length	0.45 m
Width at ¾ fuselage length	0.1 m
Height at ¾ fuselage length	0.15 m
Characteristic length	2.0 m
Total length of the fuselage	6.0
Fuselage lift-curve slope	0.0525 /rad
Height of center of pressure	0.1 m
Horizontal position of the volume center	1.3

Control Gains

C_{CX}	0.0	C_{CY}	0.0
C_{CZ}	0.0	C_{CL}	0.0
$C_{C\theta}$	0.0	C_{CN}	0.0

The simple THARWP equations of motion were developed by combining the flight vehicle equations of motion with expressions for the external forces applied to the flight vehicle due to the tether attachment. The flight vehicle equations of motion were based on the general equations of unsteady motion traditionally used to analyze the stability of aircraft (Refs. 6, 7). These equations including the forces and moments due to the tether attachment at the confluence point (denoted by the subscript $_{cp}$), are as follows:

$$X + X_{cp} - mgSIN\Theta = m(\dot{U} + QW - RV) \tag{2.1a}$$

$$Y + Y_{cp} + mgCOS\Theta SIN\Phi = m(\dot{V} + RU - PW) \tag{2.1b}$$

$$Z + Z_{cp} + mgCOS\Theta SIN\Phi = m(\dot{W} + PV - QU) \tag{2.1c}$$

$$L + L_{cp} = I_{xx}\dot{P} - I_{xz}\dot{R} + QR(I_{zz} - I_{yy}) - I_{xz}PQ \tag{2.1d}$$

$$M + M_{cp} = I_{yy}\dot{Q} + RP(I_{xx} - I_{zz}) + I_{xz}(P^2 - R^2) \tag{2.1e}$$

$$N + N_{cp} = -I_{xz}\dot{P} + I_{zz}\dot{R} + PQ(I_{yy} - I_{xx}) + I_{xz}QR \tag{2.1f}$$

Using the small perturbation linearized technique, the following assumptions can be applied to the development of the THARWP stability analysis:

1. the variables U, V, W, P, Q, R, Θ, Φ, Ψ, and the forces and moments can be expressed in terms of an equilibrium component and a perturbation component: $U = U_o+u$, $P = p_o+p$, $\Theta = \theta_o+\theta$, $X = X_o + \Delta X$, etc.;
2. only first-order terms will be retained;
3. for the reference flight condition, the flight vehicle is fixed in space and has no angular velocity ($v_o = p_o = q_o = r_o = \phi_o = \psi_o = 0$);
4. the stability axes are chosen such that $w_o = 0$ and $U_o = U_\infty$;
5. trigonometric functions can be simplified as follows:
   $SIN(\theta_o + \theta) = SIN\theta_o + \theta COS\theta_o$
   $COS(\theta_o + \theta) = COS\theta_o - \theta SIN\theta_o$

By applying the above rules and by subtracting out the reference flight condition, Eqs. (2.1a-f) become:

$$\Delta X + \Delta X_{cp} - mg\theta COS\theta_o = m\dot{u} \tag{2.2a}$$

$$\Delta Y + \Delta Y_{cp} + mg\phi COS\theta_o = m(\dot{v} + rU_o) \tag{2.2b}$$

$$\Delta Z + \Delta Z_{cp} - mg\theta SIN\theta_o = m(\dot{w} - qU_o) \tag{2.2c}$$

$$\Delta L + \Delta L_{cp} = I_{xx}\dot{p} - I_{xz}\dot{r} \tag{2.2d}$$

$$\Delta M + \Delta M_{cp} = I_{yy}\dot{q} \tag{2.2e}$$

$$\Delta N + \Delta N_{cp} = -I_{xz}\dot{p} + I_{zz}\dot{r} \tag{2.2f}$$

Now, the Δ() terms can be defined in terms of the flight vehicle's dimensional stability derivatives as follows:

$$\Delta X = X_u u + X_w w + \Delta X_c \tag{2.3a}$$

$$\Delta Y = Y_v v + Y_p p + Y_r r + \Delta Y_c \tag{2.3b}$$

$$\Delta Z = Z_u u + Z_w w + Z_{\dot{w}}\dot{w} + Z_q q + \Delta Z_c \tag{2.3c}$$

$$\Delta L = L_v v + L_p p + L_r r + \Delta L_c \tag{2.3d}$$

$$\Delta M = M_u u + M_w w + M_{\dot{w}}\dot{w} + M_q q + \Delta M_c \tag{2.3e}$$

$$\Delta N = N_v v + N_p p + N_r r + \Delta N_c \tag{2.3f}$$

Substituting Eqs. (2.3a-f) into Eqs. (2.2a-f) gives the general THARWP equations of motion:

$$mg\theta COS\theta_o + m\dot{u} - X_u u - X_w w - \Delta X_c = \Delta X_{cp} \tag{2.4a}$$

$$m\dot{v} - mg\phi COS\theta_o - Y_v v - Y_p p - (Y_r - mU_o)r - \Delta Y_c = \Delta Y_{cp} \tag{2.4b}$$

$$mg\theta SIN\theta_o - Z_u u - Z_w w + (m - Z_{\dot{w}})\dot{w} - (mU_o + Z_q)q - \Delta Z_c = \Delta Z_{cp} \tag{2.4c}$$

$$I_{xx}\dot{p} - I_{xz}\dot{r} - L_v v - L_p p - L_r r - \Delta L_c = \Delta L_{cp} \tag{2.4d}$$

$$I_{yy}\dot{q} - M_u u - M_w w - M_{\dot{w}}\dot{w} - M_q q - \Delta M_c = \Delta M_{cp} \tag{2.4e}$$

$$I_{zz}\dot{r} - I_{xz}\dot{p} - N_v v - N_p p - N_r r - \Delta N_c = \Delta N_{cp} \tag{2.4f}$$

motions of the flight vehicle are described by two independent sets of equations. The longitudinal motions are described by Eqs. (2.4a, c, e) and the lateral motions are described by Eqs. (2.4b, d, f).

The next step in developing the THARWP equations of motion is to define the tether geometry at the confluence point. The diagrams in Figure 2.1a and Figure 2.1b show both the lateral and longitudinal geometry of the tether as well as the coordinate systems used to relate the flight vehicle to the tether. From these diagrams, the force acting on the flight vehicle at the confluence point can be written in terms of the slope of the tether at the confluence point.

$$\overline{F}_{cp} = T_a\left[\left(\frac{\partial\tilde{X}}{\partial S}\right)_a \overline{e}_1 - \left(\frac{\partial\tilde{Y}}{\partial S}\right)_a \overline{e}_2 + \left(\frac{\partial\tilde{Z}}{\partial S}\right)_a \overline{e}_3\right] \qquad (2.5)$$

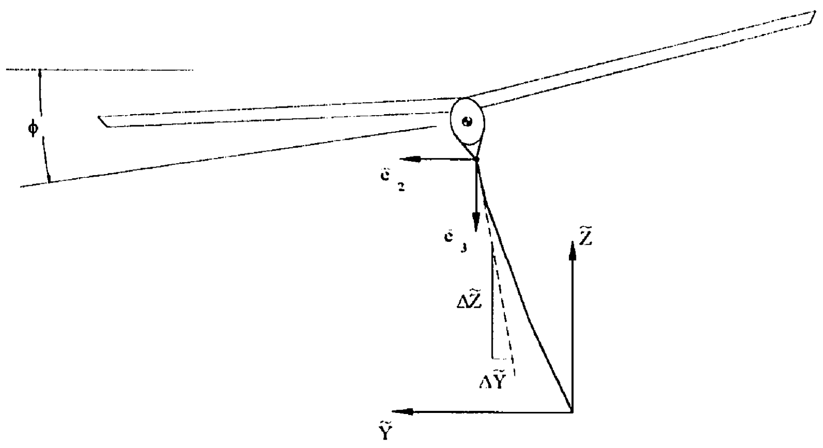


Figure 2.1a. Lateral view of the simple tether model

Then, for the small perturbation analysis, the tether coordinates, $\overline{e}_1, \overline{e}_2$, and $\overline{e}_3$, can be transformed to the flight vehicle coordinates, $\overline{n}_1, \overline{n}_2$, and $\overline{n}_3$, using:

$$\overline{e}_1 = \overline{n}_1 - \psi\overline{n}_2 + \theta\overline{n}_3 \qquad (2.6a)$$

$$\overline{e}_2 = \psi\overline{n}_1 + \overline{n}_2 - \phi\overline{n}_3 \qquad (2.6a)$$

$$\overline{e}_3 = -\theta\overline{n}_1 + \phi\overline{n}_2 + \overline{n}_3 \qquad (2.6a)$$

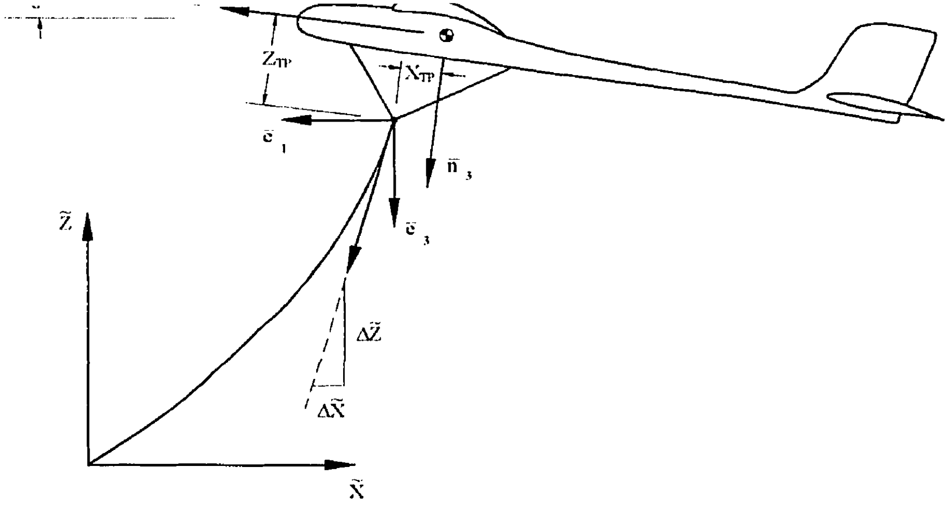


Figure 2.1b Longitudinal view of the simple tether model

Applying the coordinate transformation in Eqs. (2.6a-c) to Eq. (2.5) gives the following:

$$X_{cp} = T_a\left[\left(\frac{\partial\tilde{X}}{\partial S}\right)_a - \left(\frac{\partial\tilde{Y}}{\partial S}\right)_a \psi - \left(\frac{\partial\tilde{Z}}{\partial S}\right)_a \theta\right] \tag{2.7a}$$

$$Y_{cp} = T_a\left[-\left(\frac{\partial\tilde{X}}{\partial S}\right)_a \psi - \left(\frac{\partial\tilde{Y}}{\partial S}\right)_a + \left(\frac{\partial\tilde{Z}}{\partial S}\right)_a \phi\right] \tag{2.7b}$$

$$Z_{cp} = T_a\left[\left(\frac{\partial\tilde{X}}{\partial S}\right)_a \theta + \left(\frac{\partial\tilde{Y}}{\partial S}\right)_a \phi + \left(\frac{\partial\tilde{Z}}{\partial S}\right)_a\right] \tag{2.7c}$$

Further, if slope terms are split into equilibrium and perturbation components, the perturbation components of Eqs. (2.7a-c) become:

$$\Delta X_{cp} = T_o\left[\left(\frac{\partial\tilde{X}'}{\partial S}\right)_a - \overline{\left(\frac{\partial\tilde{Z}}{\partial S}\right)_a} \theta\right] \tag{2.8a}$$

$$\Delta Y_{cp} = T_o\left[-\overline{\left(\frac{\partial\tilde{X}}{\partial S}\right)_a} \psi - \left(\frac{\partial\tilde{Y}'}{\partial S}\right)_a + \overline{\left(\frac{\partial\tilde{Z}}{\partial S}\right)_a} \phi\right] \tag{2.8b}$$

$$\Delta Z_{cp} = T_o\left[\overline{\left(\frac{\partial\tilde{X}}{\partial S}\right)_a} \theta + \left(\frac{\partial\tilde{Z}'}{\partial S}\right)_a\right] \tag{2.8c}$$

where: $(\bar{\ })$ indicates the equilibrium component, and

$(')$ indicates the perturbation component.

By referring back to Figures 2.1a and 2.1b, the perturbation components of the moments acting on the flight vehicle due to the cable are:

$$\Delta M_{cp} = \Delta X_{cp} Z_{TP} - \Delta Z_{cp} X_{TP} \qquad (2.9a)$$

$$\Delta L_{cp} = -\Delta Y_{cp} Z_{TP} \qquad (2.9b)$$

$$\Delta N_{cp} = \Delta Y_{cp} X_{TP} \qquad (2.9c)$$

With the force and moment inputs known for the flight vehicle, the relationship between the flight vehicle's coordinates, $\bar{n}_1, \bar{n}_2$, and $\bar{n}_3$, and the attachment point coordinates, $\bar{a}_1, \bar{a}_2$, and $\bar{a}_3$, is derived. These coordinate systems are shown schematically in Figure 2.2. First, the expression for the velocity of the flight vehicle's CG can be written as follows:

$$\overline{V}_{cm} = U\bar{n}_1 + V\bar{n}_2 + W\bar{n}_3 = (U_o + \dot{X})\bar{e}_1 + \dot{Y}\bar{e}_2 + \dot{Z}\bar{e}_3 \qquad (2.10)$$

By applying the coordinate transformations given by Eqs. (2.6a-c) to Eq. (2.10), the velocity of the flight vehicle's CG is given by:

$$U = (U_o + \dot{X}) + \dot{Y}\psi - \dot{Z}\theta \qquad (2.11a)$$

$$V = -(U_o + \dot{X})\psi + \dot{Y} + \dot{Z}\phi \qquad (2.11b)$$

$$W = (U_o + \dot{X})\theta - \dot{Y}\phi + \dot{Z} \qquad (2.11c)$$

Now, assuming small perturbations of $\dot{X}, \dot{Y}, \text{and } \dot{Z}$ in Eqs. (2.11a-c), the velocity transformations become:

$$U = (U_o + \dot{X}) \qquad (2.12a)$$

$$v = -U_o\psi + \dot{Y} \qquad (2.12b)$$

$$w = U_o\theta + \dot{Z} \qquad (2.12c)$$

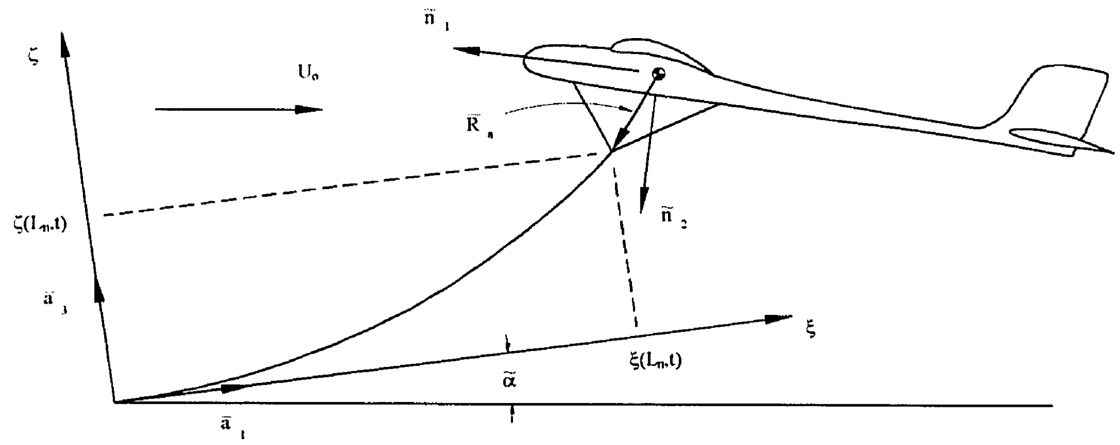


Figure 2.2. Coordinate systems used for the simple tether THARWP analysis

Further, these velocities are transformed into the attachment point coordinates, $\bar{a}_1, \bar{a}_2$, and $\bar{a}_3$, and given in terms of $\dot{\xi}(L_n,t)$, $\dot{\zeta}(L_n,t)$, and $\dot{\tilde{Y}}(L_n,t)$. First, the velocity at the tether point is:

$$\overline{V}_{TP} = \overline{V}_{cm} + \overline{\Omega} \times \overline{R}_a \tag{2.13}$$

where: $\overline{\Omega} = p\bar{n}_1 + q\bar{n}_2 + r\bar{n}_3$

$\overline{R}_a = X_{TP}\bar{n}_1 + Z_{TP}\bar{n}_3$

and, with the small perturbation assumptions on p, q, and r, and the inverse of Eqs. (2.6a-c), the velocity at the tether point can be expanded further to:

$$\overline{V}_{TP} = (U_o + \dot{X} + \dot{\theta}Z_{TP})\bar{e}_1 + (\dot{Y} + \dot{\psi}X_{TP} - \dot{\phi}Z_{TP})\bar{e}_2 + (\dot{Z} - \dot{\theta}X_{TP})\bar{e}_3 \tag{2.14}$$

At the same time, the velocity at the tether point can also be expressed in terms of $\dot{\xi}(L_n,t)$, $\dot{\zeta}(L_n,t)$, and $\dot{\tilde{Y}}(L_n,t)$, and the attachment point coordinates, $\bar{a}_1, \bar{a}_2$, and $\bar{a}_3$. This expression comes from the observation of Figure 2.2.

$$\overline{V}_{TP} = U_o\bar{e}_1 + \dot{\xi}(L_n,t)\bar{a}_1 + \dot{\tilde{Y}}(L_n,t)\bar{e}_2 + \dot{\zeta}(L_n,t)\bar{a}_3 \tag{2.15}$$

Now, by using the following relations:

$$\bar{e}_1 = -\cos\tilde{\alpha}\,\bar{a}_1 + \sin\tilde{\alpha}\,\bar{a}_3 \quad \text{and} \quad \bar{e}_3 = -\cos\tilde{\alpha}\,\bar{a}_3 - \sin\tilde{\alpha}\,\bar{a}_1$$

Eqs. (2.14) and (2.15) are combined to give expressions for $\dot{\xi}(L_n,t)$, $\dot{\zeta}(L_n,t)$, and $\dot{\tilde{Y}}(L_n,t)$ as follows:

$$\dot{\xi}(L_n,t) = -\cos\tilde{\alpha}(\dot{X} + \dot{\theta}Z_{TP}) - \sin\tilde{\alpha}(\dot{Z} - \dot{\theta}X_{TP}) \tag{2.16a}$$

$$\dot{\tilde{Y}}(L_n,t) = \dot{Y} + \dot{\psi}X_{TP} - \dot{\phi}Z_{TP} \tag{2.16b}$$

$$\dot{\zeta}(L_n,t) = \sin\tilde{\alpha}(\dot{X} + \dot{\theta}Z_{TP}) - \cos\tilde{\alpha}(\dot{Z} - \dot{\theta}X_{TP}) \tag{2.16c}$$

The final step in the development of the coordinate transformation is to manipulate Eqs. (2.16a-c) to get expressions for $\dot{X}$, $\dot{Y}$, and $\dot{Z}$. By taking $\left[\text{Eq. (2.16a)}\cdot\sin\tilde{\alpha} + \text{Eq. (2.16c)}\cdot\cos\tilde{\alpha}\right]$ we get:

$$\dot{Z} = -\dot{\xi}(L_n,t)\sin\tilde{\alpha} - \dot{\zeta}(L_n,t)\cos\tilde{\alpha} + \dot{\theta}X_{TP}$$

Similarly, by taking $\left[\text{Eq. (2.16a)}\cdot\cos\tilde{\alpha} - \text{Eq. (2.16c)}\cdot\sin\tilde{\alpha}\right]$ we get:

$$\dot{X} = -\dot{\xi}(L_n,t)\cos\tilde{\alpha} + \dot{\zeta}(L_n,t)\sin\tilde{\alpha} - \dot{\theta}Z_{TP}$$

Then, assuming that $\tilde{\alpha}$ is arranged such that $\dot{\zeta}(L_n,t) >> \dot{\xi}(L_n,t)$, the final form of the velocity transformations are given by:

$$\dot{X} = \dot{\zeta}(L_n,t)\sin\tilde{\alpha} - \dot{\theta}Z_{TP} \tag{2.17a}$$

$$\dot{Y} = \dot{\tilde{Y}}(L_n,t) - \dot{\psi}X_{TP} + \dot{\phi}Z_{TP} \tag{2.17b}$$

$$\dot{Z} = -\dot{\zeta}(L_n,t)\cos\tilde{\alpha} + \dot{\theta}X_{TP} \tag{2.17c}$$

In addition to Eqs. (2.17a-c), a transformation was required to relate the change in the direction cosines for a vector from the $\tilde{X}$- $\tilde{Z}$ coordinate system to the $\zeta(L_n,t)$- $\xi(L_n,t)$ coordinate system. From figure 2.3. we can see that the transformation is simply as follows:

$$\cos\psi' = \cos(\phi' + \theta') = \cos\phi'\cos\theta' - \sin\phi'\sin\theta'$$

$$\sin\psi' = \sin(\phi' + \theta') = \sin\phi'\cos\theta' + \cos\phi'\sin\theta'$$

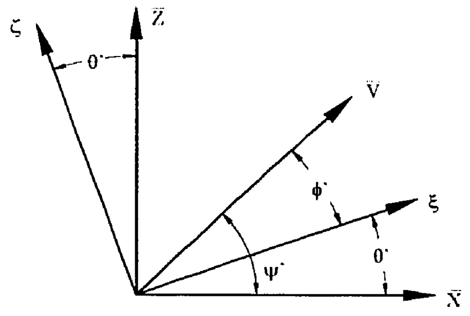


Figure 2.3. Direction cosines for an arbitrary vector, $\overline{V}$

In the form given above, the transformation is not useful. However, these equations can be expressed in terms of the tether slope at the confluence point by using the following relations:

$$\cos\psi' = \left(\frac{\partial\tilde{X}}{\partial S}\right)_a \qquad \cos\phi' = \left(\frac{\partial\xi}{\partial S}\right)(L_n,t)$$

$$\cos\theta' = \overline{\left(\frac{\partial\tilde{X}}{\partial S}\right)}_a \qquad \sin\phi' = \left(\frac{\partial\zeta}{\partial S}\right)(L_n,t)$$

$$\sin\psi' = \left(\frac{\partial\tilde{Z}}{\partial S}\right)_a \qquad \sin\theta' = \overline{\left(\frac{\partial\tilde{Z}}{\partial S}\right)}_a$$

By combining these terms with the two equations above, the expression for $\left(\frac{\partial\zeta}{\partial S}\right)(L_n,t)$ is found to be:

$$\left(\frac{\partial\zeta}{\partial S}\right)(L_n,t) = \frac{1}{\tilde{D}}\left[\left(\frac{\partial\tilde{Z}'}{\partial S}\right)_a \overline{\left(\frac{\partial\tilde{X}}{\partial S}\right)}_a - \left(\frac{\partial\tilde{X}'}{\partial S}\right)_a \overline{\left(\frac{\partial\tilde{Z}}{\partial S}\right)}_a\right] \tag{2.18}$$

$$\text{where:}\quad \tilde{D} = \overline{\left(\frac{\partial\tilde{X}}{\partial S}\right)}_a^{\,2} - \overline{\left(\frac{\partial\tilde{Z}}{\partial S}\right)}_a^{\,2}$$

conditions resulting in an expression in terms of $\left(\frac{\partial\zeta}{\partial S}\right)(L_n, t)$ and the first and second time derivative of ζ.

To simplify the solution of the longitudinal equations of motion, $\left(\frac{\partial\zeta}{\partial S}\right)(L_n, t)$ is written in terms of ζ using the following approximations:

1. the tether is approximately straight;
2. the tether experiences small, rigid-body type perturbation rotations about the attachment point;
3. the small perturbation component of $\tilde{\alpha}$ is given by $\alpha = \zeta/\ell$ where ℓ is the distance from the attachment point to the confluence point (the tether length);
4. referring to Figure 2.4, $\chi = \overline{\chi} + \chi'$ is the tether slope angle in the ζ - ξ reference frame; and
5. the small perturbation component of the tether slope, χ', is equivalent to the small perturbation tether rotation angle, α.

Using the expression for the sine of the tether angle in the ζ - ξ reference frame, the expression for the perturbation quantity of $\left(\frac{\partial\zeta}{\partial S}\right)(L_n, t)$ is given by:

$$\left(\frac{\partial\zeta'}{\partial S}\right)(L_n, t) = \frac{1}{\ell}\cos\left(\sin^{-1}\overline{\left(\frac{\partial\tilde{Z}}{\partial S}\right)_n} - \tilde{\alpha}\right)\zeta \tag{2.19}$$

At this point, all of the component equations have been developed and the final step in the development of the THARWP equations of motion will be the combination of the correct sets of equations into a form that facilitates their solution.

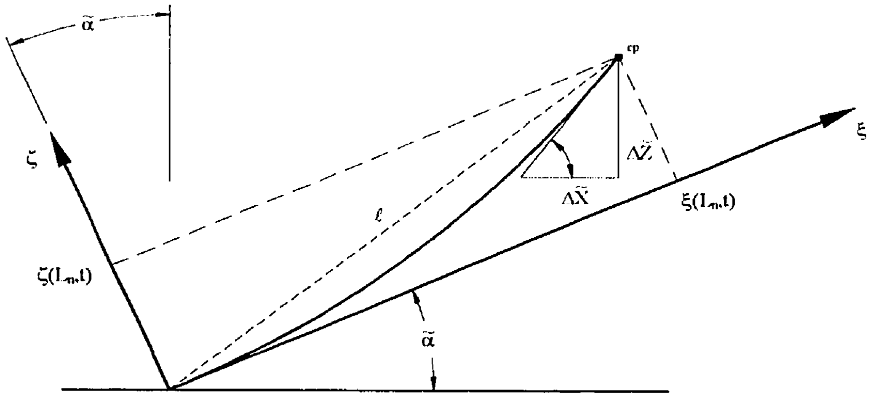


Figure 2.4. Tether geometry used to estimate the tether slope perturbation

<u>Longitudinal Solution</u>

The longitudinal problem is defined by the small perturbation THARWP equations, Eqs. (2.4a, c, e), the cable force equations, Eqs. (2.8a, c), the auxiliary condition, Eq. (2.9a), the velocity transformation, Eqs. (2.12a, c), and the transformation equations, Eqs. (2.17a, c) and (2.18). First, substitute Eqs. (2.12a, c) into Eqs. (2.4a, c, e) to give:

$$mg\theta COS\theta_o + m\ddot{X} - X_u\dot{X} - X_w U_o\theta - X_w\dot{Z} - \Delta X_c = \Delta X_{cp} \qquad (2.20a)$$

$$mg\theta SIN\theta_o - Z_u\dot{X} - Z_w U_o\theta - Z_w\dot{Z} + (m - Z_{\dot{w}})(U_o\dot{\theta} + \ddot{Z})$$
$$- (mU_o + Z_q)q - \Delta Z_c = \Delta Z_{cp} \qquad (2.20b)$$

$$I_{yy}\dot{q} - M_u\dot{X} - M_w(U_o\theta + \dot{Z}) - M_{\dot{w}}(U_o\dot{\theta} + \ddot{Z}) - M_q q - \Delta M_c = \Delta M_{cp} \qquad (2.20c)$$

Next, substitute Eqs. (2.20a-c) into Eqs. (2.8a, c) and (2.9a), and write the results in the following form:

$$[\;]\,X + [\;]\,Z + [\;]\,\theta + \text{control terms}$$

The resulting equations are as follows:

$$\left\{[mZ_{TP}]\frac{d^2}{dt^2} + [M_u - Z_{TP}X_u + X_{TP}Z_u]\frac{d}{dt}\right\}X + \left\{[M_{\dot{w}} + X_{TP}Z_{\dot{w}} - mX_{TP}]\frac{d^2}{dt^2}\right.$$

$$+\left[M_w - Z_{TP}X_w + X_{TP}Z_w\right]\frac{d}{dt}\Big\}Z + \Big\{\left[-I_{yy}\right]\frac{d^2}{dt^2} + \Big[U_oM_{\dot w} + M_q + X_{TP}U_oZ_{\dot w}$$

$$+ X_{TP}Z_q\Big]\frac{d}{dt} + U_oM_w + Z_{TP}mg\cos\theta_o - Z_{TP}U_oX_w + X_{TP}U_oZ_w$$

$$- X_{TP}mg\sin\theta_o\Big\}\theta + \Delta M_c - Z_{TP}\Delta X_c + X_{TP}\Delta Z_c = 0 \tag{2.21a}$$

$$\left\{\left[\frac{m}{T_o}\right]\frac{d^2}{dt^2} - \left[\frac{X_u}{T_o}\right]\frac{d}{dt}\right\}X + \left\{-\left[\frac{X_w}{T_o}\right]\frac{d}{dt}\right\}Z + \Bigg\{\frac{1}{T_o}\left[mg\cos\theta_o - U_oX_w\right]$$

$$+\overline{\left(\frac{\partial\tilde Z}{\partial S}\right)_a}\Bigg\}\theta - \frac{\Delta X_c}{T_o} = \left(\frac{\partial\tilde X'}{\partial S}\right)_a \tag{2.21b}$$

$$\left\{-\left[\frac{Z_u}{T_o}\right]\frac{d}{dt}\right\}X + \left\{\frac{1}{T_o}\left[m - Z_{\dot w}\right]\frac{d^2}{dt^2} - \left[\frac{Z_w}{T_o}\right]\frac{d}{dt}\right\}Z + \Bigg\{\frac{1}{T_o}\left[-U_oZ_{\dot w} - Z_q\right]\frac{d}{dt}$$

$$+\frac{1}{T_o}\left[mg\sin\theta_o - U_oZ_w\right] - \overline{\left(\frac{\partial\tilde X}{\partial S}\right)_a}\Bigg\}\theta - \frac{\Delta Z_c}{T_o} = \left(\frac{\partial\tilde Z'}{\partial S}\right)_a \tag{2.21c}$$

The coordinate transformations can now be substituted into the three above equations: Eqs. (2.17a, c) into Eq. (2.21a), and Eqs. (2.18) and (2.19) into Eqs. (2.21b, c). The result is the system of two second-order differential equations in terms of $\zeta(L_n, t)$ and θ that describe the longitudinal, small perturbation motion of the THARWP:

$$\Bigg[\left\{\left[mZ_{TP}\right]\frac{d^2}{dt^2} + \left[M_u - Z_{TP}X_u + X_{TP}Z_u\right]\frac{d}{dt}\right\}\sin\tilde\alpha + \Bigg\{\left[mX_{TP} - M_{\dot w} - X_{TP}Z_{\dot w}\right]\frac{d^2}{dt^2}$$

$$+\left[Z_{TP}X_w - M_w - X_{TP}Z_w\right]\frac{d}{dt}\Bigg\}\cos\tilde\alpha\Bigg]\zeta(L_n, t) + \Big\{\Big[-I_{yy} - mZ_{TP}^2 + X_{TP}M_{\dot w}$$

$$+ X_{TP}^2Z_{\dot w} - mX_{TP}^2\Big]\frac{d^2}{dt^2} + \Big[U_oM_{\dot w} + M_q + X_{TP}U_oZ_{\dot w} + X_{TP}Z_q - Z_{TP}M_u + Z_{TP}^2X_u$$

$$+ X_{TP}^2Z_{\dot w} - mX_{TP}^2\Big]\frac{d^2}{dt^2} + \Big[U_oM_{\dot w} + M_q + X_{TP}U_oZ_{\dot w} + X_{TP}Z_q - Z_{TP}M_u + Z_{TP}^2X_u$$

$$- X_{TP}Z_{TP}Z_u + X_{TP}M_w - X_{TP}Z_{TP}X_w + X_{TP}^2Z_w\Big]\frac{d}{dt} + \Big[U_oM_w + Z_{TP}mg\cos\theta_o$$

(2.22a)

$$
\begin{aligned}
&\left[\left\{\left[-\overline{\left(\frac{\partial \tilde{Z}}{\partial S}\right)_a}\,\frac{m}{T_o}\right]\frac{d^2}{dt^2}+\left[\overline{\left(\frac{\partial \tilde{Z}}{\partial S}\right)_a}\,\frac{X_u}{T_o}-\overline{\left(\frac{\partial \tilde{X}}{\partial S}\right)_a}\,\frac{Z_u}{T_o}\right]\frac{d}{dt}\right\}\sin\tilde{\alpha}+\left\{\left[\overline{\left(\frac{\partial \tilde{X}}{\partial S}\right)_a}\,\frac{Z_{\dot{w}}-m}{T_o}\right]\right.\right.\\
&\left.\left.\frac{d^2}{dt^2}+\left[\overline{\left(\frac{\partial \tilde{X}}{\partial S}\right)_a}\,\frac{Z_w}{T_o}-\overline{\left(\frac{\partial \tilde{Z}}{\partial S}\right)_a}\,\frac{X_w}{T_o}\right]\frac{d}{dt}\right\}\cos\tilde{\alpha}-\frac{\tilde{D}}{\ell}\cos\left(\sin^{-1}\overline{\left(\frac{\partial \tilde{Z}}{\partial S}\right)_a}-\tilde{\alpha}\right)\right]\zeta(L_n,t)\\
&+\left\{\left[\overline{\left(\frac{\partial \tilde{Z}}{\partial S}\right)_a}\,\frac{mZ_{TP}}{T_o}+\overline{\left(\frac{\partial \tilde{X}}{\partial S}\right)_a}\,\frac{X_{TP}\left[m-Z_{\dot{w}}\right]}{T_o}\right]\frac{d^2}{dt^2}+\left[\overline{\left(\frac{\partial \tilde{X}}{\partial S}\right)_a}\,\frac{Z_{TP}Z_u}{T_o}\right.\right.\\
&\left.-\overline{\left(\frac{\partial \tilde{Z}}{\partial S}\right)_a}\,\frac{Z_{TP}X_u+X_{TP}X_w}{T_o}-\overline{\left(\frac{\partial \tilde{X}}{\partial S}\right)_a}\,\frac{X_{TP}Z_w+U_oZ_{\dot{w}}+Z_q}{T_o}\right]\frac{d}{dt}\\
&\left.+\overline{\left(\frac{\partial \tilde{X}}{\partial S}\right)_a}\,\frac{mg\sin\theta_o-U_oZ_w}{T_o}-\overline{\left(\frac{\partial \tilde{X}}{\partial S}\right)_a}^{2}+\overline{\left(\frac{\partial \tilde{Z}}{\partial S}\right)_a}\,\frac{U_oX_w-mg\cos\theta_o}{T_o}-\overline{\left(\frac{\partial \tilde{Z}}{\partial S}\right)_a}^{2}\right\}\theta\\
&+\overline{\left(\frac{\partial \tilde{Z}}{\partial S}\right)_a}\,\frac{\Delta X_c}{T_o}-\overline{\left(\frac{\partial \tilde{X}}{\partial S}\right)_a}\,\frac{\Delta Z_c}{T_o}=0 \qquad (2.22b)
\end{aligned}
$$

Next, the control expressions must be defined in terms of $\zeta(L_n,t)$ and θ and substituted into Eqs. (2.22a, b). By referring to Figure 2.2a, the expression for the position of the THARWP mass center can be written as follows:

$$x_{cm}=\xi(L_n,t)\cos\tilde{\alpha}-\zeta(L_n,t)\sin\tilde{\alpha}+X_{TP}\cos\theta+Z_{TP}\sin\theta$$

$$z_{cm}=\xi(L_n,t)\sin\tilde{\alpha}+\zeta(L_n,t)\cos\tilde{\alpha}-X_{TP}\sin\theta+Z_{TP}\cos\theta$$

Assuming that $\tilde{\alpha}$ is arranged such that $\xi(L_n,t)$ remains essentially constant for small perturbations, the above equations then become:

$$x_{cm}=-\zeta(L_n,t)\sin\tilde{\alpha}+\left(-X_{TP}\sin\theta_o+Z_{TP}\cos\theta_o\right)\theta \qquad (2.23a)$$

$$z_{cm}=\zeta(L_n,t)\cos\tilde{\alpha}+\left(-X_{TP}\cos\theta_o-Z_{TP}\sin\theta_o\right)\theta \qquad (2.23b)$$

Finally, the control expressions are written as functions of Eqs. (2.23a, b):

$$\Delta X_c = \left[G_X + G_{Xd} \frac{d}{dt} \right] \cdot x_{cm} \qquad \Delta Z_c = \left[G_Z + G_{Zd} \frac{d}{dt} \right] \cdot z_{cm} \tag{2.24a b}$$

$$\Delta M_c = \left[G_M + G_{Md} \frac{d}{dt} \right] \cdot \theta \tag{2.24c}$$

The substitution of Eqs. (2.23a, b) and (2.24a-c) into Eqs. (2.22a, b) completes the longitudinal equations of motion for the THARWP. These equations can be written in matrix form as follows:

$$\underline{M}\,\ddot{\underline{\sigma}} + \underline{D}\,\dot{\underline{\sigma}} + \underline{K}\,\underline{\sigma} = \underline{0} \tag{2.25}$$

where: $\underline{\sigma} = \begin{bmatrix} \zeta \\ \theta \end{bmatrix}$, and

$\underline{M}$, $\underline{D}$, and $\underline{K}$ contain the constant coefficients from Eqs. (2.22a, b) following the above mentioned substitutions.

The method of solving this set of two second-order differential equations is to rewrite the equations as four first-order differential equations by defining the following variable:

$$\underline{\kappa} = \dot{\underline{\sigma}} = \begin{bmatrix} \dot{\zeta} \\ \dot{\theta} \end{bmatrix} \tag{2.26}$$

The combination of Eqs. (2.25) and (2.26) give the system of four first-order differential equations as follows:

$$\begin{bmatrix} \dot{\underline{\sigma}} \\ \dot{\underline{\kappa}} \end{bmatrix} = \left[\begin{array}{c:c} \underline{0} & \underline{I} \\ \hdashline \underline{K'} & \underline{M'} \end{array} \right] \begin{bmatrix} \underline{\sigma} \\ \underline{\kappa} \end{bmatrix} \tag{2.27}$$

where: $\underline{K'} = -\underline{M}^{-1}\underline{K}$,

$\underline{M'} = -\underline{M}^{-1}\underline{D}$

The roots for Eq. (2.27) can now be calculated using traditional eigenvalue solver techniques (Ref. 19). These techniques were implemented in the computer program called jdd_meth.cpp, a listing of which is given in Appendix 11.

The lateral problem is defined by the general small perturbation THARWP equations, Eqs. (2.4b, d, f), the cable force, Eq. (2.8b), the auxiliary conditions, Eqs. (2.9b, c), the velocity transformation, Eq. (2.12b), and the coordinate transformation, Eq. (2.17b). First, substitute Eq. (2.12b) into Eqs. (2.4b, d, f) to give:

$$m\ddot{Y} - mU_o\dot{\psi} - mg\phi\mathrm{COS}\theta_o + Y_vU_o\psi - Y_v\dot{Y} - Y_pp - (Y_r - mU_o)r$$

$$-\Delta Y_c = \Delta Y_{cp} \tag{2.28a}$$

$$I_{xx}\dot{p} - I_{xz}\dot{r} + L_vU_o\psi - L_v\dot{Y} - L_pp - L_rr - \Delta L_c = \Delta L_{cp} \tag{2.28b}$$

$$I_{zz}\dot{r} - I_{xz}\dot{p} + N_vU_o\psi - N_v\dot{Y} - N_pp - N_rr - \Delta N_c = \Delta N_{cp} \tag{2.28c}$$

Now apply the cable force terms, Eq. (2.8b), and the auxiliary conditions, Eqs. (2.9b, c) to Eqs. (2.28a-c) to give the following:

$$\left[(mX_{TP})\frac{d^2}{dt^2} + (N_v - X_{TP}Y_v)\frac{d}{dt}\right]Y + \left[(I_{xz})\frac{d^2}{dt^2} + (N_p - X_{TP}Y_p)\frac{d}{dt}\right.$$

$$\left. - X_{TP}mg\cos\theta_o\right]\phi + \left[(-I_{zz})\frac{d^2}{dt^2} + (N_r - X_{TP}Y_r)\frac{d}{dt} + U_o(X_{TP}Y_v - N_v)\right]\psi$$

$$+\Delta N_c - X_{TP}\Delta Y_c = 0 \tag{2.29a}$$

$$\left[(-mZ_{TP})\frac{d^2}{dt^2} + (L_v + Z_{TP}Y_v)\frac{d}{dt}\right]Y + \left[(-I_{xx})\frac{d^2}{dt^2} + (L_p + Z_{TP}Y_p)\frac{d}{dt}\right.$$

$$\left. + Z_{TP}mg\cos\theta_o\right]\phi + \left[(I_{xz})\frac{d^2}{dt^2} + (L_r + Z_{TP}Y_r)\frac{d}{dt} - U_o(Z_{TP}Y_v + L_v)\right]\psi$$

$$+\Delta L_c + Z_{TP}\Delta Y_c = 0 \tag{2.29b}$$

$$\left[(-m)\frac{d^2}{dt^2} + (Y_v)\frac{d}{dt} - T_o\left(\frac{\partial}{\partial S}\right)\right]Y + \left[(Y_p)\frac{d}{dt} + T_o\overline{\left(\frac{\partial\tilde{Z}}{\partial S}\right)}_a + mg\cos\theta_o\right]\phi$$

$$+\left[(Y_r)\frac{d}{dt} - T_o\overline{\left(\frac{\partial\tilde{X}}{\partial S}\right)}_a - U_oY_v\right]\psi + \Delta Y_c = 0 \tag{2.29c}$$

may be applied to Eqs. (2.29a-c) to give the system of three second-order differential equations that describe the lateral, small perturbation motion of the THARWP:

$$
\begin{aligned}
&\left[(mX_{TP})\frac{d^2}{dt^2}+(N_v-X_{TP}Y_v)\frac{d}{dt}\right]\tilde{Y}(L_n,t)+\left[(I_{xz}+mX_{TP}Z_{TP})\frac{d^2}{dt^2}+(Z_{TP}N_v\right.\\
&\left.-X_{TP}Z_{TP}Y_v+N_p-X_{TP}Y_p)\frac{d}{dt}-X_{TP}mg\cos\theta_o\right]\phi+\left[(-I_{zz}-mX_{TP}^2)\frac{d^2}{dt^2}\right.\\
&\left.-X_{TP}N_v+X_{TP}^2Y_v+N_r-X_{TP}Y_r)\frac{d}{dt}+U_o(X_{TP}Y_v-N_v)\right]\psi\\
&+\Delta N_c-X_{TP}\Delta Y_c=0
\end{aligned}
\qquad (2.30a)
$$

$$
\begin{aligned}
&\left[(-mZ_{TP})\frac{d^2}{dt^2}+(L_v+Z_{TP}Y_v)\frac{d}{dt}\right]\tilde{Y}(L_n,t)+\left[(-mZ_{TP}^2-I_{xx})\frac{d^2}{dt^2}+(Z_{TP}Y_p\right.\\
&\left.L_p+Z_{TP}L_v+Z_{TP}^2Y_v)\frac{d}{dt}+Z_{TP}mg\cos\theta_o\right]\phi+\left[(I_{xz}+mX_{TP}Z_{TP})\frac{d^2}{dt^2}+(L_r+Z_{TP}Y_r\right.\\
&\left.-Z_{TP}L_v-X_{TP}Z_{TP}Y_v)\frac{d}{dt}-U_oL_v-Z_{TP}U_oY_v\right]\psi+\Delta L_c+Z_{TP}\Delta Y_c=0
\end{aligned}
\qquad (30b)
$$

$$
\begin{aligned}
&\left[(-m)\frac{d^2}{dt^2}+(Y_v)\frac{d}{dt}-T_o\left(\frac{\partial}{\partial S}\right)\right]\tilde{Y}(L_n,t)+\left[(-mZ_{TP})\frac{d^2}{dt^2}+(Z_{TP}Y_v+Y_p)\frac{d}{dt}\right.\\
&\left.+T_o\overline{\left(\frac{\partial\tilde{Z}}{\partial S}\right)_a}+mg\cos\theta_o\right]\phi+\left[(mX_{TP})\frac{d^2}{dt^2}+(Y_r-X_{TP}Y_v)\frac{d}{dt}-T_o\overline{\left(\frac{\partial\tilde{X}}{\partial S}\right)_a}\right.\\
&\left.-U_oY_v\right]\psi+\Delta Y_c=0
\end{aligned}
\qquad (2.30c)
$$

As with the longitudinal solution, the control expressions must be defined in terms of the appropriate variables and substituted into the three above equations. The control expressions can be written as:

$$
\Delta Y_c=\left[G_Y+G_{Yd}\frac{d}{dt}\right]\cdot\tilde{Y}(L_n,t)\qquad \Delta L_c=\left[G_L+G_{Ld}\frac{d}{dt}\right]\cdot\phi \qquad (2.31a,b)
$$

$$
\Delta N_c=\left[G_N+G_{Nd}\frac{d}{dt}\right]\cdot\psi \qquad (2.31c)
$$

The substitution of Eqs. (2.51a-c) into Eqs. (2.50a-c) completes the lateral equations of motion for the THARWP. Again, the set of three second-order differential equations were rewritten in matrix form as six first-order differential equations and solved in the same manner as the longitudinal equations. The lateral solution techniques were also implemented in the computer program called jdd_meth.cpp listed in Appendix 11.

Rigid "String" Tether Model

The equations of motion developed in Appendix 2 for the simple THARWP analysis account for the motion of the confluence point and the flight vehicle, excluding the effects of the tether length, shape, mass, or drag . The analysis developed below begins with the linearized general equations of motion from the simple THARWP analysis and includes the effects of the tether length. The tether is modeled as a rigid-link, excluding aerodynamic and inertial effects, that connects the confluence point to the attachment point. This analysis is based on previous work completed for the analysis of tethered aerostats (Ref. 4). As in Ref. 4, only the longitudinal equations of motion will be developed since the purpose of this analysis is to validate the longitudinal portion of the discrete-element THARWP analysis. As shown in Figure 3.1, the rigid string tether, of length L, is positioned relative to the ground at an angle of $\tilde{\alpha} = \tilde{\alpha}_0 + \Delta\tilde{\alpha}$ and the tension within the tether is give by $T = T_0 + \Delta T$, where the $\Delta(\,)$ terms are the small perturbations from equilibrium, $(\,)_0$.

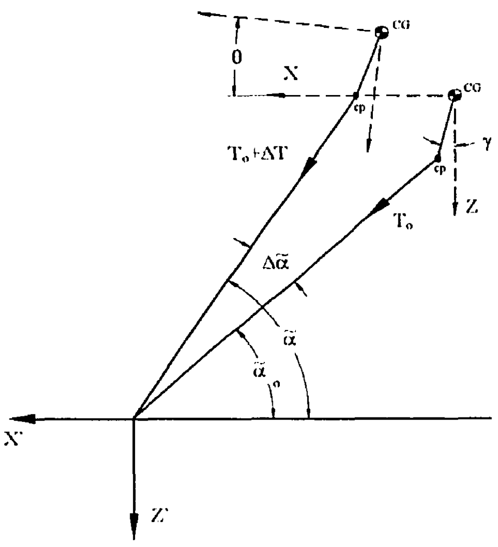


Figure 3.1. Rigid "string" tether model

Recall the longitudinal linearized general equations of motion from Appendix 2 as:

$$mg\theta COS\theta_o + m\dot{u} - X_u u - X_w w - \Delta X_c = \Delta X_{cp} \tag{3.1a}$$

$$mg\theta SIN\theta_o - Z_u u - Z_w w + (m - Z_{\dot{w}})\dot{w} - (mU_o + Z_q)q - \Delta Z_c = \Delta Z_{cp} \tag{3.1b}$$

$$I_{yy}\dot{q} - M_u u - M_w w - M_{\dot{w}}\dot{w} - M_q q - \Delta M_c = \Delta M_{cp} \tag{3.1c}$$

Now, the Δ terms, the velocities, u and w, and the accelerations, $\dot{u}$ and $\dot{w}$, need to be expressed in terms of the flight vehicle pitch angle, θ, the tether angle, $\tilde{\alpha}$, and the tether tension, T. First, the expressions for the forces and the moment acting on the flight vehicle due to the tether tension at the confluence point need to be derived. From inspection of Figure 3.1, the tension forces acting on the confluence point are given by:

$$X'_{cp} = T\cos\tilde{\alpha} \tag{3.2a}$$

$$Z'_{cp} = T\sin\tilde{\alpha} \tag{3.2b}$$

and the resulting forces acting on the flight vehicle's center of mass are given by:

$$X_{cp} = X'_{cp}\cos\theta - Z'_{cp}\sin\theta \tag{3.3a}$$

$$Z_{cp} = Z'_{cp}\cos\theta + X'_{cp}\sin\theta \tag{3.3b}$$

Substituting Eqs. (3.2a, b) into Eqs. (3.3a, b) and retaining only the small perturbation quantities gives the following expressions for the tension forces acting on the flight vehicle's CG:

$$\Delta X_{cp} = \Delta T\cos\tilde{\alpha}_o - \Delta\tilde{\alpha}\, T_o\sin\tilde{\alpha}_o - \theta\, T_o\sin\tilde{\alpha}_o \tag{3.4a}$$

$$\Delta Z_{cp} = \Delta T\sin\tilde{\alpha}_o + \Delta\tilde{\alpha}\, T_o\cos\tilde{\alpha}_o + \theta\, T_o\cos\tilde{\alpha}_o \tag{3.4b}$$

From observation of Figure 3.1, the pitching moment acting on the flight vehicle can be written simply as:

$$\Delta M_{cp} = \Delta X_{cp} Z_{TP} - \Delta Z_{cp} X_{TP} \tag{3.4c}$$

The velocities and accelerations of the flight vehicle, relative to the attachment point, are determined by calculating the first and second time derivative of the expressions describing written as follows:

$$X'_{cg} = -L\cos\tilde{\alpha} - R_{TP}\sin(\gamma + \theta)$$

$$Z'_{cg} = -L\sin\tilde{\alpha} - R_{TP}\cos(\gamma + \theta)$$

where: L is the tether length, and

R_{TP} is the distance from the flight vehicle's CG to the confluence point.

Assuming only small perturbations occur and subtracting the equilibrium quantities, the above equations become:

$$\Delta X'_{cg} = \Delta\tilde{\alpha}\, L\sin\tilde{\alpha}_o - \theta Z_{TP} \tag{3.5a}$$

$$\Delta Z'_{cg} = -\Delta\tilde{\alpha}\, L\cos\tilde{\alpha}_o + \theta X_{TP} \tag{3.5b}$$

Noting that: $X_{TP} = R_{TP}\sin\gamma$ and

$Z_{TP} = R_{TP}\cos\gamma$

The velocities and accelerations are then:

$$u = \dot{\tilde{\alpha}}\, L\sin\tilde{\alpha}_o - \dot{\theta} Z_{TP} \tag{3.6a}$$

$$w = -\dot{\tilde{\alpha}}\, L\cos\tilde{\alpha}_o + \dot{\theta} X_{TP} \tag{3.6b}$$

$$\dot{u} = \ddot{\tilde{\alpha}}\, L\sin\tilde{\alpha}_o - \ddot{\theta} Z_{TP} \tag{3.6c}$$

$$\dot{w} = -\ddot{\tilde{\alpha}}\, L\cos\tilde{\alpha}_o + \ddot{\theta} X_{TP} \tag{3.6d}$$

Finally, the control terms in Eqs. (3.1a-c) are given by:

$$\Delta X_c = \left[G_X + G_{Xd}\frac{d}{dt}\right]\Delta X'_{cg} = \left[G_X + G_{Xd}\frac{d}{dt}\right]\left(\Delta\tilde{\alpha}\, L\sin\tilde{\alpha}_o - Z_{TP}\theta\right) \tag{3.7a}$$

$$\Delta Z_c = \left[G_Z + G_{Zd}\frac{d}{dt}\right]\Delta Z'_{cg} = \left[G_Z + G_{Zd}\frac{d}{dt}\right]\left(-\Delta\tilde{\alpha}\, L\cos\tilde{\alpha}_o + \theta X_{TP}\right) \tag{3.7b}$$

$$\Delta M_c = \left[G_M + G_{Md}\frac{d}{dt}\right]\theta \tag{3.7c}$$

linearized equations of motion for the THARWP with a rigid string tether. The complete equations are given as follows:

$$\Big[(m L \sin\tilde{\alpha}_o)\frac{d^2}{dt^2} + (X_w L\cos\tilde{\alpha}_o - X_u L\sin\tilde{\alpha}_o - G_{Xd} L\sin\tilde{\alpha}_o)\frac{d}{dt}$$

$$+ (T_o \sin\tilde{\alpha}_o - G_X L\sin\tilde{\alpha}_o)\Big]\tilde{\alpha} + \Big[(-m Z_{TP})\frac{d^2}{dt^2} + (X_u Z_{TP} - X_w X_{TP} + Z_{TP} G_{Xd})\frac{d}{dt}$$

$$+ (mg\cos\theta_o + T_o \sin\tilde{\alpha}_o + Z_{TP} G_X)\Big]\theta + \big[-\cos\tilde{\alpha}_o\big]\Delta T = 0 \qquad (3.8a)$$

$$\Big[(Z_{\dot{w}} - m)L\cos\tilde{\alpha}_o \frac{d^2}{dt^2} + (Z_w L\cos\tilde{\alpha}_o - Z_u L\sin\tilde{\alpha}_o + G_{Zd} L\cos\tilde{\alpha}_o)\frac{d}{dt}$$

$$+ (-T_o\cos\tilde{\alpha}_o + G_Z L\cos\tilde{\alpha}_o)\Big]\tilde{\alpha} + \Big[(m X_{TP} - X_{TP} Z_{\dot{w}})\frac{d^2}{dt^2} + (Z_u Z_{TP} - Z_w X_{TP}$$

$$- mU_o - Z_q - X_{TP} G_{Zd})\frac{d}{dt} + (mg\sin\theta_o - T_o\cos\tilde{\alpha}_o - X_{TP} G_Z)\Big]\theta$$

$$+ \big[-\sin\tilde{\alpha}_o\big]\Delta T = 0 \qquad (3.8b)$$

$$\Big[(M_{\dot{w}} L\cos\tilde{\alpha}_o)\frac{d^2}{dt^2} + (M_w L\cos\tilde{\alpha}_o - M_u L\sin\tilde{\alpha}_o)\frac{d}{dt} + (X_{TP}\cos\tilde{\alpha}_o$$

$$+ Z_{TP}\sin\tilde{\alpha}_o)\Big]\tilde{\alpha} + \Big[(I_{yy} - M_{\dot{w}} X_{TP})\frac{d^2}{dt^2} + (M_u Z_{TP} - M_w X_{TP} - M_q - G_{Md})\frac{d}{dt}$$

$$+ (T_o X_{TP}\cos\tilde{\alpha}_o + T_o Z_{TP}\sin\tilde{\alpha}_o - G_M)\Big]\theta$$

$$+ (X_{TP}\sin\tilde{\alpha}_o - Z_{TP}\cos\tilde{\alpha}_o)\Delta T = 0 \qquad (3.8c)$$

Equations (3.8a-c) can be written in matrix form as follows:

$$\underline{M}\,\ddot{\underline{\sigma}} + \underline{D}\,\dot{\underline{\sigma}} + \underline{K}\,\underline{\sigma} = \underline{0} \qquad (3.9)$$

where: $\underline{\sigma} = \begin{bmatrix} \tilde{\alpha} \\ \theta \\ \Delta T \end{bmatrix}$, and

$\underline{M}$, $\underline{D}$, and $\underline{K}$ contain constant coefficients from Eqs. (3.8a-c) solution is of the following form:

$$\underline{\sigma} = \begin{bmatrix} \tilde{\alpha} \\ \theta \\ \Delta T \end{bmatrix} = \begin{bmatrix} A_1 e^{\lambda t} \\ A_2 e^{\lambda t} \\ A_3 e^{\lambda t} \end{bmatrix}$$

Substituting this expression, along with its first and second derivatives, into Eq. (3.9) gives a fourth-order polynomial characteristic equation. Note, the characteristic equation is only fourth-order since there are no $\Delta\dot{T}$ or $\Delta\ddot{T}$ terms in Eqs. (3.8a-c). The roots for Eq. (3.9) can now be calculated using traditional root finding techniques (Ref. 19). These techniques were implemented in the computer program called jdd_strg.cpp which is listed in Appendix 12.

APPENDIX 4: Development of the HARWP Equations of Motion using the Modified Rigid "String" Tether Model

As a further enhancement to the equations developed in Appendix 3, the complexity of the tether model will be increased by including general aerodynamic and inertial effects (Ref. 4). As with the rigid string tether model developed in Appendix 3, the analysis developed below uses the linearized general equations of motion developed in Appendix 2 and then includes the aerodynamic and inertial forces acting on the tether. The tether model, shown in Figure 4.1, is assumed to have a general curvature and the wind velocity, U_o, is assumed constant over the length of the tether. If the tether remains nearly straight during its motion, the velocities and accelerations along the tether can be approximated by the following:

$$\dot{\zeta}(S,t) \approx S\dot{\tilde{\alpha}}(t) \qquad \ddot{\zeta}(S,t) \approx S\ddot{\tilde{\alpha}}(t) \tag{4.1a,b}$$

where: S is the position along the tether, thus,

S = 0 for the attachment point, and

S = L for the confluence point location.

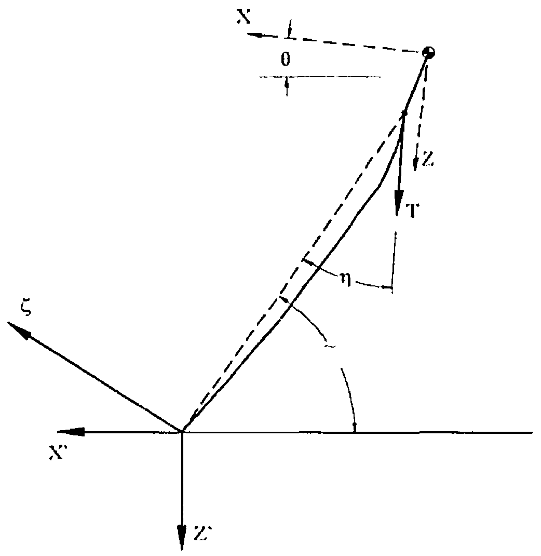


Figure 4.1. Modified rigid "string" tether model

Using Eqs. (4.1a, b) and the small perturbation assumption, the drag force acting at the position S along the tether can be written as:

$$f'_D(S,t) = 2K\rho R U_o \sin\tilde{\alpha}\, S\dot{\tilde{\alpha}}(t) \tag{4.2a}$$

where: K is the tether cross-flow coefficient of drag,

ρ is the average air density over the length of the tether, and

R is the tether radius.

Also, the inertial force acting at the position S along the tether can be given by:

$$f_I(S,t) = \rho_t\ddot{\zeta}(S,t) = \rho_t S\ddot{\tilde{\alpha}}(t) \tag{4.2b}$$

where: ρ_t is the mass per unit length of the tether

Therefore, the total forces acting on the tether due to aerodynamic and inertial effects are calculated by integrating Eqs. (4.2a, b) over the length of the tether. These total forces are given by:

$$F'_D = \int_0^L f'_D\, ds = K\rho R U_o L^2 \sin\tilde{\alpha}\,\dot{\tilde{\alpha}} \tag{4.3a}$$

$$F_I = \int_0^L f_I\, ds = \tfrac{1}{2}\rho_t L^2 \ddot{\tilde{\alpha}} \tag{4.3b}$$

The drag and inertial forces described in Eqs. (4.3a, b) vary linearly from zero at the tether attachment point to their maximum value at the confluence point. Since these force distributions are triangular in nature, the total force distribution can be approximated by applying 2/3 of the total force, Eqs. (4.3a, b), at the confluence point as follows:

$$T\eta = \tfrac{2}{3} K\rho R U_o L^2 \sin\tilde{\alpha}\,\dot{\tilde{\alpha}} + \tfrac{1}{3}\rho_t L^2 \ddot{\tilde{\alpha}} \tag{4.4}$$

Assuming only small perturbations occur, Eq. (4.4) can be written in terms of the flight vehicle's stability axes:

$$T\eta_X = -\tfrac{2}{3} K\rho R U_o L^2 \sin^2\tilde{\alpha}_o\,\dot{\tilde{\alpha}} - \tfrac{1}{3}\rho_t L^2 \sin\tilde{\alpha}_o\,\ddot{\tilde{\alpha}} \tag{4.5a}$$

$$T\eta_Z = \tfrac{2}{3} K\rho R U_o L^2 \sin\tilde{\alpha}_o \cos\tilde{\alpha}_o\,\dot{\tilde{\alpha}} + \tfrac{1}{3}\rho_t L^2 \cos\tilde{\alpha}_o\,\ddot{\tilde{\alpha}} \tag{4.5b}$$

The forces in Eqs. (4.5a, b) are now added to the confluence point forces developed in Appendix 3. Also, the velocities, accelerations, and control terms can be taken directly from the analysis developed in Appendix 3. Finally, the longitudinal linearized equations of motion become:

$$\Big[\left(m\,L+\tfrac{1}{3}\rho_t L^2\right)\sin\tilde{\alpha}_o\frac{d^2}{dt^2}+\big(X_w L\cos\tilde{\alpha}_o - X_u L\sin\tilde{\alpha}_o - G_{Xd} L\sin\tilde{\alpha}_o$$

$$\tfrac{2}{3}K\rho R U_o L^2\sin^2\tilde{\alpha}_o\big)\frac{d}{dt}+\left(T_o\sin\tilde{\alpha}_o - G_X L\sin\tilde{\alpha}_o\right)\Big]\tilde{\alpha}+\Big[\left(-\,m\,Z_{TP}\right)\frac{d^2}{dt^2}$$

$$+\left(X_u Z_{TP}-X_w X_{TP}+Z_{TP}G_{Xd}\right)\frac{d}{dt}+\left(mg\cos\theta_o+T_o\sin\tilde{\alpha}_o+Z_{TP}G_X\right)\Big]\theta$$

$$+\left[-\cos\tilde{\alpha}_o\right]\Delta T=0 \qquad (4.6a)$$

$$\Big[\left(Z_{\dot{w}}-m-\tfrac{1}{3}\rho_t L\right)L\cos\tilde{\alpha}_o\frac{d^2}{dt^2}+\big(Z_w L\cos\tilde{\alpha}_o - Z_u L\sin\tilde{\alpha}_o + G_{Zd} L\cos\tilde{\alpha}_o$$

$$-\tfrac{2}{3}K\rho R U_o L^2\sin\tilde{\alpha}_o\cos\tilde{\alpha}_o\big)\frac{d}{dt}+\left(-\,T_o\cos\tilde{\alpha}_o+G_Z L\cos\tilde{\alpha}_o\right)\Big]\tilde{\alpha}$$

$$+\Big[\left(m\,X_{TP}-X_{TP}Z_{\dot{w}}\right)\frac{d^2}{dt^2}+\left(Z_u Z_{TP}-Z_w X_{TP}-mU_o-Z_q-X_{TP}G_{Zd}\right)\frac{d}{dt}$$

$$+\left(mg\sin\theta_o-T_o\cos\tilde{\alpha}_o-X_{TP}G_Z\right)\Big]\theta+\left[-\sin\tilde{\alpha}_o\right]\Delta T=0 \qquad (4.6b)$$

$$\Big[\left(M_{\dot{w}}L\cos\tilde{\alpha}_o+\tfrac{1}{3}Z_{TP}\rho_t L^2\sin\tilde{\alpha}_o+\tfrac{1}{3}X_{TP}\rho_t L^2\cos\tilde{\alpha}_o\right)\frac{d^2}{dt^2}+\big(M_w L\cos\tilde{\alpha}_o$$

$$-M_u L\sin\tilde{\alpha}_o+\tfrac{2}{3}K\rho R U_o L^2\sin\tilde{\alpha}_o\left(Z_{TP}\sin\tilde{\alpha}_o+X_{TP}\cos\tilde{\alpha}_o\right)\big)\frac{d}{dt}+\big(X_{TP}\cos\tilde{\alpha}_o$$

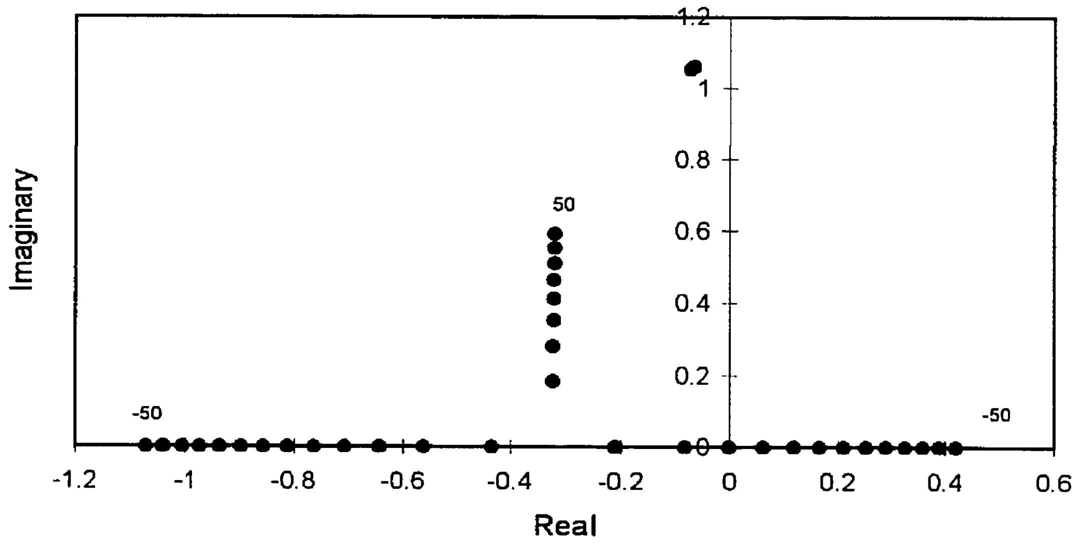
$$+Z_{TP}\sin\tilde{\alpha}_o\big)\Big]\tilde{\alpha}+\Big[\left(I_{yy}-M_{\dot{w}}X_{TP}\right)\frac{d^2}{dt^2}+\left(M_u Z_{TP}-M_w X_{TP}-M_q-G_{Md}\right)\frac{d}{dt}$$

$$+\left(T_o X_{TP}\cos\tilde{\alpha}_o+T_o Z_{TP}\sin\tilde{\alpha}_o-G_M\right)\Big]\theta$$

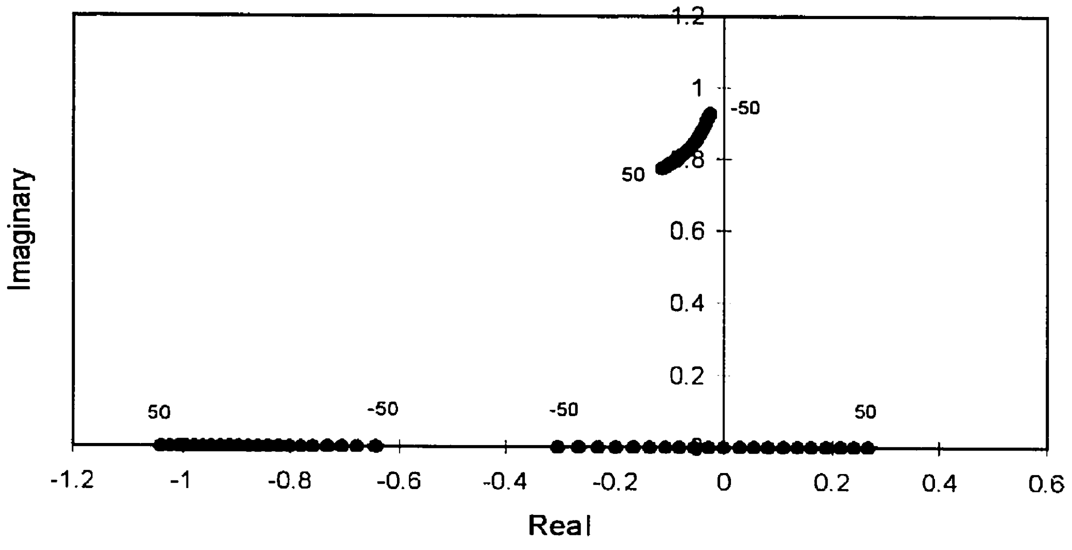
$$+\left(X_{TP}\sin\tilde{\alpha}_o-Z_{TP}\cos\tilde{\alpha}_o\right)\Delta T=0 \qquad (4.6c)$$

These methods are implemented in the computer program called jdd_mass.cpp which is listed in Appendix 13.

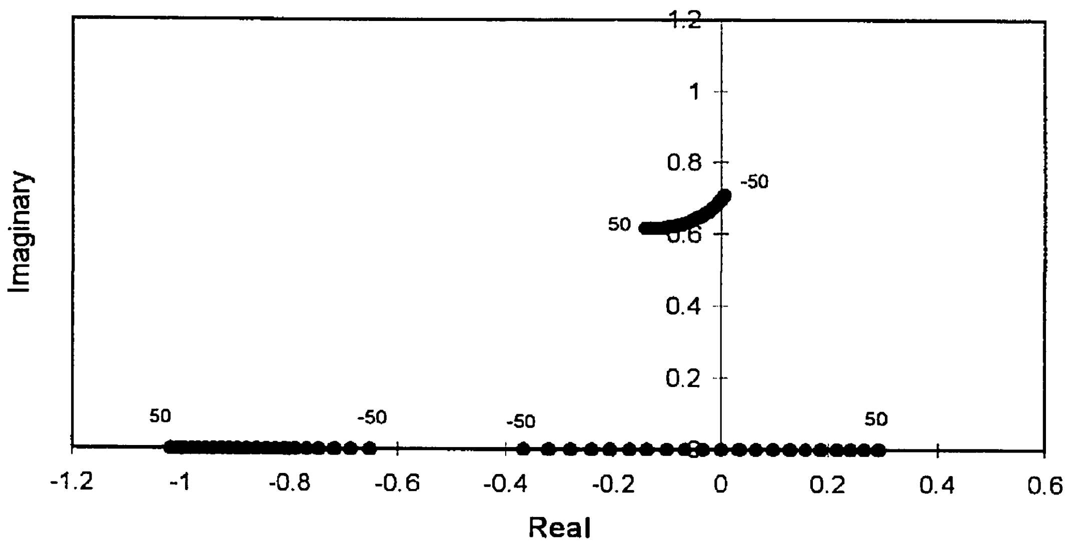
Simple Tether Model
X Gain Varied



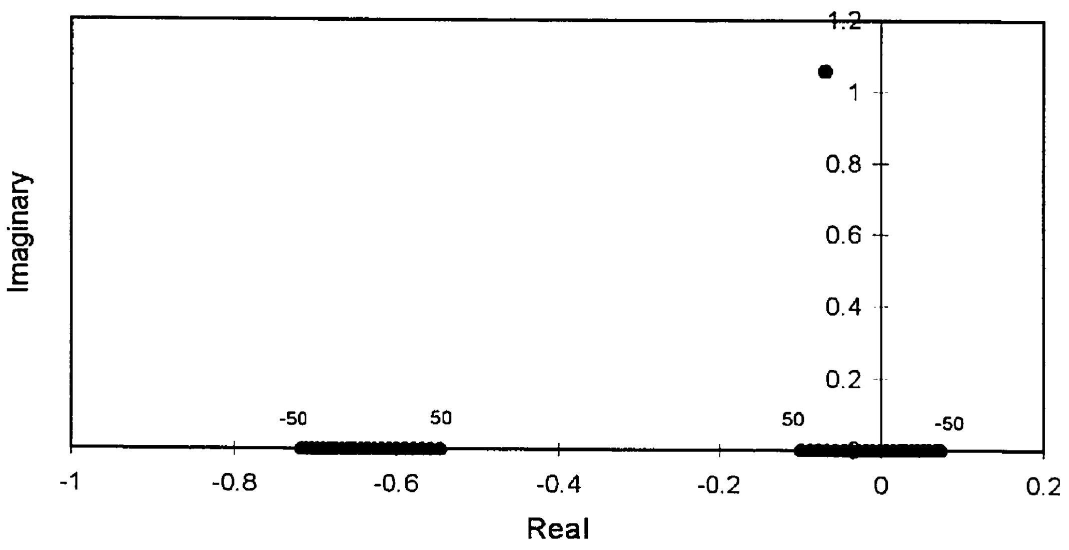
String Tether Model
X Gain Varied



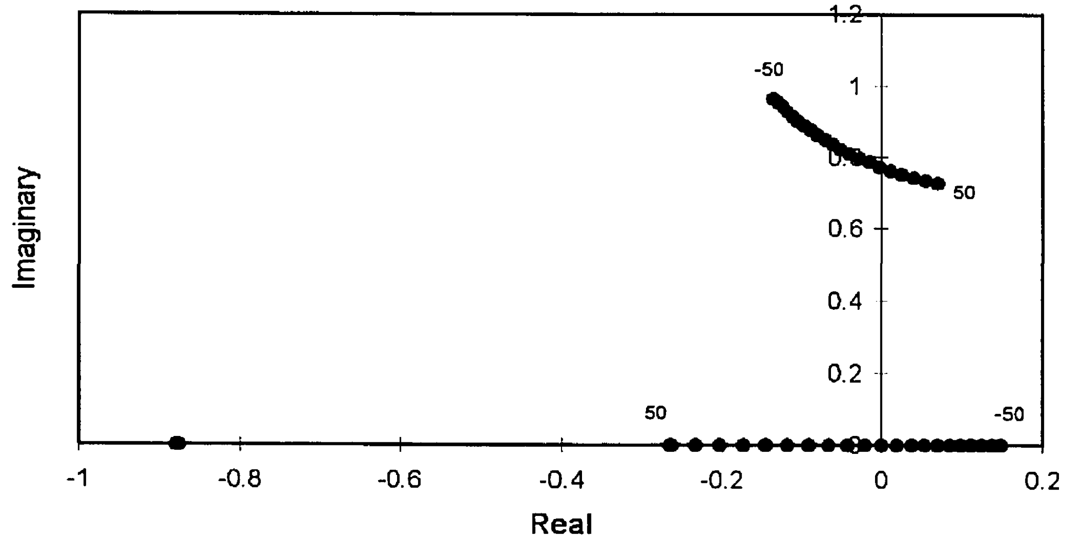
Modified String Tether Model
X Gain Varied



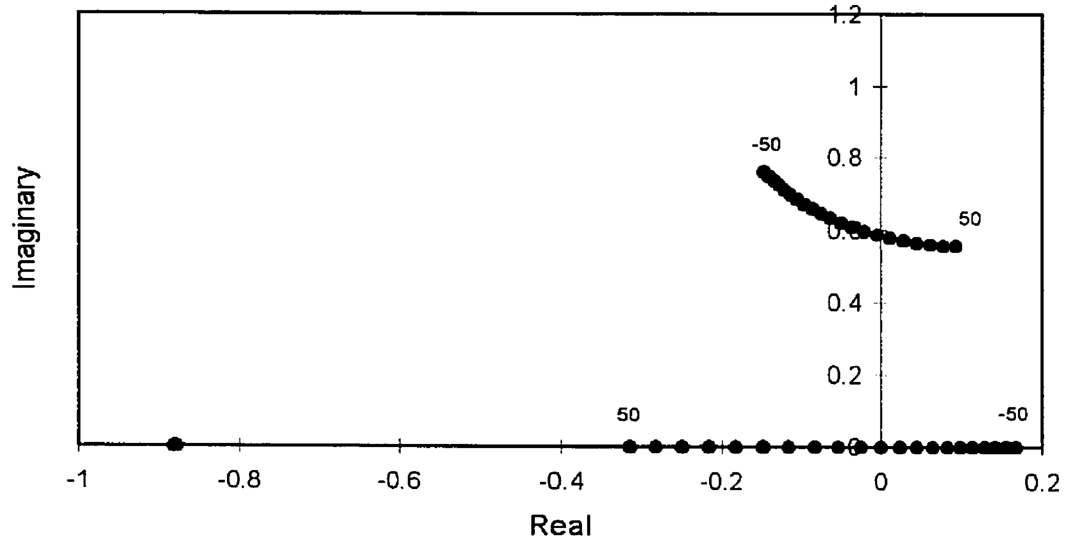
Simple Tether Model
Z Gain Varied



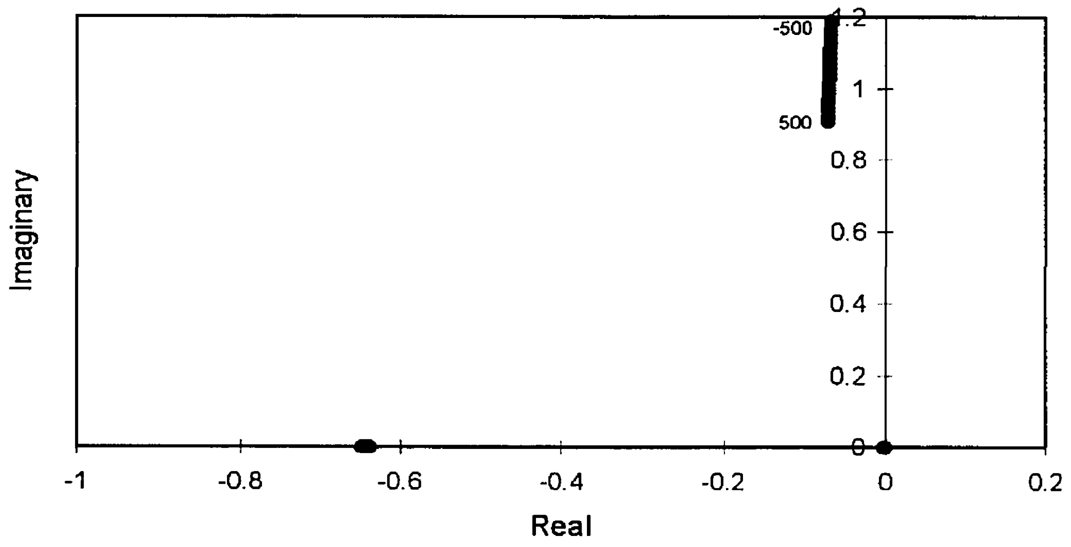
String Tether Model
Z Gain Varied



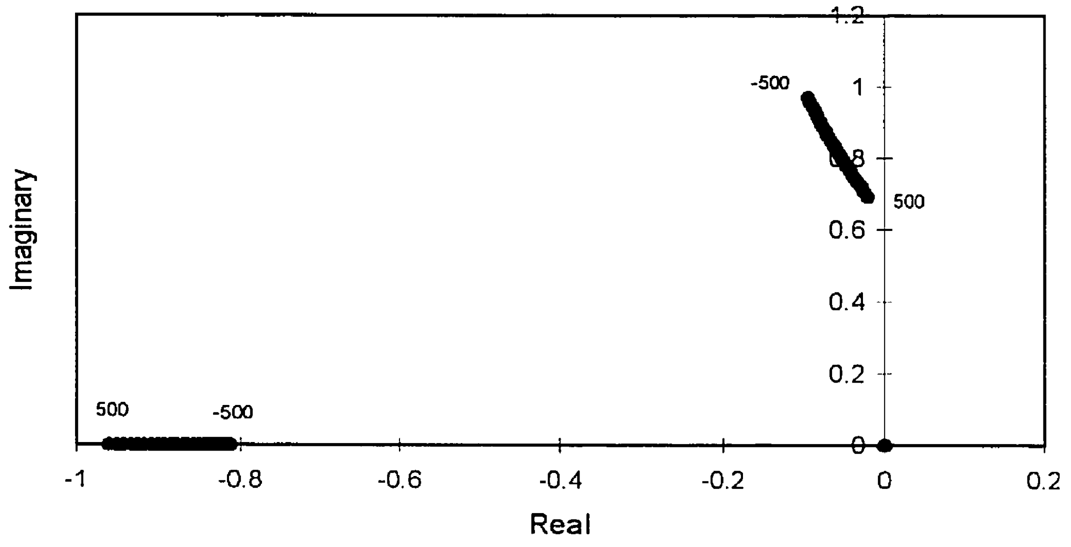
Modified String Tether Model
Z Gain Varied



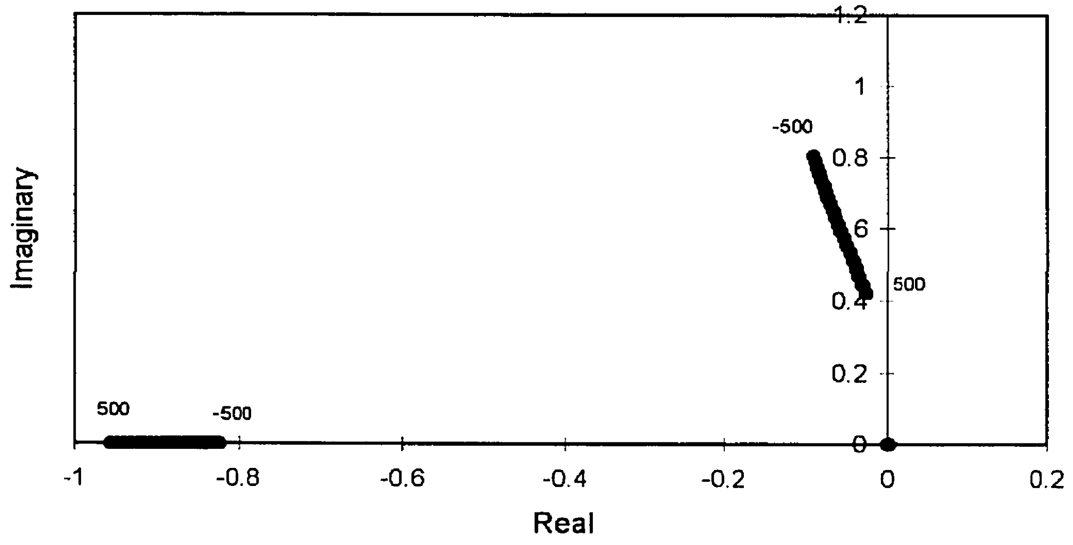
Simple Tether Model
Pitch Gain Varied



String Tether Model
Pitch Gain Varied



Modified String Tether Model
Pitch Gain Varied



Development of the THARWP Non-Dimensional Stability Derivatives

The discrete-element analysis of the THARWP combines the linearized first-order equations of motion developed for airplanes in Ref. 6 and 7 (used here to describe the motions of the flight vehicle) with the tether equations of motion formulated using a discrete-element tether model. In order to use the linearized first-order equations of motion to describe the dynamics of the flight vehicle, the non-dimensional stability derivatives, which are an integral part of these equations, must be defined for the flight vehicle. These non-dimensional stability derivatives will be estimated using a combination of the methods established previously in Ref. 6, 7, 15, and 16, and are presented below, organized in terms of the longitudinal and lateral non-dimensional stability derivatives. For reference, the following list contains frequently used subscripts and superscripts, and their meanings:

$(\ )_{ls}$	lifting surface or wing,	$(\ )_{ht}$	horizontal stabilizer,
$(\ )_{vt}$	vertical stabilizer,	$(\ )_{fuse}$	fuselage,
$(\ )^{ac}$	relating to the aerodynamic center of the flight vehicle		

The u Derivatives

$C_{X_u} = C_{Z_u} = C_{M_u} = 0$ since these are significant only in the following instances (which do not apply to the THARWP):

1. transonic speeds;
2. transitional Reynolds Number; and
3. extremely low Reynolds Number.

The α Derivatives

$$C_{X_\alpha} = C_{L_o} - \left(C_{D_\alpha}\right)_o$$

$$\left(C_{D_\alpha}\right)_o = \frac{2C_{L_o}}{\pi \underbrace{AR}_{\substack{\text{Aspect}\\ \text{Ratio}}} e} - \left(C_{L_\alpha}\right)_o$$

where: C_{L_o} is the trim-state coefficient of lift for the flight vehicle;

$\left(C_{L_\alpha}\right)_o$ is the trim-state lift-curve slope; and

e is the Oswald's efficiency factor for the flight vehicle.

$$C_{Z_\alpha} = -\left(C_{L_\alpha}\right)_o - C_{D_o}$$

where: C_{D_o} is the trim-state coefficient of drag for the flight vehicle

$$C_{M_\alpha} = \frac{C_{X_\alpha}\left(Z_{cg} - Z_{np}\right)}{\bar{c}} - \frac{C_{Z_\alpha}\left(X_{cg} - X_{np}\right)}{\bar{c}}$$

where: X_{cg} and Z_{cg} are the horizontal and vertical positions of the flight vehicle's center of gravity (CG) and

X_{np} and Z_{np} are the horizontal and vertical positions of the flight vehicle neutral point (estimated by using traditional stability theory) nose of the flight vehicle fuselage.

The q Derivatives

$$C_{X_q} = \left(C_{X_q}\right)_{ls}$$

$$\left(C_{X_q}\right)_{ls} = -\left(C_{X_\alpha}\right)_{ls}\left[\frac{2\left(X_{cg} - X_{ls}\right)}{\bar{c}} + \left(\frac{\partial\varepsilon}{\partial\hat{q}}\right)_{ls}\right]\eta_{ls} + \left(C_{X_q}\right)_{ls}^{ac}\eta_{ls} \quad \text{(Ref. 16)}$$

$$\left(C_{X_\alpha}\right)_{ls} = \left[\left(C_L\right)_{ls_0} - \left(C_{D_\alpha}\right)_{ls_0}\right]\left(\frac{S_{ls}}{S}\right)$$

$$\left(C_{X_q}\right)_{ls}^{ac} = \left(C_{X_\alpha}\right)_{ls}\left(\frac{c_{ls}}{\bar{c}}\right)$$

where: X_{ls} is the horizontal position of the aerodynamic center (AC) of the wing; S and $\bar{c}$ are the reference area and chord, respectively; η_{ls} is the ratio of the dynamic pressure at the lifting surface with the free stream dynamic pressure; and $\left(\frac{\partial\varepsilon}{\partial\hat{q}}\right)_{ls}$ the rate of change of the lifting surface downwash with respect to changes in the non-dimensional pitching rate of the flight vehicle (this term is always zero for the THARWP flight vehicle since there is no surface upstream of the wing to cause any downwash at the wing).

$$C_{Z_q} = \left(C_{Z_q}\right)_{ls} + \left(C_{Z_q}\right)_{ht}$$

$$\left(C_{Z_q}\right)_{ls} = -\left(C_{Z_\alpha}\right)_{ls}\left[\frac{2\left(X_{cg} - X_{ls}\right)}{\bar{c}} + \left(\frac{\partial\varepsilon}{\partial\hat{q}}\right)_{ls}\right]\eta_{ls} + \left(C_{Z_q}\right)_{ls}^{ac}\eta_{ls} \quad \text{(Ref. 16)}$$

$$\left(C_{Z_\alpha}\right)_{ls} = -\left[\left(C_{L_\alpha}\right)_{ls_0} + \left(C_D\right)_{ls_0}\right]\left(\frac{S_{ls}}{S}\right)$$

$$\left(C_{Z_q}\right)_{ls} = \left(C_{Z_\alpha}\right)_{ls}\left(\frac{\quad}{\bar{c}}\right)$$

$$\left(C_{Z_q}\right)_{ht} = -2a_{ht}\left(\frac{S_{ht}\ell_{ht}}{S\bar{c}}\right)$$

where: a_{ht} is the lift curve slope for the horizontal stabilizer and

ℓ_{ht} is the horizontal tail moment arm (distance from the flight vehicle's CG to the AC of the horizontal stabilizer).

$$C_{M_q} = \left(C_{M_q}\right)_{ls} + \left(C_{M_q}\right)_{ht}$$

$$\left(C_{M_q}\right)_{ls} = \left(C_{X_q}\right)_{ls}\frac{Z_{cg} - Z_{ls}}{\bar{c}} - \left(C_{Z_q}\right)_{ls}\frac{X_{cg} - X_{ls}}{\bar{c}} + \left(C_{M_q}\right)_{ls}^{ac}$$

$$\left(C_{M_q}\right)_{ls}^{ac} = -\frac{\pi}{4}\left(\frac{c_{ls}}{\bar{c}}\right)^2\left(\frac{S_{ls}}{S}\right)\left[\frac{1}{3}\left(\frac{AR^3\tan^3\Lambda}{AR + 6\cos\Lambda}\right) + 1\right]\cos\Lambda \quad \text{(Ref. 16)}$$

$$\left(C_{M_q}\right)_{ht} = -2a_{ht}\left(\frac{S_{ht}\ell_{ht}^2}{S\bar{c}^2}\right)$$

where: Λ is the sweep angle of the lifting surface ac line

The $\dot{\alpha}$ Derivatives

$$C_{Z_{\dot\alpha}} = \left(C_{Z_{\dot\alpha}}\right)_{fuse} + \left(C_{Z_{\dot\alpha}}\right)_{ht}$$

$$\left(C_{Z_{\dot\alpha}}\right)_{fuse} = -\frac{2K_3(Vol)_{fuse}}{S\bar{c}}$$

$$\left(C_{Z_{\dot\alpha}}\right)_{ht} = -2\left(C_{L_\alpha}\right)_{ht}V_H\frac{\partial\varepsilon}{\partial\alpha}$$

$$\frac{\partial\varepsilon}{\partial\alpha} = 4.44\left[K_\Lambda K_\lambda K_H\left(\cos\Lambda_{c/4}\right)^{1/2}\right]^{1.19} \quad \text{(Ref. 7)}$$

$$K_\Lambda = \frac{1}{AR} - \frac{1}{1 + AR^{1.7}} \qquad\qquad K_\lambda = \frac{10 - 3\lambda}{7}$$

$$K_H = \frac{1 - \left|\frac{\quad}{b}\right|}{\left(\frac{2\ell_{ht}}{b}\right)^{1/3}}$$

where: K_3 is a function of the fuselage fineness ratio (set to K_3=0.75 for a reasonably streamlined fuselage);

$(Vol)_{fuse}$ is the enclosed volume of the fuselage;

$V_H = \frac{S_{ht}\ell_{ht}}{S\bar{c}}$ is the horizontal tail volume coefficient for the flight vehicle;

$\Lambda_{c/4}$ is the sweep angle of the lifting surface quarter-chord line;

λ is the taper ratio for the lifting wing of the flight vehicle (ratio of the root chord to the tip chord); and

b is the wing span.

$$C_{M_{\dot{\alpha}}} = \left(C_{M_{\dot{\alpha}}}\right)_{fuse} + \left(C_{M_{\dot{\alpha}}}\right)_{ls} + \left(C_{M_{\dot{\alpha}}}\right)_{ht}$$

$$\left(C_{M_{\dot{\alpha}}}\right)_{fuse} = -\frac{\left(X_{cg} - X_{cv}\right)}{\bar{c}}\left(C_{Z_{\dot{\alpha}}}\right)_{fuse}$$

$$\left(C_{M_{\dot{\alpha}}}\right)_{ls} = -\frac{\left(X_{cg} - X_{ls}\right)}{\bar{c}}\left(C_{Z_{\dot{\alpha}}}\right)_{ls}$$

$$\left(C_{M_{\dot{\alpha}}}\right)_{ht} = -\ell_{ht}\left(C_{Z_{\dot{\alpha}}}\right)_{ht}$$

where: X_{cv} is the horizontal position of the volumetric center of the fuselage

The β Derivatives

$$C_{Y_\beta} = \left(C_{Y_\beta}\right)_{vt} + \left(C_{Y_\beta}\right)_{fuse} + \left(C_{Y_\beta}\right)_{ls}$$

$$\left(C_{Y_\beta}\right)_{vt} = \eta_{vt}\left(1 - \frac{\partial\sigma}{\partial\beta}\right)\left(C_{Y_\beta}\right)_{vt}^{*} \quad \text{(Ref. 16)}$$

$$\left(C_{Y_\beta}\right)_{fuse} = -K_i\left(C_{L_\alpha}\right)_{fuse}\left(\frac{(\mathrm{Vol})^{2/3}}{S}\right) \quad \text{(Ref. 16)}$$

$$\left(C_{Y_\beta}\right)_{ls} = -0.75\left(C_{L_\alpha}\right)_{ls}\frac{S_{ls}}{S}\sin^2\Gamma$$

$$\left(C_{Y_\beta}\right)_{vt}^{*} = -\left[\left(C_{L_\alpha}\right)_{vt} + \left(C_{D_o}\right)_{vt}\right]\frac{S_{vt}}{S}$$

where: η_{vt} is the ratio of the lateral dynamic pressure at the vertical stabilizer to the free stream lateral dynamic pressure;

$\frac{\partial\sigma}{\partial\beta}$ is the sidewash factor (approximately zero for a large aspect ratio wing and a narrow fuselage);

K_i is based on the position of the wing relative to the fuselage (Ref. 16);

$K_i = 2.0$ for a high-wing configuration

$K_i = 1.0$ for a mid-wing configuration

$K_i = 1.5$ for a low-wing configuration

$\left(C_{L_\alpha}\right)_{fuse}$ is the lateral lift-curve slope for the fuselage; (Ref. 16)

$\left(C_{L_\alpha}\right)_{fuse} = 0.0525$ /rad for rounded fuselages

$= 0.1243$ for rectangular fuselages

$\left(C_{L_\alpha}\right)_{vt}$ is the lateral lift-curve slope of the vertical stabilizer; and

Γ is the wing dihedral angle.

$$C_{L_\beta} = \left(C_{L_\beta}\right)_{vt} + \left(C_{L_\beta}\right)_{fuse} + \left(C_{L_\beta}\right)_{ls}$$

$$\left(C_{L_\beta}\right)_{vt} = -\frac{Z_{vt}}{b}\left(C_{Y_\beta}\right)_{vt}$$

$$\left(C_{L_\beta}\right)_{fuse} = -\frac{\left(Z_{cp}\right)_{fuse}}{b}\left(C_{Y_\beta}\right)_{fuse}$$

$$\left(C_{L_\beta}\right)_{ls} = -\tfrac{1}{4}\Gamma\left(C_{L_o}\right)_{ls}$$

where: Z_{vt} is the height of the vertical stabilizer's AC relative to the flight vehicle's CG;

$\left(Z_{cp}\right)_{fuse}$ is the height of the fuselage center of pressure relative to the flight vehicle's CG; and

$\left(C_{L_o}\right)_{ls}$ is the trim-state coefficient of lift for the wing.

$$C_{N_\beta} = \left(C_{N_\beta}\right)_{vt} + \left(C_{N_\beta}\right)_{fuse} + \left(C_{N_\beta}\right)_{ls}$$

$$\left(C_{N_\beta}\right)_{vt} = \frac{\ell_{vt}}{b}\left(C_{Y_\beta}\right)_{vt}$$

$$\left(C_{N_\beta}\right)_{fuse} = -0.96K_\beta\left(\frac{S_{side}}{S}\right)\left(\frac{\ell_{fuse}}{b}\right)\left(\frac{h_1}{h_2}\right)^{1/2}\left(\frac{w_2}{w_1}\right)^{1/3} \quad \text{(Ref. 16)}$$

$$K_\beta = 0.257\left(\frac{X_{cg}}{\ell_{fuse}}\right) + .0027\left(\frac{\ell_{fuse}}{h_{max}}\right)^2 - 0.06\left(\frac{\ell_{fuse}}{h_{max}}\right) + 0.307$$

$$\left(C_{N_\beta}\right)_{ls} = \frac{X_{ac}}{b}\left(C_{Y_\beta}\right)_{ls} + \left(C_{N_\beta}\right)_{ls}^{ac}$$

$$\left(C_{N_\beta}\right)_{ls}^{ac} = \left(\frac{C_{N_\beta}}{C_L^2}\right)\left(C_{L_o}\right)_{ls}^2 \frac{S_{ls}}{S}\frac{b_{ls}}{b}$$

$$\left(\frac{C_{N_\beta}}{C_L^2}\right) = \frac{1}{57.3\pi AR}\left\{\frac{1}{4} - \frac{\tan\Lambda_{c/4}}{AR + 4\cos\Lambda_{c/4}}\left[\cos\Lambda_{c/4} - \tfrac{1}{2}AR - \frac{AR^2}{8\cos\Lambda_{c/4}}\right.\right.$$
$$\left.\left. + 6\left(h_{n_w} - h_{cg}\right)\frac{\sin\Lambda_{c/4}}{AR}\right]\right\} \quad \text{(Ref. 7)}$$

where: l_{vt} is the horizontal distance between the vertical stabilizer's AC and the flight vehicle's CG;

h_1, w_1, and h_2, w_2 are the heights and widths of the fuselage at ¼ the length and ¾ the length of the fuselage, respectively;

h_{n_w} is the non-dimensional position of the neutral point of the wing referenced to the wing leading edge; and

h_{cg} is the non-dimensional position of the flight vehicle's CG referenced to the wing leading edge.

The p Derivatives

$$C_{Y_p} = \left(C_{Y_p}\right)_{vt} + \left(C_{Y_p}\right)_{fuse} + \left(C_{Y_p}\right)_{ls}$$

$$\left(C_{Y_p}\right)_{vt} = \eta_{vt}\left(-2.2\frac{Z_{vt}}{b} + \frac{\partial\sigma_{vt}}{\partial\hat{p}}\right)\left(C_{Y_\beta}\right)^{*}_{vt} \quad \text{(Ref. 16)}$$

$$\left(C_{Y_p}\right)_{fuse} = -2\frac{\left(Z_{cp}\right)_{fuse}}{b}\left(C_{Y_\beta}\right)_{fuse}$$

$$\left(C_{Y_p}\right)_{ls} = -\left(C_{Y_\beta}\right)_{ls}\left(-\frac{2Z_{ls}}{b} + \frac{\partial\sigma_{ls}}{\partial\hat{p}}\right)\eta_{ls} + \left(C_{Y_p}\right)^{ac}_{ls}$$

$$\left(C_{Y_p}\right)^{ac}_{ls} = \left[\left(\frac{AR + \cos\Lambda}{AR + 4\cos\Lambda}\right)\tan\Lambda + \frac{1}{AR}\right]\frac{b_{ls}}{b}\frac{S_{ls}}{S}\left(C_{L_\alpha}\right)_{ls} \quad \text{(Ref. 16)}$$

where: η_{ls} is the ratio of the lateral dynamic pressure at the wing to the free stream lateral dynamic pressure and

$\frac{\partial\sigma_{ls}}{\partial\hat{p}}$ is the change in sidewash due to a change in the flight vehicle roll rate.

$$C_{L_p} = \left(C_{L_p}\right)_{vt} + \left(C_{L_p}\right)_{fuse} + \left(C_{L_p}\right)_{ls}$$

$$\left(C_{L_p}\right)_{vt} = -1.1\frac{Z_{vt}}{b}\left(C_{Y_p}\right)_{vt}$$

$$\left(C_{L_p}\right)_{fuse} = -\frac{\left(Z_{cp}\right)_{fuse}}{b}\left(C_{Y_p}\right)_{fuse}$$

$$\left(C_{L_p}\right)_{ls} = -\frac{Z_{ls}}{b}\left(C_{Y_p}\right)_{ls} + \left(C_{L_p}\right)_{ls}^{ac}$$

$$\left(C_{L_p}\right)_{ls}^{ac} = \frac{S_{ls}}{S}\left(\frac{b_{ls}}{b}\right)^2\left(C_{L_p}\right)_{planform}$$

where: $\left(C_{L_p}\right)_{planform}$ is dependant on the dihedral and taper ratio of the wing (set to -0.58 for $\Lambda = 0$ and $\lambda = 1.0$)

$$C_{N_p} = \left(C_{N_p}\right)_{vt} + \left(C_{N_p}\right)_{fuse} + \left(C_{N_p}\right)_{ls}$$

$$\left(C_{N_p}\right)_{vt} = \frac{\ell_{vt}}{b}\left(C_{Y_p}\right)_{vt}$$

$$\left(C_{N_p}\right)_{fuse} = 2\frac{\left(Z_{cp}\right)_{fuse}}{b}\left(C_{N_\beta}\right)_{fuse}$$

$$\left(C_{N_p}\right)_{ls} = \frac{X_{ls}}{b}\left(C_{Y_p}\right)_{ls} + \left(C_{N_p}\right)_{ls}^{ac}$$

$$\left(C_{N_p}\right)_{ls}^{ac} = \left[\underbrace{\left(\frac{\left(\Delta C_{np}\right)_1}{C_L}\right)_{ls}}_{\text{from Ref. 14}} C_{L_{ls}} + \underbrace{\left(\frac{\left(\Delta C_{np}\right)_2}{\left(C_{D_0}\right)_\alpha}\right)_{ls}}_{\text{from Ref. 14}} \left(C_{Dp}\right)_{\alpha_{ls}}\right]\left(\frac{b_{ls}}{b}\right)^2\left(\frac{S_{ls}}{S}\right)$$

$$\left(C_{Dp}\right)_{\alpha_{ls}} = \left(\frac{dC_{Dp}}{d\alpha}\right)_{ls}$$ (this is estimated from the drag polar of the wing airfoil)

The r Derivatives

$$C_{Y_r} = \left(C_{Y_r}\right)_{vt} + \left(C_{Y_r}\right)_{ls}$$

$$\left(C_{Y_r}\right)_{vt} = \eta_{vt}\left(2\frac{\ell_{vt}}{b} + \frac{\partial\sigma_{vt}}{\partial\hat{r}}\right)\left(C_{Y_\beta}\right)_{vt}^{*} + \left(C_{Y_r}\right)_{vt}^{ac}$$

$$\left(C_{Y_r}\right)_{ls} = \left(C_{Y_\beta}\right)_{ls}\left(\frac{2X_{ls}}{b} + \frac{\partial\sigma_{ls}}{\partial\hat{r}}\right)\eta_{ls}$$

$$\left(C_{Y_r}\right)_{vt}^{ac} = -\left(C_{Y_\beta}\right)_{vt}\frac{\bar{c}_{vt}}{b}$$

where: $\bar{c}_{vt}$ is the mean aerodynamic chord of the vertical stabilizer and

$\frac{\partial\sigma_{ls}}{\partial\hat{r}}$ is the change in sidewash due to a change in the flight vehicle yaw rate.

$$C_{L_r} = \left(C_{L_r}\right)_{vt} + \left(C_{L_r}\right)_{ls}$$

$$\left(C_{L_r}\right)_{vt} = -\frac{Z_{vt}}{b}\left(C_{Y_r}\right)_{vt}$$

$$\left(C_{L_r}\right)_{ls} = -\frac{Z_{ls}}{b}\left(C_{Y_r}\right)_{ls} + \left(C_{L_r}\right)_{ls}^{ac}$$

$$\left(C_{L_r}\right)_{ls}^{ac} = \left(\frac{C_{L_r}}{C_L}\right)_{ls}\left(\frac{b_{ls}}{b}\right)^2\left(\frac{S_{ls}}{S}\right)\left(C_{L_\alpha}\right)_{ls}$$

$$\left(\frac{C_{L_r}}{C_L}\right)_{ls} = \frac{1 + \frac{AR\left(1 - B^2\right)}{2B\left(AR\,B + 2\cos\Lambda_{c/4}\right)} + \frac{AR\,B + 2\cos\Lambda_{c/4}}{AR\,B + 4\cos\Lambda_{c/4}}\frac{\tan^2\Lambda_{c/4}}{8}}{1 + \frac{AR\,B + 2\cos\Lambda_{c/4}}{AR\,B + 4\cos\Lambda_{c/4}}\frac{\tan^2\Lambda_{c/4}}{8}} \quad \text{(Ref. 7)}$$

$AR = \frac{b^2}{S} \Rightarrow$ the aspect ratio

$$B = \sqrt{1 - M^2\cos^2\Lambda_{c/4}}$$

where: M is the mach number for the flight vehicle at equilibrium

$$C_{N_r} = \left(C_{N_r}\right)_{vt} + \left(C_{N_r}\right)_{ls}$$

$$\left(C_{N_r}\right)_{vt} = \frac{\ell_{vt}}{b}\left(C_{Y_r}\right)_{vt} + \left(C_{N_r}\right)_{vt}^{ac}$$

$$\left(C_{N_r}\right)_{ls} = \frac{X_{ls}}{b}\left(C_{Y_r}\right)_{ls} + \left(C_{N_r}\right)_{ls}^{ac}$$

$$\left(C_{N_r}\right)_{vt} = -\frac{\pi}{4}\left(\frac{S_{vt}}{b}\right)\left(\frac{S_{vt}}{S}\right)$$

$$\left(C_{N_r}\right)_{ls}^{ac} = \left[\underbrace{\left(\frac{\left(\Delta C_{N_r}\right)_1}{C_L^2}\right)_{ls}}_{\text{Ref.14}}\left(C_{L_\alpha}\right)_{ls}^2 + \underbrace{\left(\frac{\left(\Delta C_{N_r}\right)_2}{C_{D_0}}\right)_{ls}}_{\text{Ref.14}}\left(C_{D_p}\right)_{ls}\right]\left(\frac{S_{ls}}{S}\right)\left(\frac{b_{ls}}{b}\right)^2$$

The THARWP equations of motion, developed using Lagrange's equations and a discrete-element tether model, were transformed into first-order linear differential equations to facilitate the calculation of their solutions. In order to use the Newmark method, the longitudinal and the lateral equations of motion, Eqs. (74) and (76), must be rewritten in the form of second-order linear differential equations. The longitudinal time step analysis will be discussed first.

Recall the form of the longitudinal equations of motion from Eq. (73) as follows:

$$\underline{A}\,\underline{\zeta} = \underline{B}\,\dot{\underline{\zeta}}$$

where: $\underline{\zeta}^{T} = \begin{bmatrix} u & w & q & \theta & \delta T & \gamma_{B} & \underline{h}_{lg}^{T} & \underline{q}^{T} \end{bmatrix}$

$\underline{q}^{T} = [\delta\ell_{1}, \gamma_{1}, \delta\ell_{2}, \gamma_{2}, \cdots \delta\ell_{N}, \gamma_{N}]$ and

$\underline{h}_{lg}^{T} = [\delta\dot{\ell}_{1}, \dot{\gamma}_{1}, \delta\dot{\ell}_{2}, \dot{\gamma}_{2}, \cdots \delta\dot{\ell}_{N}, \dot{\gamma}_{N}]$

The equivalent second-order system of equations is given by:

$$\underline{M}\,\ddot{\underline{\zeta}}' + \underline{C}\,\dot{\underline{\zeta}}' + \underline{K}\,\underline{\zeta}' = \underline{F}_{C} \tag{7.1}$$

where: $\underline{\zeta}'^{T} = \begin{bmatrix} x & z & \theta & \delta T & \gamma_{B} & \underline{q}^{T} \end{bmatrix}$ is the 2N+5 element state vector;

$\underline{M}$, $\underline{C}$, and $\underline{K}$ are the mass, damping, and stiffness matrices, respectively, constructed from Eqs. (57), (62a-d), (66a, c, e), (70a, c, e), and (72a, c); and

$\underline{F}_{C}$ are forces and moments acting on the THARWP due to longitudinal external disturbances.

Once the substitutions are made for $\underline{M}$, $\underline{C}$, and $\underline{K}$, the Newmark method can be used to integrate the longitudinal equations of motion over a specified time duration. Let Δt be the constant time step for which the accelerations are assumed constant. Using the Newmark method, the equations for the displacements and the velocities are as follows:

$$\underline{\zeta}_{t+\Delta t} = \underline{\zeta}_{t} + \underline{\zeta}_{t}\Delta t + \underline{\zeta}_{av}\frac{\quad}{2} \tag{7.2a}$$

$$\underline{\dot{\zeta}}'_{t+\Delta t} = \underline{\dot{\zeta}}'_{t} + \underline{\ddot{\zeta}}'_{av}\Delta t \tag{7.2b}$$

$$\text{where:} \quad \underline{\ddot{\zeta}}'_{av} = \tfrac{1}{2}\left(\underline{\ddot{\zeta}}'_{t} + \underline{\ddot{\zeta}}'_{t+\Delta t}\right) \tag{7.2c}$$

Through substitution of Eqs. (7.2a-c) into Eq. (7.1) at t = t +Δt, the following recurrence formula can be written in terms of an effective stiffness matrix and load matrix:

$$\underline{\widetilde{K}}\,\underline{\zeta}'_{t+\Delta t} = \underline{\widetilde{F}}_{t+\Delta t} \tag{7.3a}$$

$$\text{where:} \quad \underline{\widetilde{K}} = \underline{K} + \frac{2}{\Delta t}\underline{C} + \frac{4}{\Delta t^{2}}\underline{M} \text{ is the effective stiffness matrix and} \tag{7.3b}$$

$$\underline{\widetilde{F}}_{t+\Delta t} = \underline{F}_{t+\Delta t} + \underline{C}\left(\frac{2}{\Delta t}\underline{\zeta}'_{t} + \underline{\dot{\zeta}}'_{t}\right) + \underline{M}\left(\frac{4}{\Delta t^{2}}\underline{\zeta}'_{t} + \frac{4}{\Delta t}\underline{\dot{\zeta}}'_{t} + \underline{\ddot{\zeta}}'_{t}\right) \tag{7.3c}$$

is the effective load matrix

Note that the effective stiffness matrix is independent of time and, therefore, can be evaluated prior to the integration. For each time increment the effective load matrix is evaluated, the effective stiffness matrix is then inverted, and the displacements at t = t +Δt are calculated using:

$$\underline{\zeta}'_{t+\Delta t} = \underline{\widetilde{K}}^{-1}\,\underline{\widetilde{F}}_{t+\Delta t} \tag{7.4}$$

Subsequently, the velocities and accelerations at time = t +Δt can be calculated by substituting the displacements from Eq. (7.4) back into Eqs. (7.2a-c). This process was repeated until the end of the specified analysis duration, and this results in the total response of the THARWP to some given disturbance. Note that there must be an initial disturbance force or moment in order for there to be any motion of the THARWP.

The lateral time-integration analysis was developed in the same way as above. The only difference was the initial system of second-order linear differential equations. The lateral equations of motion from Eq. (75) were as follows:

$$\underline{A}\,\underline{\eta} = \underline{B}\,\dot{\underline{\eta}}$$

where: $\underline{\eta}^T = \begin{bmatrix} v & p & r & \phi & \psi & \beta_B & \underline{h}_{lt}^T & \underline{q}^T \end{bmatrix}$

$\underline{q}^T = [\beta_1, \beta_2, \cdots \beta_N]$ and

$\underline{h}_{lt}^T = [\dot{\beta}_1, \dot{\beta}_2, \cdots \dot{\beta}_N]$

The equivalent second-order system of equations is given by:

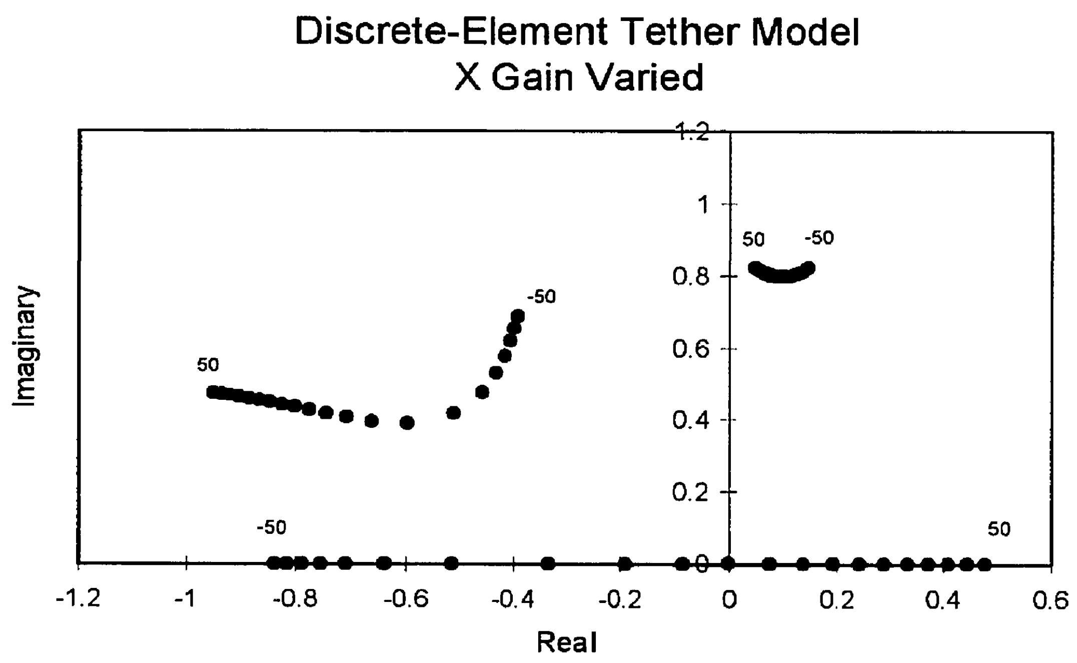
$$\underline{M}\,\ddot{\underline{\eta}}' + \underline{C}\,\dot{\underline{\eta}}' + \underline{K}\,\underline{\eta}' = \underline{F}_C \qquad (7.5)$$

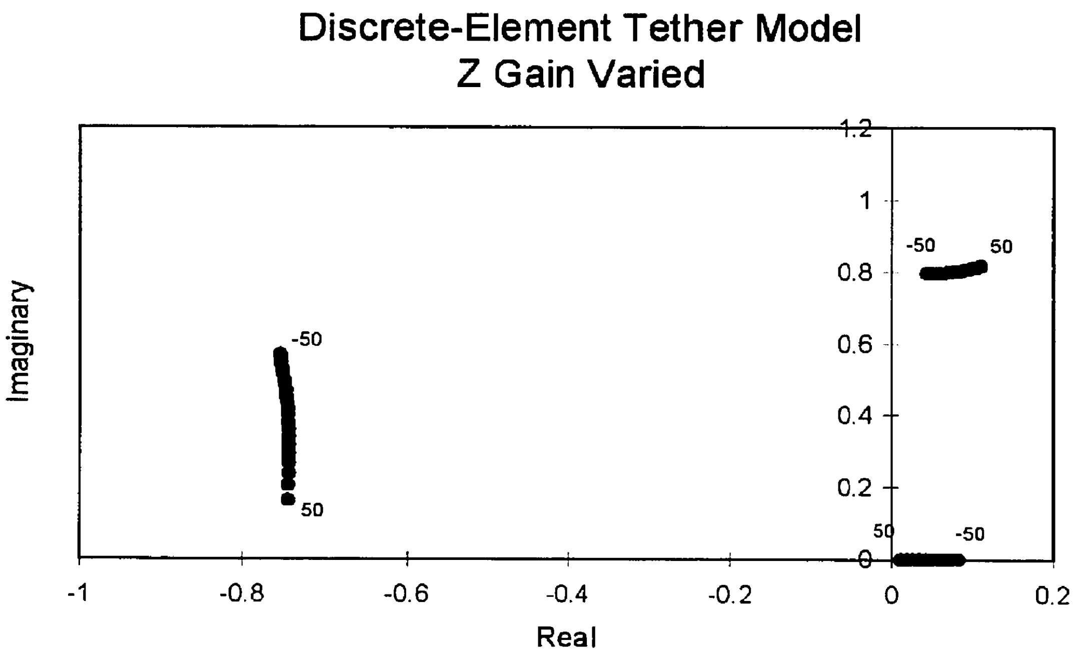
where: $\underline{\eta}'^T = \begin{bmatrix} y & \phi & \psi & \beta_B & \underline{q}^T \end{bmatrix}$ is the N+4 element state vector;

$\underline{M}$, $\underline{C}$, and $\underline{K}$ are the mass, damping, and stiffness matrices, respectively, constructed from Eqs. (61), (62e-i), (66b, d, f), (70b, d, f), and (72b); and

$\underline{F}_C$ are forces and moments acting on the THARWP due to lateral external disturbances.

The remaining lateral time-integration analysis is developed in the identical manner as the longitudinal analysis, substituting Eq. (7.5) for Eq. (7.1). Both the longitudinal and the lateral time-integration analyses were implemented as subroutines contained in the C++ file called suprkit3.cpp, which is part of the Trimstat source code listed in Appendix 14.


Discrete-Element Tether Model
X Gain Varied
Imaginary
Real
50
-50
50
-50
-50
50
1.2
1
0.8
0.6
0.4
0.2
0
-1.2
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6


Discrete-Element Tether Model
Z Gain Varied
Imaginary
Real
-50
50
-50
50
50
-50
1.2
1
0.8
0.6
0.4
0.2
0
-1
-0.8
-0.6
-0.4
-0.2
0
0.2

Discrete-Element Tether Model
Pitch Gain Varied

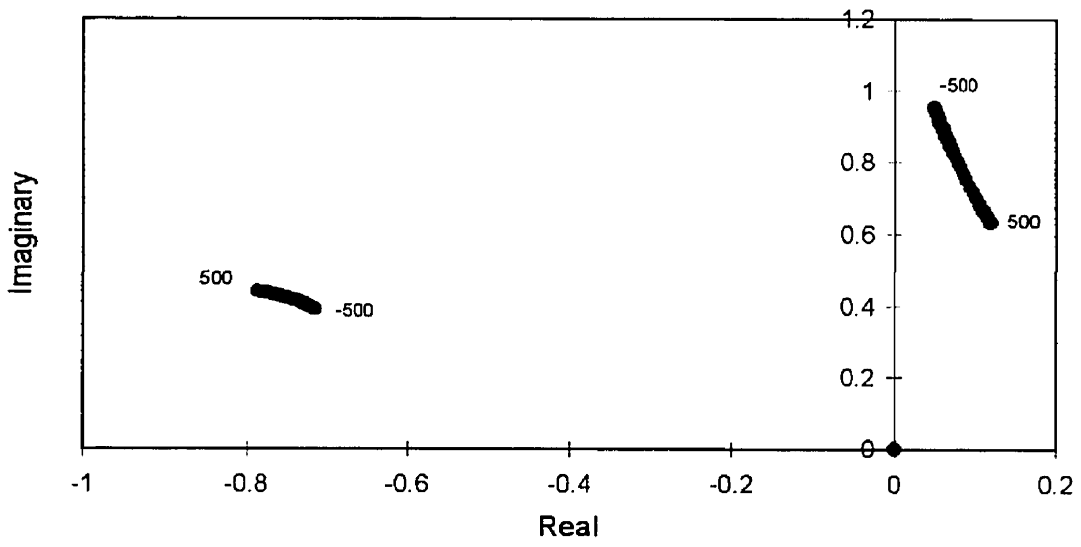


Figure A9.1

I_{xx} = 1000...2000 kg m^2

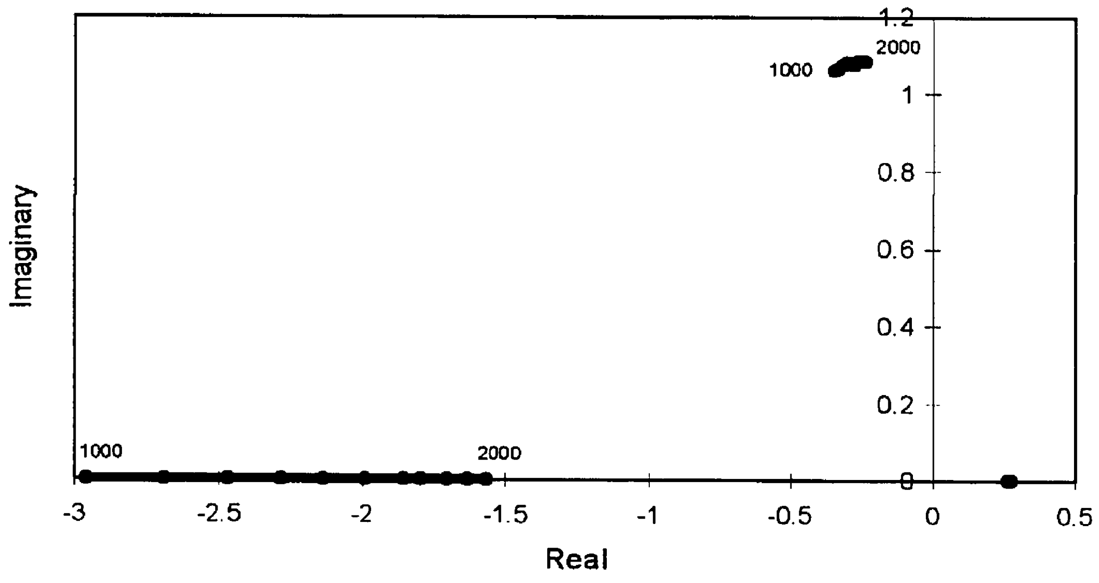


Figure A9.2

I_{zz} = 1500...2500 kg m^2

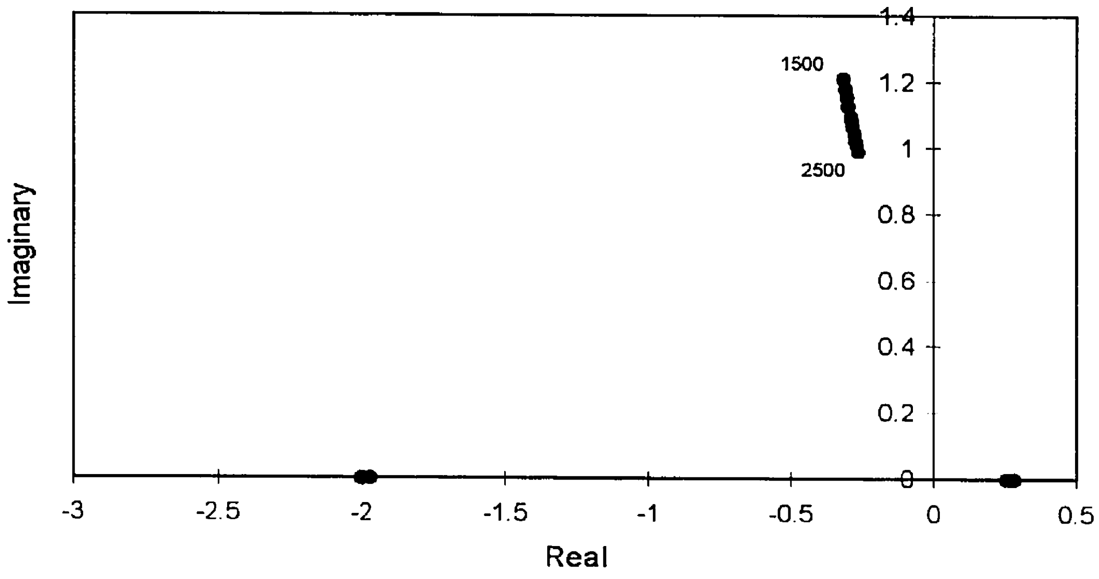


Figure A9.3

I_{yy} = 1200...2200 kg m^2

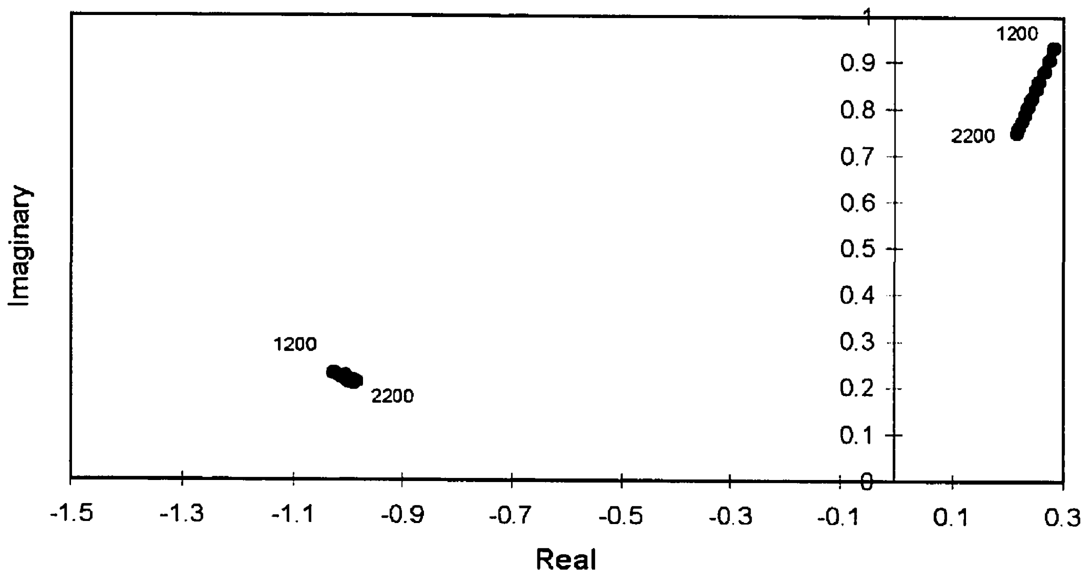


Figure A9.4

Trim-State Tether Profiles vs. Tether Material

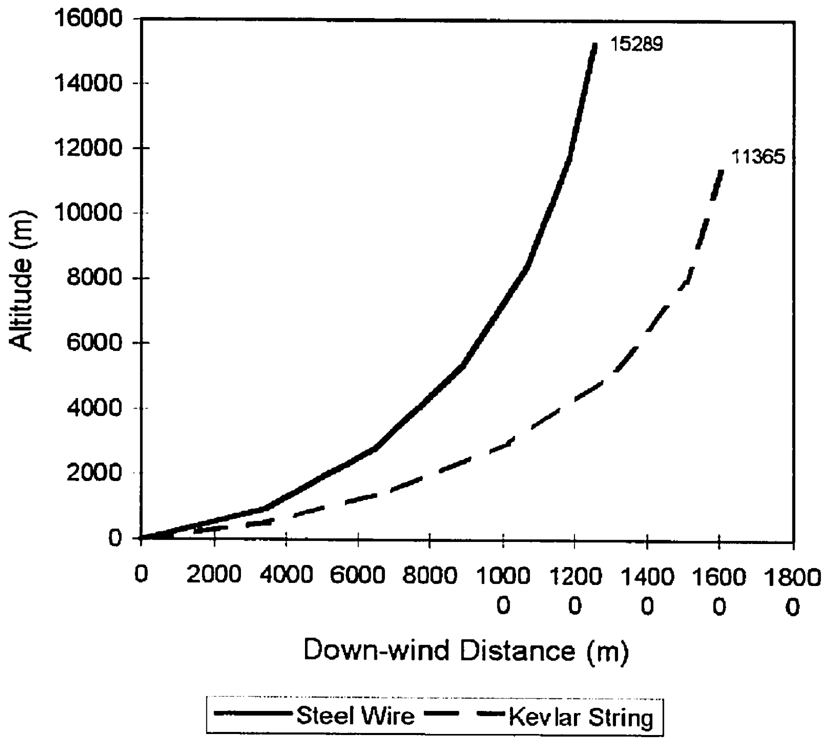


Figure A9.5

Effects of Minimum Tether Angle Variation

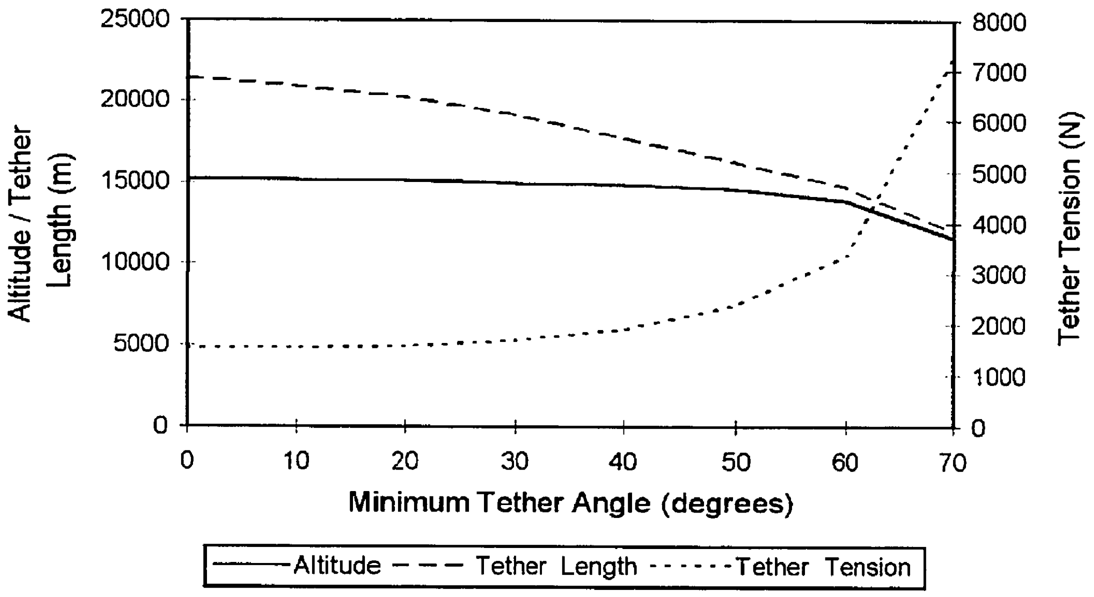


Figure A9.6a

Long. Stability Roots vs. Minimum Tether Angle

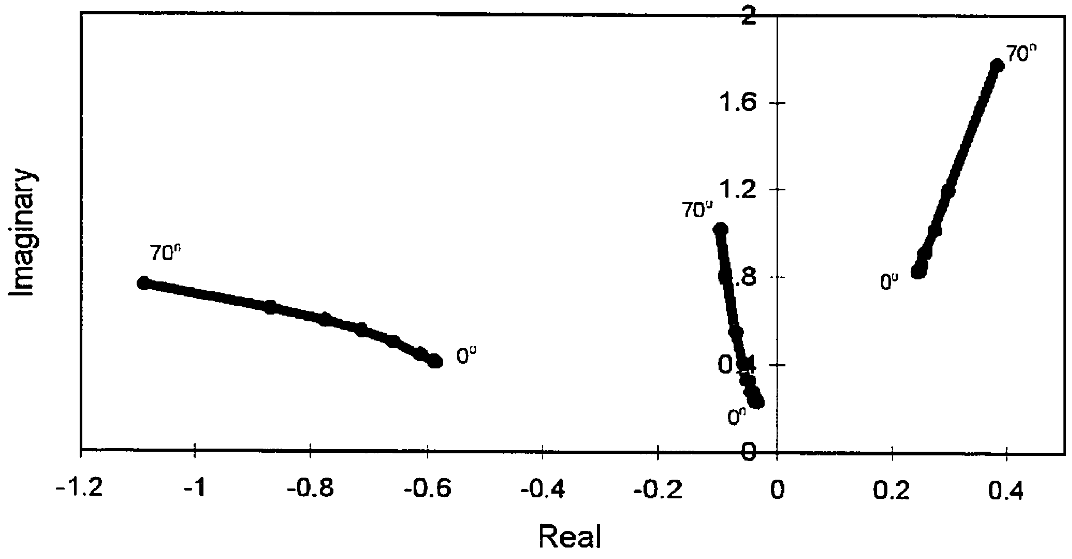


Figure A9.6b

Lat. Stability Roots vs. Minimum Tether Angle

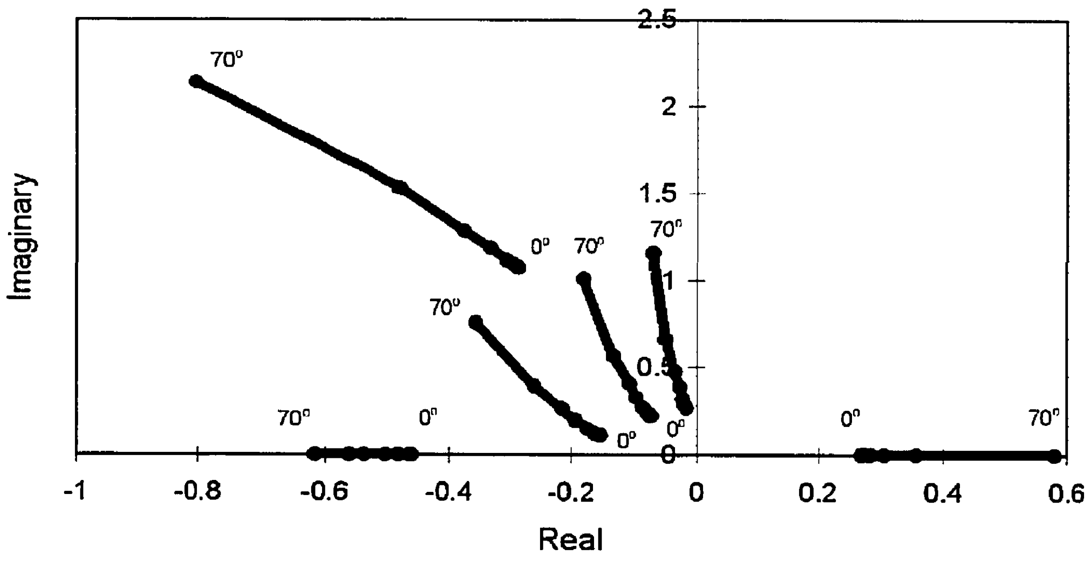


Figure A9.7

Wing Dihedral, Γ = -10°...40°

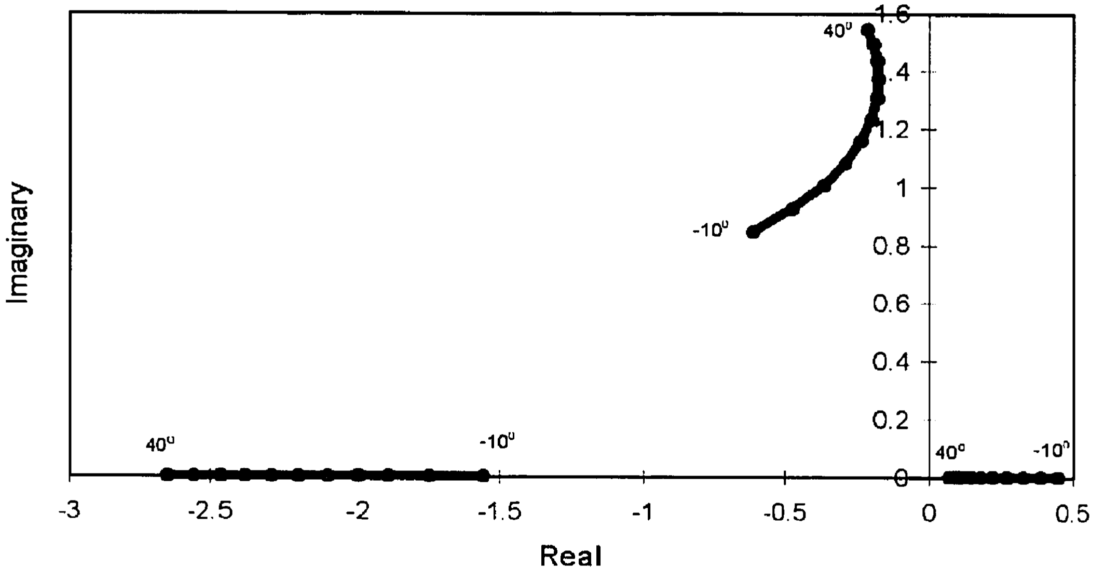


Figure A9.8a

Long. Stability Roots (X_{cg} = 1.40...2.00)

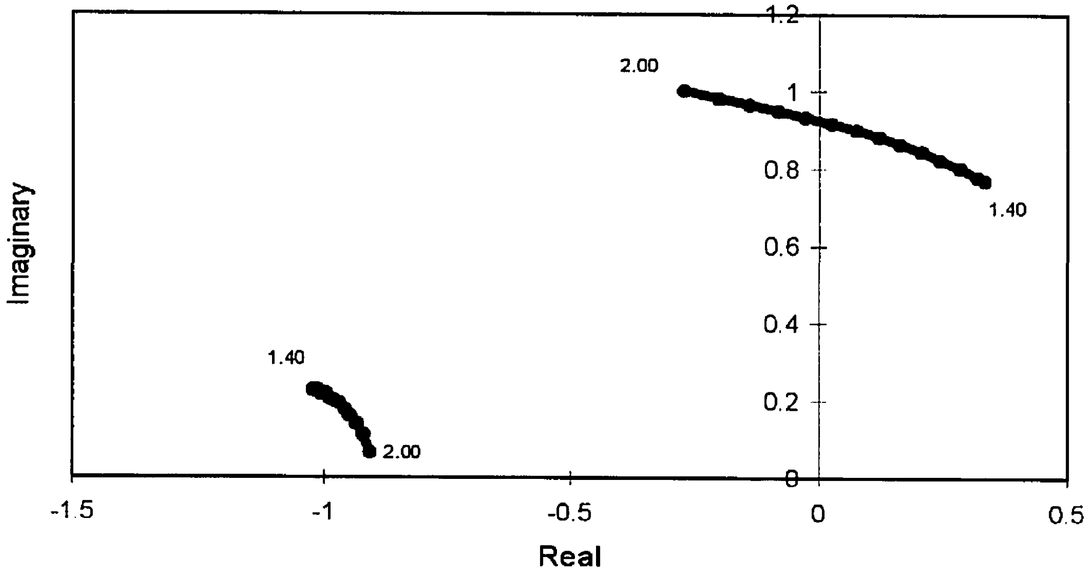


Figure A9.8b

Lat. Stability Roots (X_{cg} = 1.40...2.00)

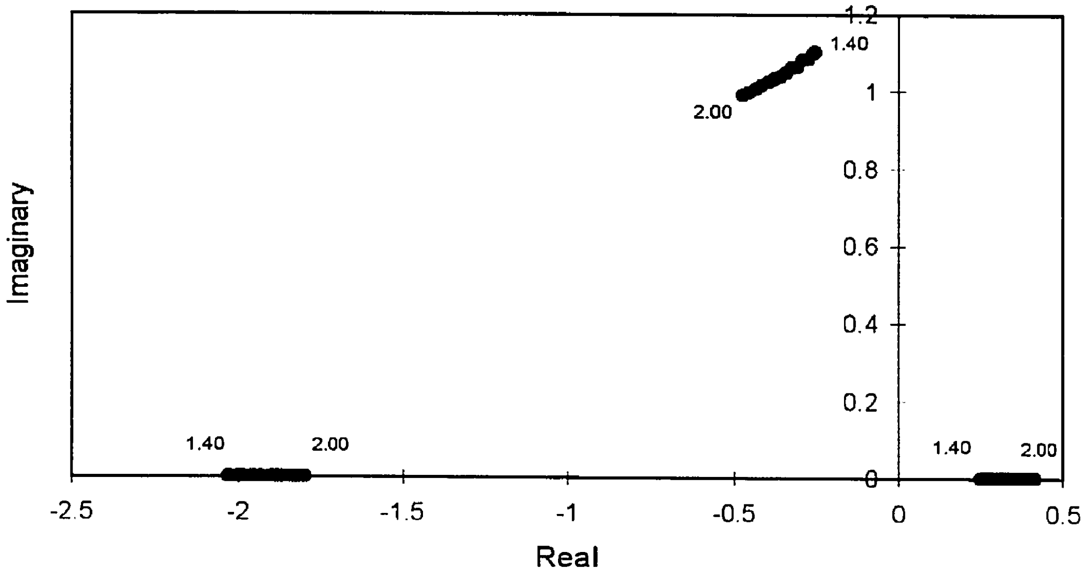


Figure A9.9a

Long. Stability Roots (X_{ac} = 1.60...0.60)

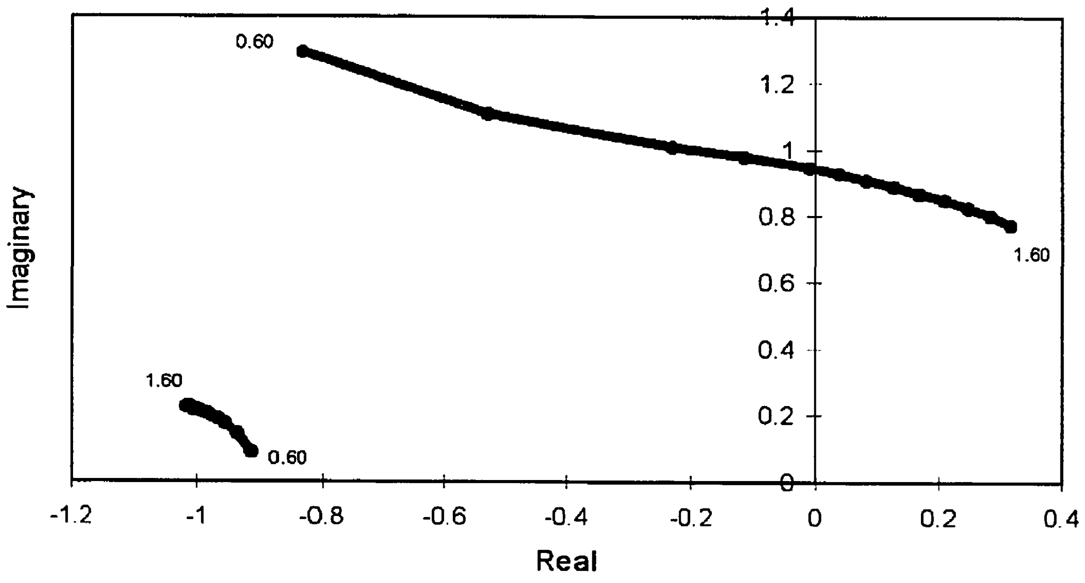


Figure A9.9b

Lat. Stability Roots (X_{ac} = 1.60...0.60)

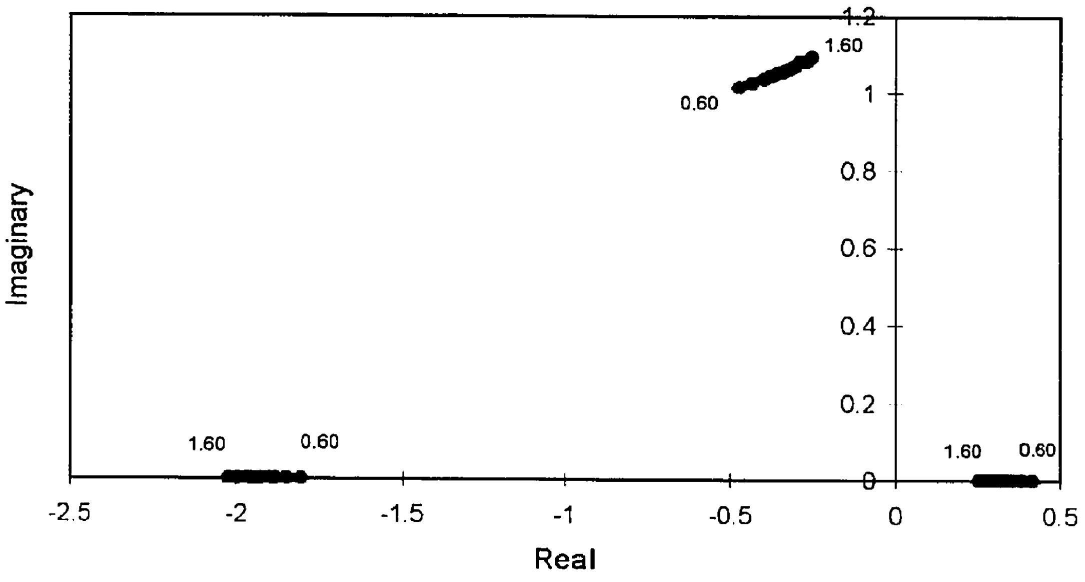


Figure A9.10

Lat. Stability Roots vs. Vertical Stabilizer Position

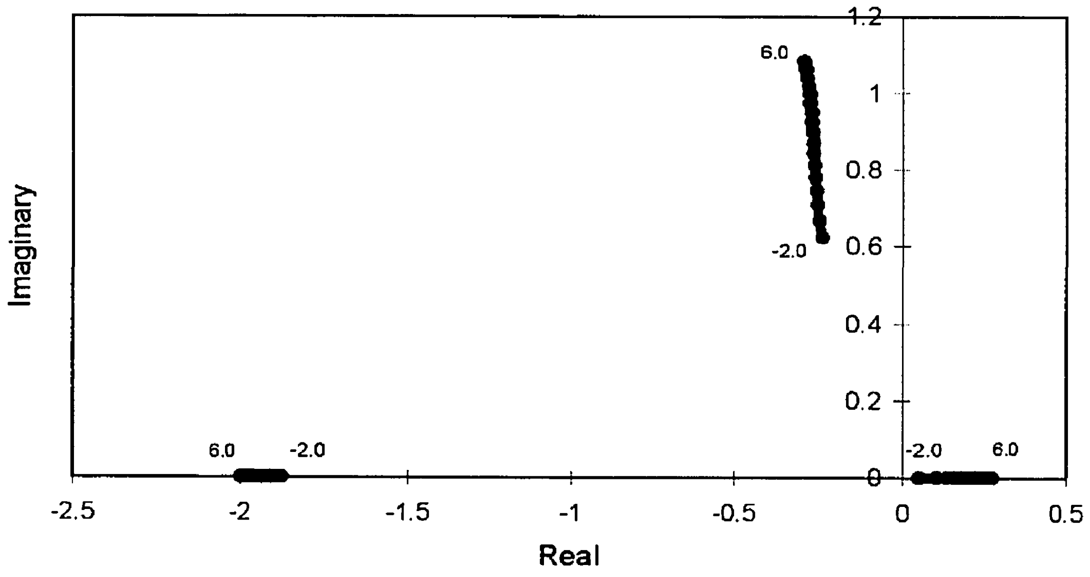


Figure A9.11

Lat. Stability Roots vs. Second Vertical Stabilizer Position

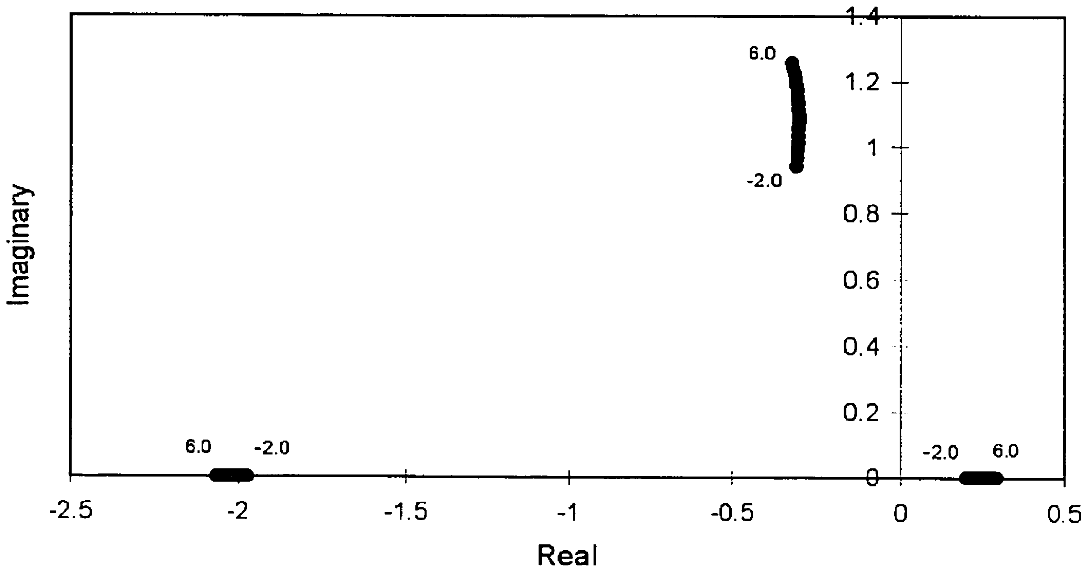


Figure A9.12

Long. Stability Roots vs. Horizontal Stabilizer Position

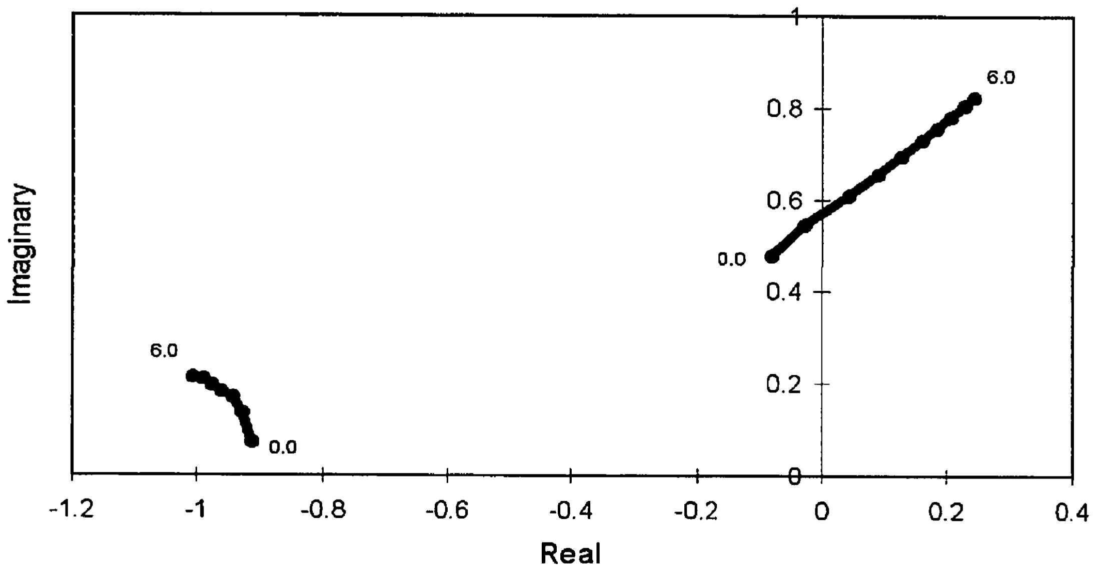


Figure A9.13a

Long. Stability Roots vs. Horizontal Stabilizer Span (Aft)

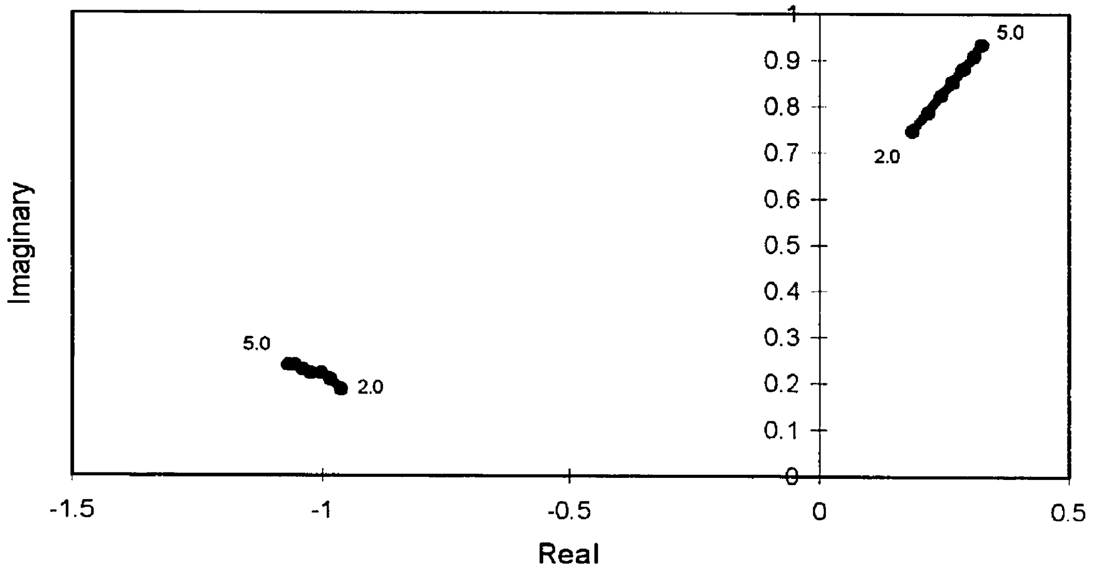


Figure A9.13b

Long. Stability Roots vs. Horizontal Stabilizer Span (forward)

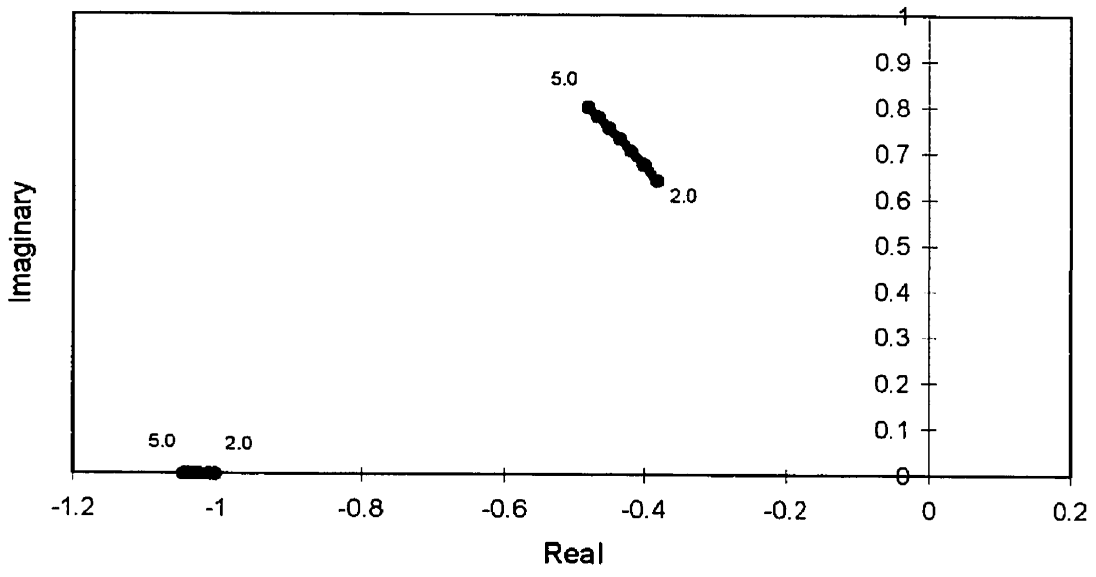


Figure A9.14

Long. Stability Roots vs. Second Horizontal Stabilizer Position

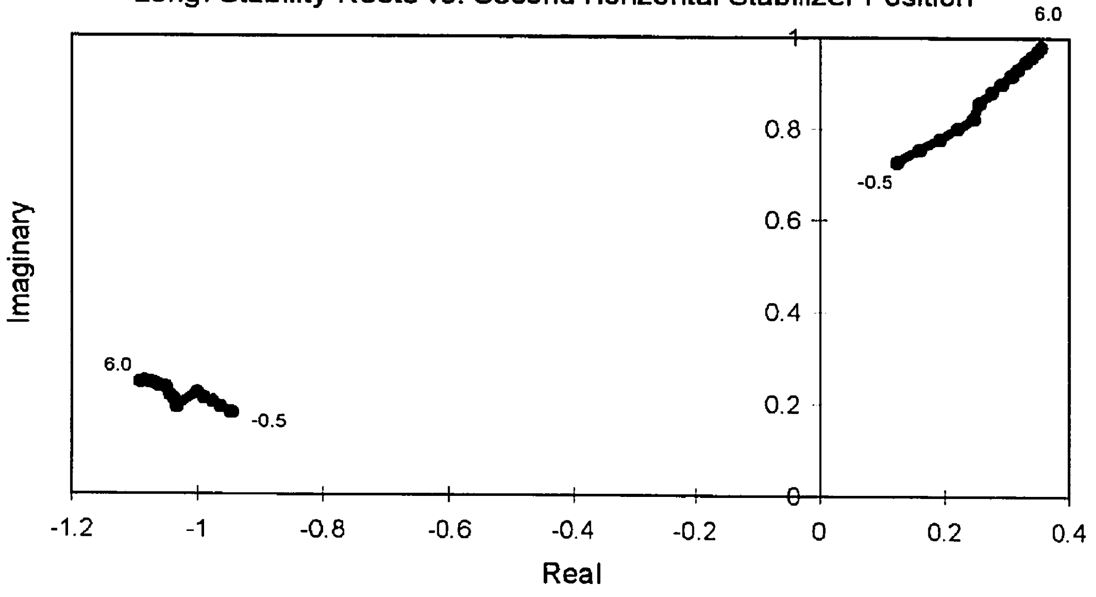


Figure A9.15a

C_{cX} = -50...50

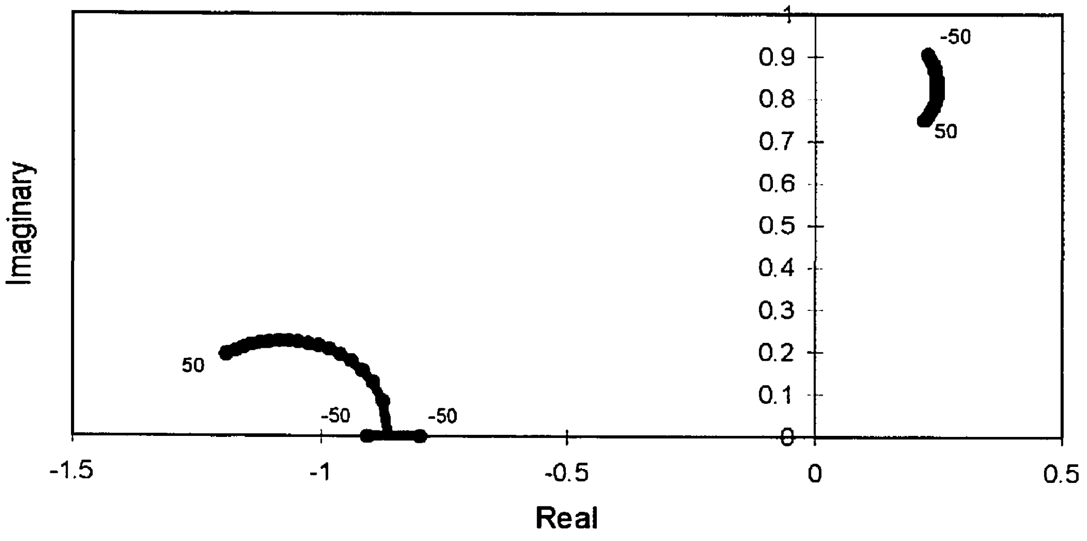


Figure A9.15b

C_{cZ} = -50...50

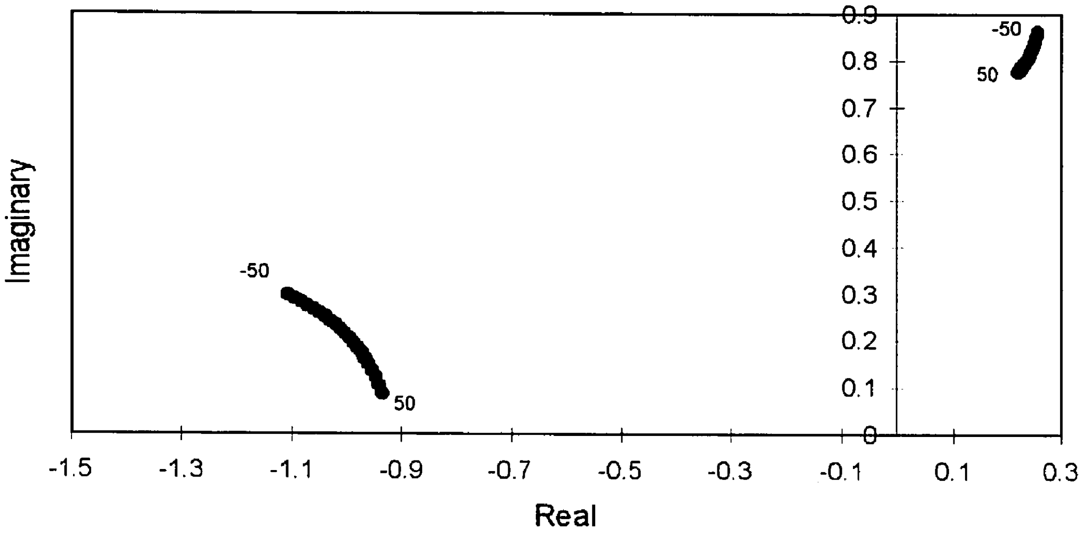


Figure A9.15c

$C_{cM} = -500...500$

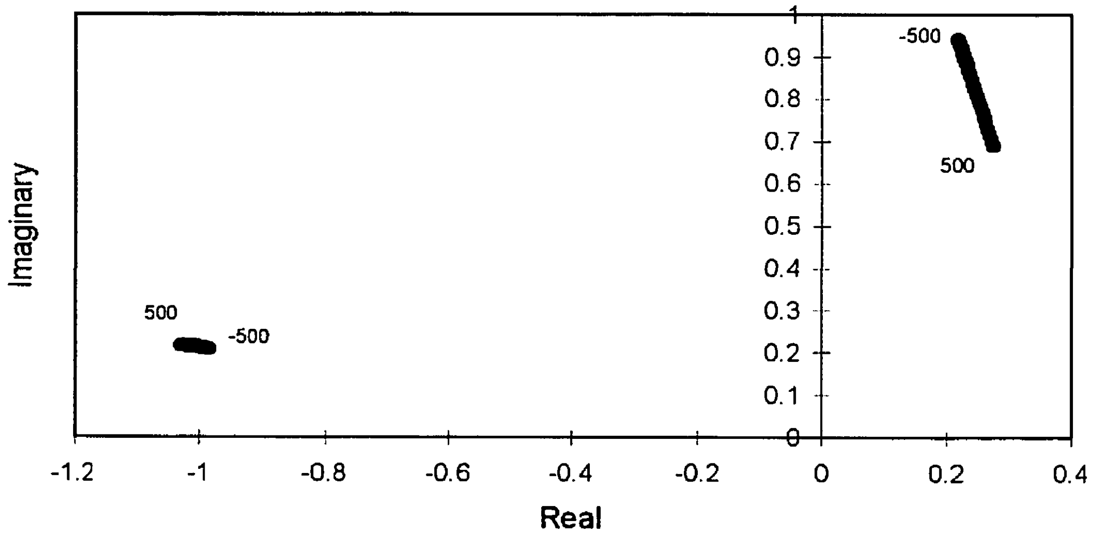


Figure A9.16a

$C_{cY} = -50...50$

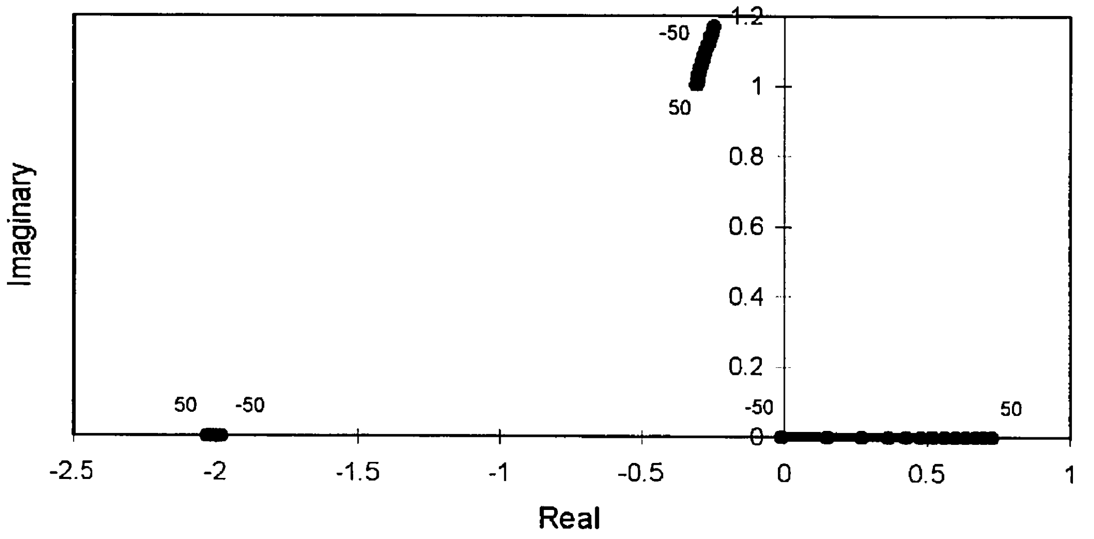


Figure A9.16b

C_{cL} = -500...500

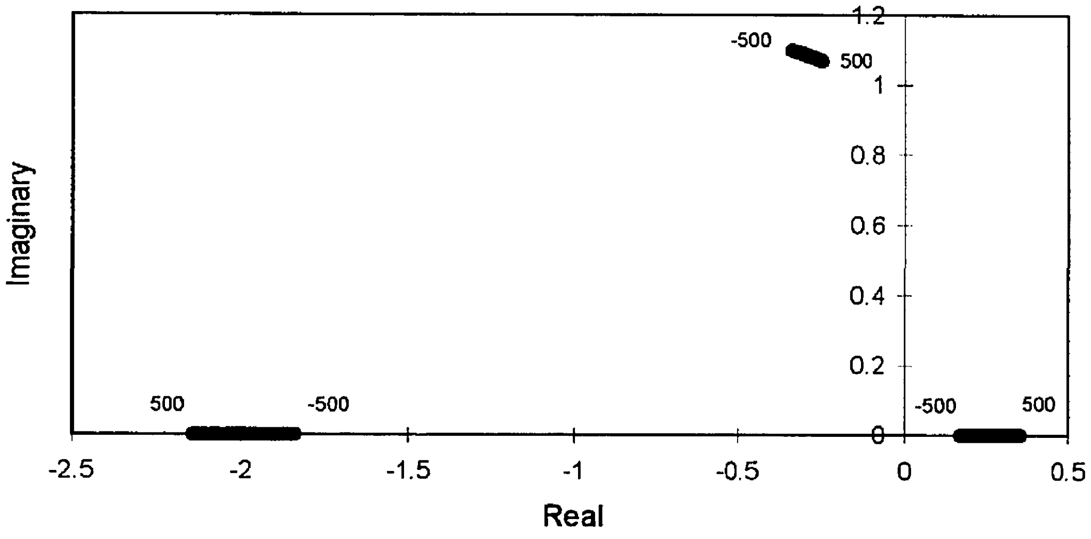


Figure A9.16c

C_{cN} = -500...500

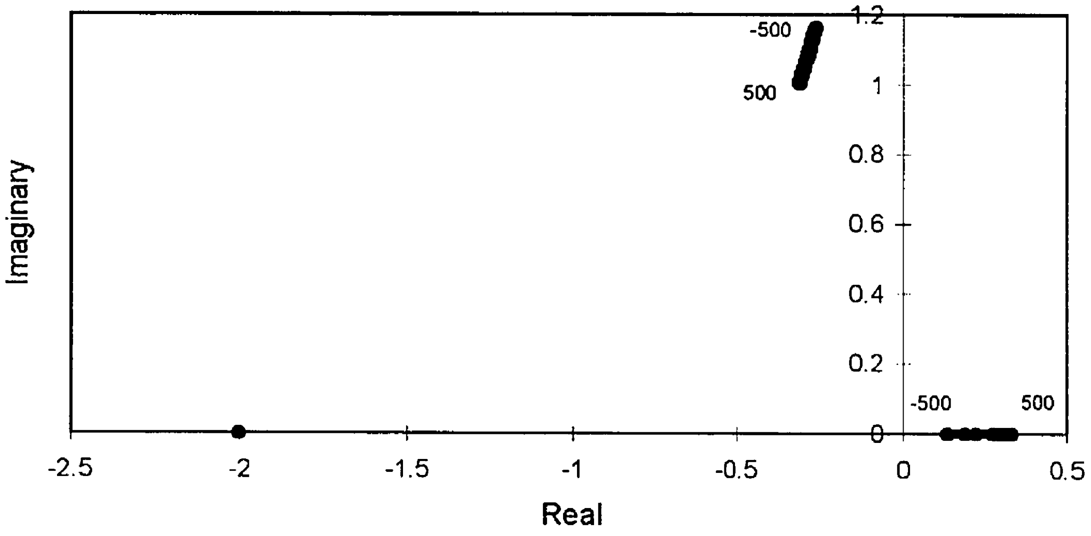


Figure A9.17a

Time-Step Analysis - Longitudinal

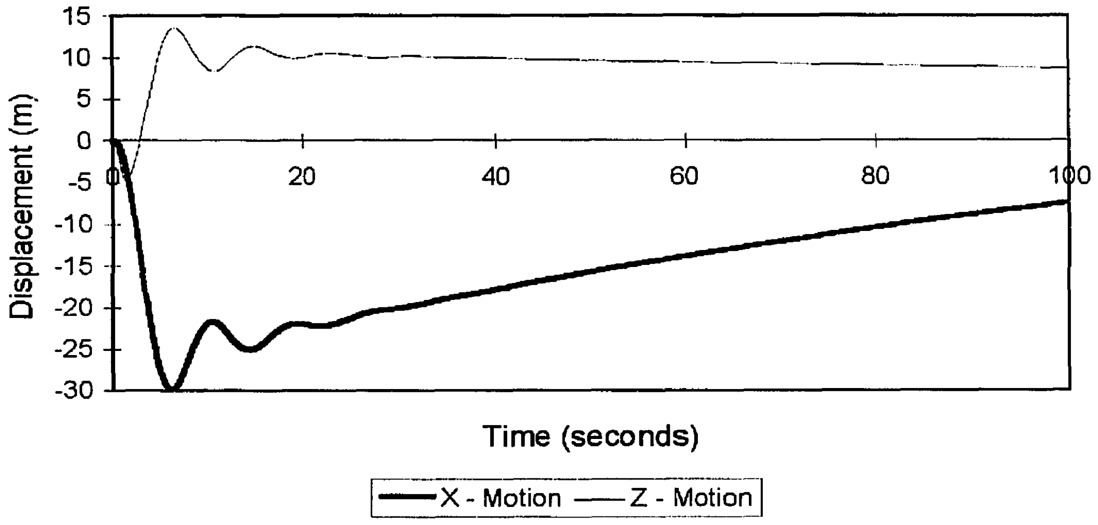
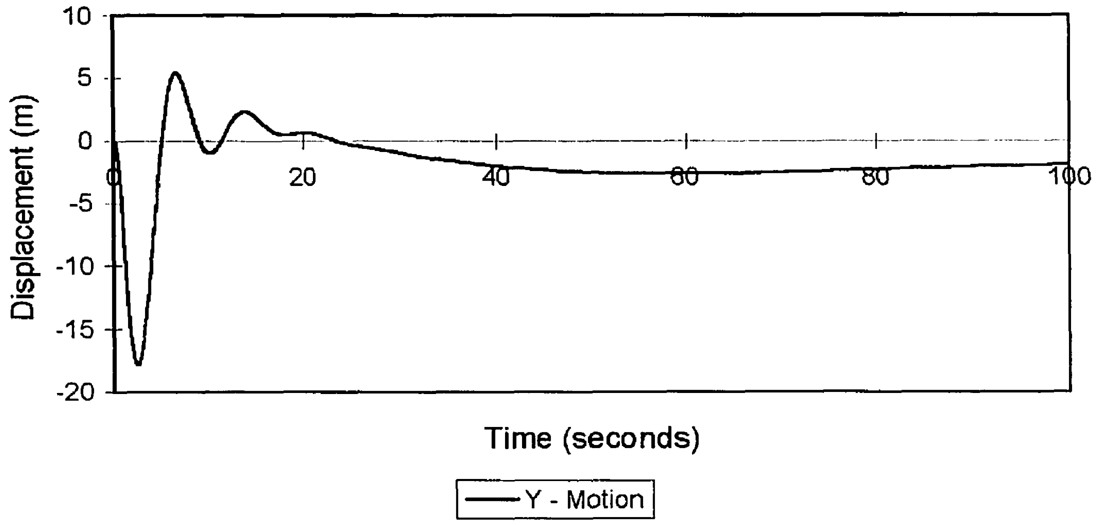


Figure A9.17b

Time-Step Analysis - Lateral



General Specifications

Flight vehicle mass (includes payload)	100.0 kg
Horizontal position of the flight vehicle's CG	1.53 m
Vertical position of the flight vehicle's CG	0.05 m
Horizontal position of the confluence point	1.391 m
Vertical position of the confluence point	-0.5 m
I_{xx} moment of inertia	1500 kg m^2
I_{yy} moment of inertia	1700 kg m^2
I_{zz} moment of inertia	2000 kg m^2
I_{xz} moment of inertia	10 kg m^2
Tether material density	7860 kg/m^3
Tether elastic modulus	200 GPa
Tether diameter	0.001 m
Tether cross-flow coefficient of drag	0.8
Tether axial-flow coefficient of drag	0.1
Internal tether damping	100.0 N s/m
Minimum tether angle at the attachment point	20°

Flight Vehicle Wing

Span	15.0 m
Root chord	1.5 m
Tip chord	1.5 m
Lift-curve slope	5.0 /rad
Maximum coefficient of lift, $C_{L\max}$	2.0
Coefficient of moment, C_M	0.1
Dihedral	15°
Leading edge sweep	0°
¼ line sweep	0°
Incidence angle	8°
Zero lift angle	13°
Horizontal position of the aerodynamic center (AC)	1.5 m
Vertical position of the AC	0.2 m
Position of the AC aft of the leading edge (given as a percentage of the mean chord length)	0.3
Zero lift downwash angle	1°

Horizontal Stabilizer #1

Span	4.0 m
Mean chord	0.5 m
Lift-curve slope	4.0 /rad
Horizontal position of the AC	6.0 m
Vertical position of the AC	0.1 m
Incidence angle	0°
Zero lift angle	0°

Vertical Stabilizer #1

Span	1.25 m
Mean chord	0.5 m
Lift-curve slope	4.0 /rad
Horizontal position of the AC	6.0 m
Vertical position of the AC	0.6 m

Horizontal Stabilizer #2

Span	3.0 m
Mean chord	1.0 m
Lift-curve slope	4.0 /rad
Horizontal position of the AC	-1.0 m
Vertical position of the AC	0.1 m
Incidence angle	0°
Zero lift angle	0°

Vertical Stabilizer #2

Span	2.0 m
Mean chord	1.5 m
Lift-curve slope	4.0 /rad
Horizontal position of the AC	-1.0 m
Vertical position of the AC	0.6 m

Fuselage

Total enclosed volume	1.0 m^3
Maximum width	0.25 m
Maximum height	0.5 m
Width at ¼ fuselage length	0.22 m
Height at ¼ fuselage length	0.45 m
Width at ¾ fuselage length	0.1 m
Height at ¾ fuselage length	0.15 m
Characteristic length	3.0 m
Total length of the fuselage	7.0
Fuselage lift-curve slope	0.0525 /rad
Height of center of pressure	0.1 m
Horizontal position of the volume center	1.3

Control Gains

C_{CX}	-10.0	C_{CY}	0.0
C_{CZ}	-10.0	C_{CL}	-150.0
$C_{C\theta}$	-50.0	C_{CN}	0.0

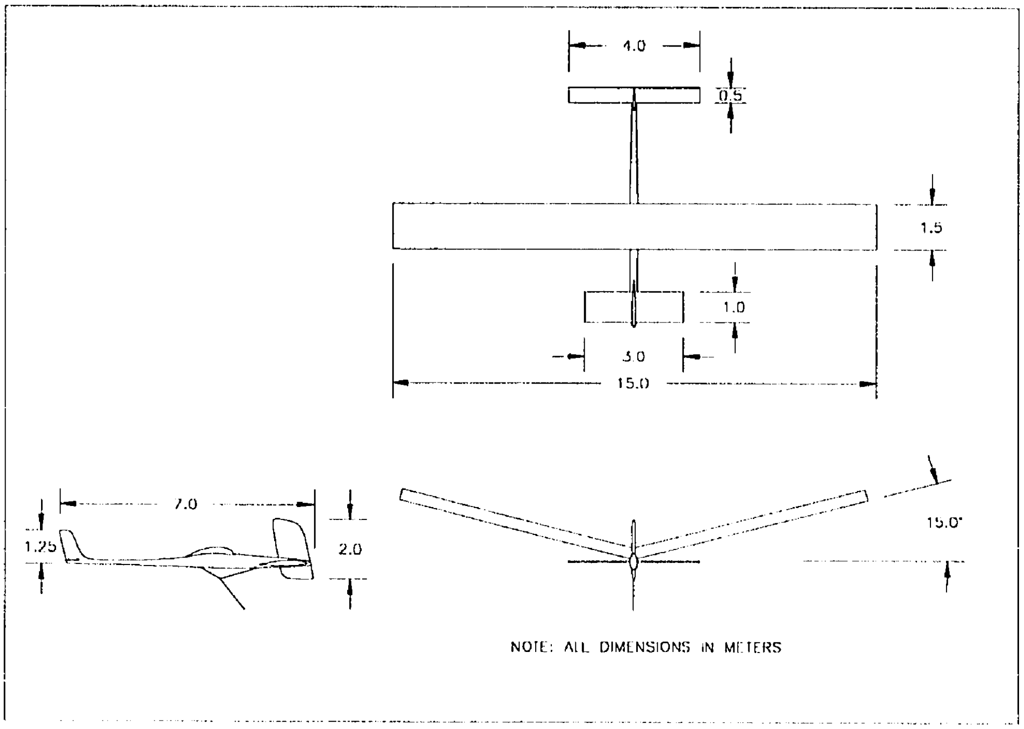


Figure A10.1. 3 - View Diagram of the Optimized THARWP

JDD_METH.H

```
#ifndef _JDD_METH_H_
#define _JDD_METH_H_

void balanc(double **a, int n);
void elmhes(double **a, int n);
void hqr(double **a, int n, double *wr, double *wi);
void nrerror(char error_text[]);

#endif
```

JDD_METH.CPP

```
//Eigen Value Calculation for JDD_Method

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <float.h>
#include <except.h>
#include <dos.h>
#include <string.h>
#include <iostream.h>
#include <iomanip.h>
#include <fstream.h>
#include "matrix.h"
#include "jdd_meth.h"

const double pi=3.141592654;
int proceed=0;
const double res=20.0;
char *filename = "rootslcs.txt";
fstream dest;

//Function to set which gain will be varied and what the limits are
int GainSet(matrix &G)
{
   int test;
   proceed=0;
   do
   {
      cout<<"\n\nWhich control gain would you like to vary?:\n";
      cout<<"1.  GX\t\t";
      cout<<"2.  GXd\n";
      cout<<"3.  GZ\t\t";
      cout<<"4.  GZd\n";
      cout<<"5.  GM\t\t";
      cout<<"6.  GMd\n\n";
      cout<<"7.  GN\t\t";
      cout<<"8.  GNd\n";
      cout<<"9.  GL\t\t";
      cout<<"10.  GLd\n";
      cout<<"11.  GY\t\t";
      cout<<"12.  GYd\n";
      cout<<"\nPlease enter the number of your choice: ";
      cin>>test;
      cout<<"\n\n";

      switch(test)
      {
         case 2:
         case 3:
         case 4:
         case 5:
         case 6:

            //Longitudinal Control Vector
            //Zero the control gain array
            matrix_zero(G);

            _fpreset();
            cout<<"\n\nPlease enter the lower limit of the chosen gain: ";
            cin>>G(test,1);
            cout<<"\nNow the upper limit of the chosen gain: ";
            cin>>G(test,2);
            G(test,3)=G(test,1);
            proceed=2;
            break;

         case 7:
         case 8:
         case 9:
         case 10:
         case 11:
         case 12:

            //Lateral Control Vector
            //Zero the control gain array
            matrix_zero(G);

            _fpreset();
            cout<<"\n\nPlease enter the lower limit of the chosen gain: ";
            cin>>G(test-6,1);
            cout<<"\nNow the upper limit of the chosen gain: ";
            cin>>G(test-6,2);
            G(test-6,3)=G(test-6,1);
            proceed=3;
            test-=6;
            break;

         default:
            cout<<"\n\nYou have made an invalid selection\n\n";
            proceed=0;
            break;
      }
   }while(proceed==0);

   return(test);
}


void main()
{
   int i,j,choice,select,invflag,flag;
   matrix G(6,3),M(3,3),Ml(2,2),D(3,3),Dl(2,2),K(3,3),Kl(2,2);
   matrix Minv(3,3),Mlinv(2,2),Dp(3,3),Dlp(2,2),Kp(3,3),Klp(2,2);
   double **m,*er,*ei;
   double mass,b,S,dens,AR,Xcg,Zcg,Xls,Zls,Xcp,Zcp,Xtp,Ztp,Xf,Ixz,Ixx,Iyy,Izz,Re;
   double Kvisc,Nfg,Azll,dEpsilondq,VH,CWo,Cmcpo,bf;
   double grav,Uo,Thetao,To,dZdS,dYdS,dXdS,dSigfdP,sweep,dSigfdr,dSiglsdr;
   double Xnp,Znp,taper,croot,ctip,cmean,cswp,B,Mach,Temp,Sside,dSiglsdP;
   double Nf,Nls,Nlsl,dSigmadB,Ki,Vol,Gamma,Sf,Zf,Zcpbody,KB,lbody,hmax,h1,h2,w1,w2;
   double lf,af,cf,abody,als,ils,ihs,CLo,CDo,CNbCL,Cnp1,Cnp2,CDpls,Cnr1,Cnr2;
   double KA,Klam,KH,dEpsilonda,Xt,Zt,lt,bt,ct,St,at,Xcv,atether,ath,ddS;
   double Cdof,Cybfs,Cybf,Cybbody,Cybls,Cyb,CLbf,CLbls,CLbbody,CLb,CNbbody;
   double CNbf,CNblsac,CNbls,CNb,Cypbody,Cypf,Cyplsac,Cypls,Cyp,CLpbody;
double CYrfac,CYrl,CYrls,CYr,CLrl,CLrCL,CLrCLo,CLrlsac,CLrls,CLr;
double CNrfac,CNrf,CNrlsac,CNrls,CNr,Yv,Yp,Yr,Lv,Lp,Lr,Nv,Np,Nr;
double Cxu,Czu,Cmu,Cxalpha,Czalpha,Cmalpha,Czalphals,Czqlsac,Czqls,Czqt,Czq;
double Cxalphals,Cxqlsac,Cxqls,Czadot,Czadotbody,Czadotls,Czadott;
double Cmadot,Cmadotbody,Cmadotls,Cmadott,Cxcpo,Czcpo,Cmqlsac,Cmqls,Cmqt;
double Cmq,Xu,Xw,Zu,Zw,Zwdot,Zq,Mu,Mw,Mwdot,Mq,Lth;

//Define all Kite parameters
//Total Kite
mass=50.0;                 //Aircraft mass (kg)
Xcg=1.5;                   //distance of CG from nose ref. pt.
Zcg=0.05;                  //distance +ve up of CG above nose ref. pt
Xcp=1.31;                  //distance of confluence pt. from nose ref. pt.
Zcp=-0.5;                  //distance of confluence pt. above nose ref. pt. (+ve above)
Xtp=Xcg-Xcp;               //distance of confluence pt. in front of CG
Ztp=Zcg-Zcp;               //distance of confluence pt. below CG
Xnp=1.70;                  //distance of neutral pt. from nose ref. pt.
Znp=0.15;                  //distance of neutral pt. above nose ref. pt. (+ve above)
Thetao=0.0;                //Aircraft Equilibrium AOA (degrees)
To=163.3;                  //Cable tension at equilibrium (Newtons)
dZdS=0.945265405;          //SIN of longitudinal cable angle at cp
dYdS=0.0;                  //COS of lateral cable angle at cp
dXdS=0.326302486;          //COS of longitudinal cable angle at cp
CDo=0.1593875390;          //Coeff. of Drag for kite at ref. AOA
atether=15.0;              //Angle of tether at ground (degs)
ath=atether*pi/180.0;      //and in radians
Lth=18000;                 //Length of tether(meters)
ddS=1.0/Lth*cos(asin(dZdS)-ath);//approx. for change in Eta with cable length
Nfg=0.9;                   //Fudge factor for Czq calculation
Ixz=10.0;                  //
Ixx=1500.0;                //
Iyy=1700.0;                //
Izz=2000.0;                //

//Wing
b=15.0;                    //Wing Span (meters)
croot=1.5;                 //Wing root chord (meters)
ctip=1.5;                  //Wing tip chord (meters)
Xls=1.5;                   //distance of wing ac from nose ref. pt.
Zls=0.2;                   //distance of wing ac above nose ref. pt.
als=5.0;                   //Wing liftcurve slope
ils=7.0;                   //Incidence angle of wing (degrees)
Azll=15.0;                 //Zero Lift Line angle for the airfoil (deg)
Gamma=0.0;                 //Wing Dihedral angle in radians
sweep=0.0;                 //Wing 1/4 chord sweep (degrees)
Nls=1.0;                   //ratio of velocity at ls with freestream velocity
Nlsl=0.95;                 //ratio of lateral vel. at ls with freestream lat. vel.
dSiglsdP=0.0;              //dsigma_ls/dP_hat
dSiglsdr=0.0;              //Change in sidewash with yaw rate at wing
taper=ctip/croot;          //Taper ratio of wing
cmean=(ctip+croot)/2.0;    //Mean wing chord (meters)
S=cmean*b;                 //Wing area (square meters)
AR=pow(b,2.0)/S;           //Aspect Ratio
CLo=als*(ils+Thetao+Azll)*pi/180.0;     //Coeff. of lift of the wing at ref. AOA

//Horizontal Stabilizer
Xt=6.0;                    //distance of horizontal stab. mac from nose ref. pt.
Zt=0.1;                    //distance of horizontal stab. mac above nose ref. pt.
at=4.0;                    //Horizontal Stab. lift curve slope (/rads)
bt=3.0;                    //Horizontal Stab. span
ct=0.5;                    //Horizontal Stab. chord
ihs=0.0;                   //Incidence angle of Horizontal Stabilator (degrees)
St=bt*ct;                  //Horizontal Stab. Area
lt=Xt-Xcg;                 //Distance from wing mac to Horizontal Stab. mac
VH=St*lt/S/cmean;          //Horizontal Tail Volume Coefficient
```

```
Xf=[illegible];                   //distance of vertical fin mac from nose ref. pt.
Zf=0.6;                   //distance of vertical fin cp above Nose ref. pt. (+ve above)
af=4.0;                   //Fin lift curve slope  (/rads)
cf=0.5;                   //Fin mean aerodynamic chord
bf=1.25;                  //Fin height
Cdof=0.0006;              //Fin drag coeff. reference
Nf=0.95;                  //ratio of velocity at fin with freestream velocity
dSigmadB=0.0;             //Change in side-wash with sideslipe angle
dSigfdr=0.0;              //Change in sidewash with yaw rate at vertical fin
dSigfdP=0.0;              //dsigma_fin/dP_hat
Sf=cf*bf;                 //Fin area
lf=Xf-Xcg;                //Distance from wing mac to fin mac

//Fuselage
Xcv=1.3;                  //distance of volume center from nose ref. pt.
Ki=2.0;                   //2 for high wing
abody=0.0525;             //Fuselage lateral liftcurve slope /rads
Vol=1.0;                  //Fuselage Volume
Zcpbody=0.1;              //Distance of body cp above nose ref. pt.
Sside=1.0;                //Area of fuselage side (m2)
lbody=6.0;                //Total length of fuselage
hmax=0.5;                 //Fuselage Maximum height
h1=0.45;                  //Fuselage height at 1/4 length
h2=0.15;                  //Fuselage height at 3/4 length
w1=0.22;                  //Fuselage width at 1/4 length
w2=0.1;                   //Fuselage width at 3/4 length
KB=0.257*Xcg/lbody+0.00266667*pow(lbody/hmax,2.0)-0.06*lbody/hmax+0.30733;

//Atmospheric
grav=9.81;                //gravity  (m/s2)
dens=0.38857;             //Air density at altitude (kg/m3)
Kvisc=0.000037060;        //Kinematic Viscosity (m2/s)
Temp=220.0;               //Temperature at given altitude
Uo=8.75;                  //Prevailing Wing velocity (m/s)
dEpsilondq=0.001;         //Change in downwash with change in pitch rate
Mach=Uo/pow((1.4*287.05*Temp),0.5);       //Mach number
Re=Uo*cmean/Kvisc;        //Reynold's number for the wing

//General
CNbCL=0.01;               //CNbeta/pow(Coeff of lift,2)  ->  Etkin or NACA 1098
                          //could refine with an equation in future
CLppf=-0.58;              //(CLp)planform  ->  Etkin or NACA 1098 (pg 20)
Cnp1=-0.08;               // ->  Etkin or NACA 1098
Cnp2=8.5;                 // ->  Etkin or NACA 1098
CDpls=0.011;              //Change in CDo for ls wrt AOA (/degree!!!)
CLrCLo=0.28;              // -> Etkin and Reid pg 351
Cnr1=-0.005;              // ->  Etkin or NACA 1098
Cnr2=-0.3;                // ->  Etkin or NACA 1098
//Change in Downwash with change in alpha
   KA=1.0/AR-1.0/(1.0+pow(AR,1.7));
   Klam=(10.0-3.0*taper)/7.0;
   KH=(1.0-fabs((Zt-Zls)/b))/pow(2.0*lt/b,1.0/3.0);
   dEpsilonda=4.44*pow(KA*Klam*KH*pow(cos(sweep*pi/180.0),0.5),1.19);
CWo=mass*grav*2.0/dens/pow(Uo,2.0)/S;     //Coeff. of Weight
//Coefficient of Vertical tether force at the cp.
   Czcpo=(To*dZdS)/(0.5*dens*pow(Uo,2.0)*S);
//Coefficient of Horizontal tether force at the cp.
   Cxcpo=(To*dXdS)/(0.5*dens*pow(Uo,2.0)*S);
//Coefficient of tether Moment
   Cmcpo=(To*dXdS*(Zcg-Zcp)+To*dZdS*(Xcp-Xcg))/(0.5*dens*pow(Uo,2.0)*S*cmean);


//Calculate the Longitudinal non-dimensional stability derivatives
//U derivatives
//    X
Cxu=0.0;
```

```
//     Z
Czu=0.0;

//     M
Cmu=0.0;

//Alpha Derivatives
//     X
Cxalpha=CLo-CDpls*180.0/pi;

//     Z
Czalpha=-als-CDo;

//     M
Cmalpha=Cxalpha*(Zcg-Znp)/cmean-Czalpha*(Xcg-Xnp)/cmean;

//q derivatives
//     X
Cxalphals=(CLo-CDpls*180.0/pi);
Cxqlsac=Cxalphals;
Cxqls=-Cxalphals*(2.0*(Xcg-Xls)/cmean+dEpsilondq)*Nfg+Cxqlsac*Nfg;

//     Z
Czalphals=-(als+CDo);
Czqlsac=Czalphals;
Czqls=-Czalphals*(2.0*(Xcg-Xls)/cmean+dEpsilondq)*Nfg+Czqlsac*Nfg;
Czqt=-2.0*at*VH;
Czq=Czqls+Czqt;

//     M
Cmqlsac=-pi/4.0*cos(sweep*pi/180.0)*(pow((AR*tan(sweep*pi/180.0)),3)/3.0
   /(AR+6.0*cos(sweep*pi/180.0))+1.0);
Cmqls=Cxqls*(Zcg-Zls)/cmean-Czqls*(Xcg-Xls)/cmean+Cmqlsac;
Cmqt=-2.0*at*VH*lt/cmean;
Cmq=Cmqls+Cmqt;

//alpha_dot derivatives
//     Z
Czadotbody=-2.0*0.75*Vol/S/cmean;
Czadotls=0.0;
Czadott=-2.0*at*VH*dEpsilonda;
Czadot=Czadotbody+Czadotls+Czadott;

//     M
Cmadotbody=-(Xcg-Xcv)/cmean*Czadotbody;
Cmadotls=-(Xcg-Xls)/cmean*Czadotls;
Cmadott=-lt*Czadott;
Cmadot=Cmadotbody+Cmadotls+Cmadott;


//Calculate the Lateral non-dimensional stability derivatives

//Beta derivatives
//     Y
Cybfs=-(af+Cdof)*Sf/S;
Cybf=Nf*(1.0-dSigmadB)*Cybfs;
Cybbody=-Ki*abody*pow(Vol,(2.0/3.0))/S;
Cybls=-0.75*als*pow(sin(Gamma),2.0);
Cyb=Cybf+Cybbody+Cybls;

//     L
CLbf=-(Zcg-Zf)/b*Cybf;
CLbls=0.25*Gamma*CLo;
CLbbody=-(Zcg-Zcpbody)/b*Cybbody;
CLb=CLbf+CLbls+CLbbody;
```

```
CNbbody=-0.96*KB*Sside/S*lbody/b*pow(h1/h2,0.5)*pow(w2/w1,1.0/3.0);
CNbf=lf/b*Cybf;
CNblsac=CNbCL*pow(CLo,2.0);
CNbls=(Xcg-Xls)/b*Cybls+CNblsac;
CNb=CNbbody+CNbf+CNbls;

//P derivatives

//     Y
Cypbody=-2.0*(Zcg-Zcpbody)/b*Cybbody;
Cypf=Cybfs*(-2.2*(Zcg-Zf)/b+dSigfdP)*Nf;
Cyplsac=((AR+cos(sweep*pi/180.0))/(AR+4.0*cos(sweep*pi/180.0))*tan(sweep*pi/180.0)
             +1.0/AR)*CLo;
Cypls=-Cybls*(-2.0*(Zcg-Zls)/b+dSiglsdP)*Nls+Cyplsac;
Cyp=Cypbody+Cypf+Cypls;

//     L
CLpbody=-(Zcg-Zcpbody)/b*Cypbody;
CLpf=-1.1*(Zcg-Zf)/b*Cypf;
CLplsac=CLppf;
CLpls=-(Zcg-Zls)/b*Cypls+CLplsac;
CLp=CLpbody+CLpf+CLpls;

//     N
CNpbody=2.0*(Zcg-Zcpbody)/b*CNbbody;
CNpf=lf/b*Cypf;
CNplsac=(Cnp1*CLo+Cnp2*CDpls);
CNpls=(Xcg-Xls)/b*Cypls+CNplsac;
CNp=CNpbody+CNpf+CNpls;

//R derivatives

//     Y
Cyrfac=-Cybf*cf/b;
Cyrf=Cybfs*(2.0*lf/b+dSigfdr)*Nf+Cyrfac;
Cyrls=Cybls*(2.0*(Xcg-Xls)/b+dSiglsdr)*Nlsl;
Cyr=Cyrf+Cyrls;

//     L
CLrf=-(Zcg-Zf)/b*Cyrf;
cswp=cos(sweep*pi/180.0);
B=pow((1-pow(Mach*cswp,2.0)),0.5);
CLrCL=(1.0+AR*(1.0-pow(B,2.0))/(2.0*AR*(AR*B+2.0*cswp))+(AR*B+2.0*cswp)
   /(AR*B+4.0*cswp)*pow(tan(sweep*pi/180.0),2.0)/8.0)/(1.0+(AR+2.0*cswp)
   /(AR+4.0*cswp)*pow(tan(sweep*pi/180.0),2.0)/8.0)*CLrCLo;
CLrlsac=CLrCL*CLo;
CLrls=-(Zcg-Zls)/b*Cyrls+CLrlsac;
CLr=CLrf+CLrls;

//     N
CNrfac=-pi/4.0*pow(cf/b,2.0)*Sf/S;
CNrf=lf/b*Cyrf+CNrfac;
CNrlsac=(Cnr1*pow(CLo,2.0)+Cnr2*CDpls);
CNrls=(Xcg-Xls)/b*Cyrls+CNrlsac;
CNr=CNrf+CNrls;

//Calculate the Dimensional Stability derivatives

//Longitudinal
//     X
Xu=CWo*dens*Uo*S*sin(Thetao*pi/180.0)-dens*Uo*S*Cxcpo+0.5*dens*Uo*S*Cxu;
Xw=0.5*dens*Uo*S*Cxalpha;

//     Z
Zu=-CWo*dens*Uo*S*cos(Thetao*pi/180.0)-Czcpo*dens*Uo*S+0.5*dens*Uo*S*Czu;
Zw=0.5*dens*Uo*S*Czalpha;
```

```
Zq=0.25*dens*Uo*cmean*S*Czq;


//    M
Mu=-dens*Uo*cmean*S*Cmcpo+0.5*dens*Uo*S*cmean*Cmu;
Mw=0.5*dens*Uo*S*cmean*Cmalpha;
Mwdot=0.25*dens*S*pow(cmean,2.0)*Cmadot;
Mq=0.25*dens*Uo*pow(cmean,2.0)*S*Cmq;

//Lateral
//    Y
Yv=0.5*dens*Uo*S*Cyb;
Yp=0.25*dens*Uo*b*S*Cyp;
Yr=0.25*dens*Uo*b*S*Cyr;

//    L
Lv=0.5*dens*Uo*b*S*CLb;
Lp=0.25*dens*Uo*pow(b,2.0)*S*CLp;
Lr=0.25*dens*Uo*pow(b,2.0)*S*CLr;

//    N
Nv=0.5*dens*Uo*b*S*CNb;
Np=0.25*dens*Uo*pow(b,2.0)*S*CNp;
Nr=0.25*dens*Uo*pow(b,2.0)*S*CNr;

try {                                       // TEST FOR EXCEPTIONS.
   m = new double*[7];                    // STEP 1: SET UP THE ROWS.
   for (int j = 1; j < 7; j++)
       m[j] = new double[7];              // STEP 2: SET UP THE COLUMNS
   er = new double[7];
   ei = new double[7];
   }
catch (xalloc) {
   cout << "Could not allocate.";
   exit(-1);
   }

cout<<"\n\nEigen Value Solver to generate Roots-Locus Plots";

do
{
   cout<<"\n\nOptions:\n";
   cout<<"1.  Change Control gain selection and ranges.\n";
   cout<<"2.  Calculate the Longitudinal Stability Roots for the settings chosen.\n";
   cout<<"3.  Calculate the Lateral Stability Roots for the settings chosen.\n";
   cout<<"0.  Quit\n";
   cout<<"\nPlease enter the number of your choice: ";
   cin>>choice;
   cout<<"\n\n";

   switch(choice)
   {
      case 1:
         select=GainSet(G);
         break;

      case 2:
         if(proceed==choice)
         {
            dest.open(filename,ios::out|ios::app);
            dest<<"\n\n\nLongitudinal Stability Root Calculation\n\n\n";
            dest<<"The following stability roots have G"<<select<<" varying from:\n";
            dest<<G(select,1)<<"\t\n"<<G(select,2)<<"\nin "<<res<<" steps.\n\n";
            flag=0;
            double Tr = Thetao*pi/180.0;
            do
            Ml(1,1)=mass*Ztp*sin(ath)+(mass*Xtp-Mwdot-Xtp*Zwdot)*cos(ath);
            Ml(1,2)=-Iyy-mass*pow(Ztp,2.0)+Xtp*Mwdot+pow(Xtp,2.0)*(Zwdot-mass);
            Ml(2,1)=-dZdS*mass*sin(ath)/To+dXdS*cos(ath)/To*(Zwdot-mass);
            Ml(2,2)=dZdS*Ztp*mass/To+dXdS*Xtp/To*(mass-Zwdot);
            Dl(1,1)=(Mu-Ztp*Xu+Xtp*Zu)*sin(ath)+(Ztp*Xw-Mw-Xtp*Zw)*cos(ath)
                        +Xtp*cos(ath)*G(4,3)+Ztp*sin(ath)*G(2,3);
            Dl(1,2)=Uo*Mwdot+Mq+Xtp*Uo*Zwdot+Xtp*Zq-Ztp*Mu+pow(Ztp,2.0)*Xu
                        -Xtp*Ztp*Zu+Xtp*Mw-Xtp*Ztp*Xw+pow(Xtp,2.0)*Zw+G(6,3)
                        -Ztp*(Ztp*cos(Tr)-Xtp*sin(Tr))*G(2,3)-Xtp*(Xtp*cos(Tr)
                        +Ztp*sin(Tr))*G(4,3);
            Dl(2,1)=dZdS*Xu*sin(ath)/To-dXdS*Zu*sin(ath)/To+dXdS*Zw*cos(ath)/To
                        -dZdS*Xw*cos(ath)/To-(dXdS*cos(ath)*G(4,3)
                        +dZdS*sin(ath)*G(2,3))/To;
            Dl(2,2)=dXdS*Ztp*Zu/To-dZdS*Ztp*Xu/To+dZdS*Xtp*Xw/To-dXdS*Xtp*Zw/To
                        -dXdS*(Uo*Zwdot+Zq)/To+(dZdS*(Ztp*cos(Tr)-Xtp*sin(Tr))
                        *G(2,3)+dXdS*(Xtp*cos(Tr)+Ztp*sin(Tr))*G(4,3))/To;
            Kl(1,1)=Xtp*cos(ath)*G(3,3)+Ztp*sin(ath)*G(1,3);
            Kl(1,2)=Uo*Mw+Ztp*mass*grav*cos(Tr)-Ztp*Uo*Xw+Xtp*Uo*Zw-Xtp*mass
                        *grav*sin(Tr)+G(5,3)-Ztp*(Ztp*cos(Tr)-Xtp*sin(Tr))*G(1,3)
                        -Xtp*(Xtp*cos(Tr)+Ztp*sin(Tr))*G(3,3);
            Kl(2,1)=-(pow(dXdS,2.0)+pow(dZdS,2.0))*ddS-(dXdS*cos(ath)*G(3,3)+
                        dZdS*sin(ath)*G(1,3))/To;
            Kl(2,2)=(dXdS*(mass*grav*sin(Tr)-Uo*Zw)+dZdS*(Uo*Xw-mass*grav*cos(Tr)))
                        /To-pow(dXdS,2.0)-pow(dZdS,2.0)+(dZdS*(Ztp*cos(Tr)-Xtp
                        *sin(Tr))*G(1,3)+dXdS*(Xtp*cos(Tr)+Ztp*sin(Tr))*G(3,3))/To;
            matrix_zero(Mlinv);
            matrix_zero(Dlp);
            matrix_zero(Klp);
            if((invflag=matrix_inv(Ml,Mlinv))!=0)
               nrerror("Mass Matrix Non-invertable!");
            if((invflag=matrix_mult(Mlinv,Dl,Dlp))!=0)
               nrerror("Matrix Multiplication error!");
            if((invflag=matrix_mult(Mlinv,Kl,Klp))!=0)
               nrerror("Matrix Multiplication error!");

            for(i=1; i<=2;i++)
            {
               for(j=1; j<=4;j++)
               {
                  if(j==i+2) m[i][j]=1.0; else m[i][j]=0.0;
               }
            }
            for(i=3; i<=4;i++)
            {
               for(j=1; j<=2;j++)
               {
                  m[i][j]=-Klp(i-2,j);
               }
               for(j=3; j<=4;j++)
               {
                  m[i][j]=-Dlp(i-2,j-2);
               }
            }
            balanc(m,4);
            elmhes(m,4);
            hqr(m,4,er,ei);
            dest<<er[1]<<"\t"<<ei[1]<<"\n";
            dest<<er[2]<<"\t\t"<<ei[2]<<"\n";
            dest<<er[3]<<"\t\t\t"<<ei[3]<<"\n";
            dest<<er[4]<<"\t\t\t\t"<<ei[4]<<"\n";
            G(select,3)+=(G(select,2)-G(select,1))/res;
            flag++;
         } while (flag<=res);
         G(select,3)=G(select,1);
         dest.close();
      } else
```

```
            cout<<"\n\nControl gain parameters are not set yet.\n";
         }
         break;

      case 3:
         if(proceed==choice)
         {
            dest.open(filename,ios::out|ios::app);
            dest<<"\n\n\nLateral Stability Root Calculation\n\n\n";
            dest<<"The following stability roots have G"<<select<<" varying from:\n";
            dest<<G(select,1)<<"\t\n"<<G(select,2)<<"\nin "<<res<<" steps.\n\n";
            flag=0;
            do
            {
               M(1,1)=mass*Xtp;
               M(1,2)=Ixz+mass*Xtp*Ztp;
               M(1,3)=-(Izz+mass*pow(Xtp,2.0));
               M(2,1)=-mass*Ztp;
               M(2,2)=-(mass*pow(Ztp,2.0)+Ixx);
               M(2,3)=Ixz+mass*Xtp*Ztp;
               M(3,1)=-mass;
               M(3,2)=-mass*Ztp;
               M(3,3)=mass*Xtp;
               D(1,1)=Nv-Xtp*Yv-Xtp*G(6,3);
               D(1,2)=Ztp*Nv-Xtp*Ztp*Yv+Np-Xtp*Yp;
               D(1,3)=-Xtp*Nv+pow(Xtp,2.0)*Yv+Nr-Xtp*Yr+G(2,3);
               D(2,1)=Lv+Ztp*Yv+Ztp*G(6,3);
               D(2,2)=Ztp*Lv+pow(Ztp,2.0)*Yv+Lp+Ztp*Yp+G(4,3);
               D(2,3)=Lr+Ztp*Yr-Xtp*Lv-Xtp*Ztp*Yv;
               D(3,1)=Yv+G(6,3);
               D(3,2)=Ztp*Yv+Yp;
               D(3,3)=Yr-Xtp*Yv;
               K(1,1)=-Xtp*G(5,3);
               K(1,2)=-Xtp*mass*grav*cos(Thetao*pi/180.0);
               K(1,3)=Xtp*Yv*Uo-Nv*Uo+G(1,3);
               K(2,1)=Ztp*G(5,3);
               K(2,2)=Ztp*mass*grav*cos(Thetao*pi/180.0)+G(3,3);
               K(2,3)=-(Uo*Lv+Ztp*Uo*Yv);
               K(3,1)=-To*dYdS+G(5,3);
               K(3,2)=To*dZdS+mass*grav*cos(Thetao*pi/180.0);
               K(3,3)=-(To*dXdS+Uo*Yv);
               matrix_zero(Minv);
               matrix_zero(Dp);
               matrix_zero(Kp);
               if((invflag=matrix_inv(M,Minv))!=0)
                  nrerror("Mass Matrix Non-invertable!");
               if((invflag=matrix_mult(Minv,D,Dp))!=0)
                  nrerror("Matrix Multiplication error!");
               if((invflag=matrix_mult(Minv,K,Kp))!=0)
                  nrerror("Matrix Multiplication error!");

               for(i=1; i<=3;i++)
               {
                  for(j=1; j<=6;j++)
                  {
                     if(j==i+3) m[i][j]=1.0; else m[i][j]=0.0;
                  }
               }
               for(i=4; i<=6;i++)
               {
                  for(j=1; j<=3;j++)
                  {
                     m[i][j]=-Kp(i-3,j);
                  }
                  for(j=4; j<=6;j++)
                  {
                  }
                  balanc(m,6);
                  elmhes(m,6);
                  hqr(m,6,er,ei);
                  dest<<er[1]<<"\t"<<ei[1]<<"\n";
                  dest<<er[2]<<"\t\t"<<ei[2]<<"\n";
                  dest<<er[3]<<"\t\t\t"<<ei[3]<<"\n";
                  dest<<er[4]<<"\t\t\t\t"<<ei[4]<<"\n";
                  dest<<er[5]<<"\t\t\t\t\t"<<ei[5]<<"\n";
                  dest<<er[6]<<"\t\t\t\t\t\t"<<ei[6]<<"\n";
                  G(select,3)+=(G(select,2)-G(select,1))/res;
                  flag++;
               } while (flag<=res);
               G(select,3)=G(select,1);
               dest.close();
            } else
            {
               cout<<"\n\nControl gain parameters are not set yet.\n";
            }
            break;

         case 0:
            break;

         default:
            cout<<"\n\nYou have made an invalid selection\n\n";
            break;
      }

   //Loop back to do another calculation
   } while (choice!=0);


   for (i = 1; i <7;  i++)
       delete[] m[i];
   delete[] er;
   delete[] ei;

   exit(0);
}
```

JDD_STRG.H

```
#ifndef _JDD_STRG_H_
#define _JDD_STRG_H_

void balanc(double **a, int n);
void hqr(double **a, int n, double *wr, double *wi);
void zrhqr(double *a, int m, double *rtr, double *rti);
void nrerror(char error_text[]);

#endif
```

JDD_STRG.CPP

```
//Eigen Value Calculation for JDD_Method using the stiff string model

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <complex.h>
#include <float.h>
#include <except.h>
#include <dos.h>
#include <string.h>
#include <iostream.h>
#include <iomanip.h>
#include <fstream.h>
#include "matrix.h"
#include "jdd_strg.h"

const double pi=3.141592654;
int proceed=0;
const double res=20.0;
char *filename = "rootslcs.txt";
fstream dest;

//Function to set which gain will be varied and what the limits are
int GainSet(matrix &G)
{
   int test;
   proceed=0;
   do
   {
      cout<<"\n\nWhich control gain would you like to vary?:\n";
      cout<<"1.  GX\t\t";
      cout<<"2.  GXd\n";
      cout<<"3.  GZ\t\t";
      cout<<"4.  GZd\n";
      cout<<"5.  GM\t\t";
      cout<<"6.  GMd\n\n";
      cout<<"7.  GN\t\t";
      cout<<"8.  GNd\n";
      cout<<"9.  GL\t\t";
      cout<<"10.  GLd\n";
      cout<<"11.  GY\t\t";
      cout<<"12.  GYd\n";
      cout<<"\nPlease enter the number of your choice: ";
      cin>>test;
      cout<<"\n\n";

      switch(test)
      {
            case 2:
            case 3:
            case 4:
            case 5:
            case 6:

               //Longitudinal Control Vector
               //Zero the control gain array
               matrix_zero(G);
               _fpreset();
               cout<<"\n\nPlease enter the lower limit of the chosen gain: ";
               cin>>G(test,1);
               cout<<"\nNow the upper limit of the chosen gain: ";
               cin>>G(test,2);
               G(test,3)=G(test,1);
               proceed=2;
               break;

            case 7:
            case 8:
            case 9:
            case 10:
            case 11:
            case 12:

               //Lateral Control Vector
               //Zero the control gain array
               cout<<"\n\nLateral option not implemented\n\n";
               proceed=0;
/*             matrix_zero(G);

               cout<<"\n\nPlease enter the lower limit of the chosen gain: ";
               cin>>G(test-6,1);
               cout<<"\nNow the upper limit of the chosen gain: ";
               cin>>G(test-6,2);
               G(test-6,3)=G(test-6,1);
               proceed=3;
               test-=6;
*/             break;

            default:
               cout<<"\n\nYou have made an invalid selection\n\n";
               proceed=0;
               break;
         }
   }while(proceed==0);

   return(test);
}


void main()
{
   int i,j,choice,select,invflag,flag;
   matrix G(6,3),PI(21);
   complex *xc;
   double *m,*rtr,*rti, *result;
   double mass,b,S,dens,AR,Xcg,Zcg,Xls,Zls,Xcp,Zcp,Xtp,Ztp,Xf,Ixz,Ixx,Iyy,Izz,Re;
   double Kvisc,Nfg,Azll,dEpsilondq,VH,CWo,Cmcpo,bf,altitude,Lth;
   double grav,Uo,Thetao,To,dZdS,dYdS,dXdS,dSigfdP,sweep,dSigfdr,dSiglsdr;
   double Xnp,Znp,taper,croot,ctip,cmean,cswp,B,Mach,Temp,Sside,dSiglsdP;
   double Nf,Nls,Nlsl,dSigmadB,Ki,Vol,Gamma,Sf,Zf,Zcpbody,KB,lbody,hmax,h1,h2,w1,w2;
   double lf,af,cf,abody,als,ils,ihs,CLo,CDo,CNbCL,Cnp1,Cnp2,CDpls,Cnr1,Cnr2;
   double KA,Klam,KH,dEpsilonda,Xt,Zt,lt,bt,ct,St,at,Xcv,atether,ath;
   double Cdof,Cybfs,Cybf,Cybbody,Cybls,Cyb,CLbf,CLbls,CLbbody,CLb,CNbbody;
   double CNbf,CNblsac,CNbls,CNb,Cypbody,Cypf,Cyplsac,Cypls,Cyp,CLpbody;
double Cyrfac,Cyrf,Cyrls,Cyr,Clrf,Clrcb,Clrcbo,Clrlsac,Clrls,Clr;
double CNrfac,CNrf,CNrlsac,CNrls,CNr,Yv,Yp,Yr,Lv,Lp,Lr,Nv,Np,Nr;
double Cxu,Czu,Cmu,Cxalpha,Czalpha,Cmalpha,Czalphals,Czqlsac,Czqls,Czqt,Czq;
double Cxalphals,Cxqlsac,Cxqls,Czadot,Czadotbody,Czadotls,Czadott;
double Cmadot,Cmadotbody,Cmadotls,Cmadott,Cxcpo,Czcpo,Cmqlsac,Cmqls,Cmqt;
double Cmq,Xu,Xw,Zu,Zw,Zwdot,Zq,Mu,Mw,Mwdot,Mq;

//Define all Kite parameters
//Total Kite
mass=50.0;                  //Aircraft mass (kg)
Xcg=1.5;                    //distance of CG from nose ref. pt.
Zcg=0.05;                   //distance +ve up of CG above nose ref. pt
Xcp=1.31;                   //distance of confluence pt. from nose ref. pt.
Zcp=-0.5;                   //distance of confluence pt. above nose ref. pt. (+ve above)
Xtp=Xcg-Xcp;                //distance of confluence pt. in front of CG
Ztp=Zcg-Zcp;                //distance of confluence pt. below CG
Xnp=1.70;                   //distance of neutral pt. from nose ref. pt.
Znp=0.15;                   //distance of neutral pt. above nose ref. pt. (+ve above)
Thetao=0.0;                 //Aircraft Equilibrium AOA (degrees)
To=163.3;                   //Cable tension at equilibrium (Newtons)
dZdS=0.945265405;           //SIN of longitudinal cable angle at cp
dYdS=0.0;                   //COS of lateral cable angle at cp
dXdS=0.326302486;           //COS of longitudinal cable angle at cp
CDo=0.1593875390;           //Coeff. of Drag for kite at ref. AOA
atether=80.0;               //Angle of tether to the ground (degs)
ath=atether*pi/180.0;       //and in radians
altitude=10000.0;           //altitude of kite (meters)
Lth=altitude/sin(ath);      //Length of tether
Nfg=0.9;                    //Fudge factor for Czq calculation
Ixz=10.0;                   //
Ixx=1500.0;                 //
Iyy=1700.0;                 //
Izz=2000.0;                 //

//Wing
b=15.0;                      //Wing Span (meters)
croot=1.5;                //Wing root chord (meters)
ctip=1.5;                    //Wing tip chord (meters)
Xls=1.5;                  //distance of wing ac from nose ref. pt.
Zls=0.2;                  //distance of wing ac above nose ref. pt.
als=5.0;                  //Wing liftcurve slope
ils=7.0;                  //Incidence angle of wing (degrees)
Azll=15.0;                //Zero Lift Line angle for the airfoil (deg)
Gamma=0.0;                //Wing Dihedral angle in radians
sweep=0.0;                //Wing 1/4 chord sweep (degrees)
Nls=1.0;                  //ratio of velocity at ls with freestream velocity
Nlsl=0.95;                //ratio of lateral vel. at ls with freestream lat. vel.
dSiglsdP=0.0;             //dsigma_ls/dP_hat
dSiglsdr=0.0;         //Change in sidewash with yaw rate at wing
taper=ctip/croot;      //Taper ratio of wing
cmean=(ctip+croot)/2.0;     //Mean wing chord (meters)
S=cmean*b;                //Wing area (square meters)
AR=pow(b,2.0)/S;       //Aspect Ratio
CLo=als*(ils+Thetao+Azll)*pi/180.0;     //Coeff. of lift of the wing at ref. AOA


//Horizontal Stabilizer
Xt=6.0;                      //distance of horizontal stab. mac from nose ref. pt.
Zt=0.1;                      //distance of horizontal stab. mac above nose ref. pt.
at=4.0;                      //Horizontal Stab. lift curve slope (/rads)
bt=3.0;                      //Horizontal Stab. span
ct=0.5;                      //Horizontal Stab. chord
ihs=0.0;                  //Incidence angle of Horizontal Stabilator (degrees)
St=bt*ct;                    //Horizontal Stab. Area
lt=Xt-Xcg;                //Distance from wing mac to Horizontal Stab. mac
VH=St*lt/S/cmean;      //Horizontal Tail Volume Coefficient
```

```
//Vertical Stabilizer
Xf=6.0;                    //distance of vertical fin mac from nose ref. pt.
Zf=0.6;                    //distance of vertical fin cp above Nose ref. pt. (+ve above)
af=4.0;                    //Fin lift curve slope  (/rads)
cf=0.5;                    //Fin mean aerodynamic chord
bf=1.25;                //Fin height
Cdof=0.0006;            //Fin drag coeff. reference
Nf=0.95;                //ratio of velocity at fin with freestream velocity
dSigmadB=0.0;           //Change in side-wash with sideslipe angle
dSigfdr=0.0;            //Change in sidewash with yaw rate at vertical fin
dSigfdP=0.0;            //dsigma_fin/dP_hat
Sf=cf*bf;                  //Fin area
lf=Xf-Xcg;              //Distance from wing mac to fin mac


//Fuselage
Xcv=1.3;                //distance of volume center from nose ref. pt.
Ki=2.0;                    //2 for high wing
abody=0.0525;           //Fuselage lateral liftcurve slope /rads
Vol=1.0;                //Fuselage Volume
Zcpbody=0.1;            //Distance of body cp above nose ref. pt.
Sside=1.0;              //Area of fuselage side (m2)
lbody=6.0;              //Total length of fuselage
hmax=0.5;                  //Fuselage Maximum height
h1=0.45;                //Fuselage height at 1/4 length
h2=0.15;                //Fuselage height at 3/4 length
w1=0.22;                //Fuselage width at 1/4 length
w2=0.1;                    //Fuselage width at 3/4 length
KB=0.257*Xcg/lbody+0.00266667*pow(lbody/hmax,2.0)-0.06*lbody/hmax+0.30733;


//Atmospheric
grav=9.81;              //gravity  (m/s2)
dens=0.38857;           //Air density at altitude (kg/m3)
Kvisc=0.000037060;   //Kinematic Viscosity (m2/s)
Temp=220.0;          //Temperature at given altitude
Uo=8.75;                //Prevailing Wing velocity (m/s)
dEpsilondq=0.001;    //Change in downwash with change in pitch rate
Mach=Uo/pow((1.4*287.05*Temp),0.5);       //Mach number
Re=Uo*cmean/Kvisc;   //Reynold's number for the wing

//General
CNbCL=0.01;             //CNbeta/pow(Coeff of lift,2)  ->  Etkin or NACA 1098
                              //could refine with an equation in future
CLppf=-0.58;            //(CLp)planform  ->  Etkin or NACA 1098 (pg 20)
Cnp1=-0.08;          // ->  Etkin or NACA 1098
Cnp2=8.5;            // ->  Etkin or NACA 1098
CDpls=0.011;            //Change in CDo for ls wrt AOA (/degree!!!)
CLrCLo=0.28;            // -> Etkin and Reid pg 351
Cnr1=-0.005;            // ->  Etkin or NACA 1098
Cnr2=-0.3;              // ->  Etkin or NACA 1098
//Change in Downwash with change in alpha
   KA=1.0/AR-1.0/(1.0+pow(AR,1.7));
   Klam=(10.0-3.0*taper)/7.0;
   KH=(1.0-fabs((Zt-Zls)/b))/pow(2.0*lt/b,1.0/3.0);
   dEpsilonda=4.44*pow(KA*Klam*KH*pow(cos(sweep*pi/180.0),0.5),1.19);
CWo=mass*grav*2.0/dens/pow(Uo,2.0)/S;     //Coeff. of Weight
//Coefficient of Vertical tether force at the cp.
   Czcpo=(To*dZdS)/(0.5*dens*pow(Uo,2.0)*S);
//Coefficient of Horizontal tether force at the cp.
   Cxcpo=(To*dXdS)/(0.5*dens*pow(Uo,2.0)*S);
//Coefficient of tether Moment
   Cmcpo=(To*dXdS*(Zcg-Zcp)+To*dZdS*(Xcp-Xcg))/(0.5*dens*pow(Uo,2.0)*S*cmean);
```

```
//Calculate the Longitudinal non-dimensional stability derivatives
//U derivatives
//    X
Cxu=0.0;

//    Z
Czu=0.0;

//    M
Cmu=0.0;

//Alpha Derivatives
//    X
Cxalpha=CLo-CDpls*180.0/pi;

//    Z
Czalpha=-als-CDo;

//    M
Cmalpha=Cxalpha*(Zcg-Znp)/cmean-Czalpha*(Xcg-Xnp)/cmean;

//q derivatives
//    X
Cxalphals=(CLo-CDpls*180.0/pi);
Cxqlsac=Cxalphals;
Cxqls=-Cxalphals*(2.0*(Xcg-Xls)/cmean+dEpsilondq)*Nfg+Cxqlsac*Nfg;

//    Z
Czalphals=-(als+CDo);
Czqlsac=Czalphals;
Czqls=-Czalphals*(2.0*(Xcg-Xls)/cmean+dEpsilondq)*Nfg+Czqlsac*Nfg;
Czqt=-2.0*at*VH;
Czq=Czqls+Czqt;

//    M
Cmqlsac=-pi/4.0*cos(sweep*pi/180.0)*(pow((AR*tan(sweep*pi/180.0)),3)/3.0
   /(AR+6.0*cos(sweep*pi/180.0))+1.0);
Cmqls=Cxqls*(Zcg-Zls)/cmean-Czqls*(Xcg-Xls)/cmean+Cmqlsac;
Cmqt=-2.0*at*VH*lt/cmean;
Cmq=Cmqls+Cmqt;

//alpha_dot derivatives
//    Z
Czadotbody=-2.0*0.75*Vol/S/cmean;
Czadotls=0.0;
Czadott=-2.0*at*VH*dEpsilonda;
Czadot=Czadotbody+Czadotls+Czadott;

//    M
Cmadotbody=-(Xcg-Xcv)/cmean*Czadotbody;
Cmadotls=-(Xcg-Xls)/cmean*Czadotls;
Cmadott=-lt*Czadott;
Cmadot=Cmadotbody+Cmadotls+Cmadott;


//Calculate the Lateral non-dimensional stability derivatives

//Beta derivatives
//    Y
Cybfs=-(af+Cdof)*Sf/S;
Cybf=Nf*(1.0-dSigmadB)*Cybfs;
Cybbody=-Ki*abody*pow(Vol,(2.0/3.0))/S;
Cybls=-0.75*als*pow(sin(Gamma),2.0);
Cyb=Cybf+Cybbody+Cybls;

//    L
```

```
CLbls=0.25*Gamma*CLo;
CLbbody=-(Zcg-Zcpbody)/b*Cybbody;
CLb=CLbf+CLbls+CLbbody;

//    N
CNbbody=-0.96*KB*Sside/S*lbody/b*pow(h1/h2,0.5)*pow(w2/w1,1.0/3.0);
CNbf=lf/b*Cybf;
CNblsac=CNbCL*pow(CLo,2.0);
CNbls=(Xcg-Xls)/b*Cybls+CNblsac;
CNb=CNbbody+CNbf+CNbls;

//P derivatives

//    Y
Cypbody=-2.0*(Zcg-Zcpbody)/b*Cybbody;
Cypf=Cybfs*(-2.2*(Zcg-Zf)/b+dSigfdP)*Nf;
Cyplsac=((AR+cos(sweep*pi/180.0))/(AR+4.0*cos(sweep*pi/180.0))*tan(sweep*pi/180.0)
            +1.0/AR)*CLo;
Cypls=-Cybls*(-2.0*(Zcg-Zls)/b+dSiglsdP)*Nls+Cyplsac;
Cyp=Cypbody+Cypf+Cypls;

//    L
CLpbody=-(Zcg-Zcpbody)/b*Cypbody;
CLpf=-1.1*(Zcg-Zf)/b*Cypf;
CLplsac=CLppf;
CLpls=-(Zcg-Zls)/b*Cypls+CLplsac;
CLp=CLpbody+CLpf+CLpls;

//    N
CNpbody=2.0*(Zcg-Zcpbody)/b*CNbbody;
CNpf=lf/b*Cypf;
CNplsac=(Cnp1*CLo+Cnp2*CDpls);
CNpls=(Xcg-Xls)/b*Cypls+CNplsac;
CNp=CNpbody+CNpf+CNpls;

//R derivatives

//    Y
Cyrfac=-Cybf*cf/b;
Cyrf=Cybfs*(2.0*lf/b+dSigfdr)*Nf+Cyrfac;
Cyrls=Cybls*(2.0*(Xcg-Xls)/b+dSiglsdr)*Nlsl;
Cyr=Cyrf+Cyrls;

//    L
CLrf=-(Zcg-Zf)/b*Cyrf;
cswp=cos(sweep*pi/180.0);
B=pow((1-pow(Mach*cswp,2.0)),0.5);
CLrCL=(1.0+AR*(1.0-pow(B,2.0))/(2.0*AR*(AR*B+2.0*cswp))+(AR*B+2.0*cswp)
   /(AR*B+4.0*cswp)*pow(tan(sweep*pi/180.0),2.0)/8.0)/(1.0+(AR+2.0*cswp)
   /(AR+4.0*cswp)*pow(tan(sweep*pi/180.0),2.0)/8.0)*CLrCLo;
CLrlsac=CLrCL*CLo;
CLrls=-(Zcg-Zls)/b*Cyrls+CLrlsac;
CLr=CLrf+CLrls;

//    N
CNrfac=-pi/4.0*pow(cf/b,2.0)*Sf/S;
CNrf=lf/b*Cyrf+CNrfac;
CNrlsac=(Cnr1*pow(CLo,2.0)+Cnr2*CDpls);
CNrls=(Xcg-Xls)/b*Cyrls+CNrlsac;
CNr=CNrf+CNrls;


//Calculate the Dimensional Stability derivatives

//Longitudinal
//    X
```

```
Xw=0.5*dens*Uo*S*Cxalpha;

//    Z
Zu=-CWo*dens*Uo*S*cos(Thetao*pi/180.0)-Czcpo*dens*Uo*S+0.5*dens*Uo*S*Czu;
Zw=0.5*dens*Uo*S*Czalpha;
Zwdot=0.25*dens*S*cmean*Czadot;
Zq=0.25*dens*Uo*cmean*S*Czq;

//    M
Mu=-dens*Uo*cmean*S*Cmcpo+0.5*dens*Uo*S*cmean*Cmu;
Mw=0.5*dens*Uo*S*cmean*Cmalpha;
Mwdot=0.25*dens*S*pow(cmean,2.0)*Cmadot;
Mq=0.25*dens*Uo*pow(cmean,2.0)*S*Cmq;


//Lateral
//    Y
Yv=0.5*dens*Uo*S*Cyb;
Yp=0.25*dens*Uo*b*S*Cyp;
Yr=0.25*dens*Uo*b*S*Cyr;

//    L
Lv=0.5*dens*Uo*b*S*CLb;
Lp=0.25*dens*Uo*pow(b,2.0)*S*CLp;
Lr=0.25*dens*Uo*pow(b,2.0)*S*CLr;

//    N
Nv=0.5*dens*Uo*b*S*CNb;
Np=0.25*dens*Uo*pow(b,2.0)*S*CNp;
Nr=0.25*dens*Uo*pow(b,2.0)*S*CNr;

try {                                    // TEST FOR EXCEPTIONS.
   m = new double[5];
   rtr = new double[5];
   rti = new double[5];
   xc = new complex[5];
   result = new double[5];
   }
catch (xalloc) {
   cout << "Could not allocate.";
   exit(-1);
   }

cout<<"\n\nEigen Value Solver to generate Roots-Locus Plots";
cout<<"\nJDD Method - 'String Cable'";

do
{
   cout<<"\n\nOptions:\n";
   cout<<"1.  Change Control gain selection and ranges.\n";
   cout<<"2.  Calculate the Longitudinal Stability Roots for the settings chosen.\n";
   cout<<"3.  Calculate the Lateral Stability Roots for the settings chosen.\n";
   cout<<"0.  Quit\n";
   cout<<"\nPlease enter the number of your choice: ";
   cin>>choice;
   cout<<"\n\n";

   switch(choice)
   {
      case 1:
         select=GainSet(G);
         break;

      case 2:
         if(proceed==choice)
         {
               dest<<"\n\n\nLongitudinal Stability Root Calculation";
               dest<<"\n\n\nThe following stability roots have G"<<select<<" varying
from:\n";
               dest<<G(select,1)<<"\t\n"<<G(select,2)<<"\nin "<<res<<" steps.\n\n";
               flag=0;
               do
               {
                  PI(1)=mass*Lth*sin(ath);
                  PI(2)=-mass*Ztp;
                  PI(3)=(Zwdot-mass)*Lth*cos(ath);
                  PI(4)=Xtp*(mass-Zwdot);
                  PI(5)=Mwdot*Lth*cos(ath);
                  PI(6)=Iyy-Xtp*Mwdot;
                  PI(7)=Lth*(Xw*cos(ath)-Xu*sin(ath)-G(2,3)*sin(ath));
                  PI(8)=Ztp*Xu-Xtp*Xw+Ztp*G(2,3);
                  PI(9)=Lth*(Zw*cos(ath)-Zu*sin(ath)+G(4,3)*cos(ath));
                  PI(10)=Ztp*Zu-Xtp*Zw-mass*Uo-Zq-G(4,3)*Xtp;
                  PI(11)=Mw*Lth*cos(ath)-Mu*Lth*sin(ath);
                  PI(12)=Ztp*Mu-Xtp*Mw-Mq-G(6,3);
                  PI(13)=To*sin(ath)-G(1,3)*Lth*sin(ath);
                  PI(14)=mass*grav*cos(Thetao*pi/180.0)+To*sin(ath)+Ztp*G(1,3);
                  PI(15)=-cos(ath);
                  PI(16)=-To*cos(ath)+G(3,3)*Lth*cos(ath);
                  PI(17)=mass*grav*sin(Thetao*pi/180.0)-To*cos(ath)-G(3,3)*Xtp;
                  PI(18)=-sin(ath);
                  PI(19)=To*(Xtp*cos(ath)+Ztp*sin(ath));
                  PI(20)=To*(Xtp*cos(ath)+Ztp*sin(ath))-G(5,3);
                  PI(21)=Xtp*sin(ath)-Ztp*cos(ath);

                  m[4]=PI(1)*(PI(4)*PI(21)-PI(6)*PI(18))-PI(2)*(PI(3)*PI(21)-
                        PI(5)*PI(18))+PI(15)*(PI(3)*PI(6)-PI(5)*PI(4));
                  m[3]=PI(1)*(PI(10)*PI(21)-PI(12)*PI(18))+PI(7)*(PI(4)*PI(21)-
                        PI(6)*PI(18))-PI(2)*(PI(9)*PI(21)-PI(11)*PI(18))-PI(8)*
                        (PI(3)*PI(21)-PI(5)*PI(18))+PI(15)*(PI(3)*PI(12)+
                        PI(6)*PI(9)-PI(5)*PI(10)-PI(4)*PI(11));
                  m[2]=PI(1)*(PI(17)*PI(21)-PI(18)*PI(20))+PI(7)*(PI(10)*PI(21)-
                        PI(12)*PI(18))+PI(13)*(PI(4)*PI(21)-PI(6)*PI(18))-PI(2)*
                        (PI(16)*PI(21)-PI(18)*PI(19))-PI(8)*(PI(9)*PI(21)-PI(11)*
                        PI(18))-PI(14)*(PI(3)*PI(21)-PI(5)*PI(18))+PI(15)*(PI(3)*
                        PI(20)+PI(9)*PI(12)+PI(6)*PI(16)-PI(5)*PI(17)-PI(10)*PI(11)-
                        PI(4)*PI(19));
                  m[1]=PI(7)*(PI(17)*PI(21)-PI(18)*PI(20))+PI(13)*(PI(10)*PI(21)-
                        PI(12)*PI(18))-PI(8)*(PI(16)*PI(21)-PI(18)*PI(19))-PI(14)*
                        (PI(9)*PI(21)-PI(11)*PI(18))+PI(15)*(PI(9)*PI(20)+PI(12)*
                        PI(16)-PI(11)*PI(17)-PI(10)*PI(19));
                  m[0]=PI(13)*(PI(17)*PI(21)-PI(18)*PI(20))-PI(14)*(PI(16)*PI(21)-
                        PI(18)*PI(19))+PI(15)*(PI(16)*PI(2)-PI(17)*PI(19));

                  zrhqr(m,4,rtr,rti);

                  for(i=1; i<=4;i++)
                  {
                     xc[i]=complex(rtr[i],rti[i]);
                     result[i]=abs(m[0]+xc[i]*(m[1]+xc[i]*(m[2]+xc[i]*(m[3]+xc[i]*m[4]))));
                  }

                  dest<<rtr[1]<<"\t"<<rti[1]<<"\t\t\t\t"<<result[1]<<"\n";
                  dest<<rtr[2]<<"\t\t"<<rti[2]<<"\t\t\t"<<result[2]<<"\n";
                  dest<<rtr[3]<<"\t\t\t"<<rti[3]<<"\t\t"<<result[3]<<"\n";
                  dest<<rtr[4]<<"\t\t\t\t"<<rti[4]<<"\t"<<result[4]<<"\n";
                  G(select,3)+=(G(select,2)-G(select,1))/res;
                  flag++;
               } while (flag<=res);
               G(select,3)=G(select,1);
               dest.close();
            } else
```

```
               cout<<"\n\nControl gain parameters are not set yet.\n";
            }
            break;

         case 3:
            cout<<"\n\nThis option not implemented\n\n";
/*          if(proceed==choice)
            {
               dest.open(filename,ios::out|ios::app);
               dest<<"\n\n\nLateral Stability Root Calculation";
               dest<<"\n\n\nThe following stability roots have G"<<select<<" varying
from:\n";
               dest<<G(select,1)<<"\t\n"<<G(select,2)<<"\nin "<<res<<" steps.\n\n";
               flag=0;
               do
               {
                  M(1,1)=mass*Xtp;
                  M(1,2)=Ixz+mass*Xtp*Ztp;
                  M(1,3)=-(Izz+mass*pow(Xtp,2.0));
                  M(2,1)=-mass*Ztp;
                  M(2,2)=-(mass*pow(Ztp,2.0)+Ixx);
                  M(2,3)=Ixz+mass*Xtp*Ztp;
                  M(3,1)=-mass;
                  M(3,2)=-mass*Ztp;
                  M(3,3)=mass*Xtp;

                  D(1,1)=Nv-Xtp*Yv-Xtp*G(6,3);
                  D(1,2)=Ztp*Nv-Xtp*Ztp*Yv+Np-Xtp*Yp;
                  D(1,3)=-Xtp*Nv+pow(Xtp,2.0)*Yv+Nr-Xtp*Yr+G(2,3);
                  D(2,1)=Lv+Ztp*Yv+Ztp*G(6,3);
                  D(2,2)=Ztp*Lv+pow(Ztp,2.0)*Yv+Lp+Ztp*Yp+G(4,3);
                  D(2,3)=Lr+Ztp*Yr-Xtp*Lv-Xtp*Ztp*Yv;
                  D(3,1)=Yv+G(6,3);
                  D(3,2)=Ztp*Yv+Yp;
                  D(3,3)=Yr-Xtp*Yv;

                  K(1,1)=-Xtp*G(5,3);
                  K(1,2)=-Xtp*mass*grav*cos(Thetao*pi/180.0);
                  K(1,3)=Xtp*Yv*Uo-Nv*Uo+G(1,3);
                  K(2,1)=Ztp*G(5,3);
                  K(2,2)=Ztp*mass*grav*cos(Thetao*pi/180.0)+G(3,3);
                  K(2,3)=-(Uo*Lv+Ztp*Uo*Yv);
                  K(3,1)=-To*dYdS+G(5,3);
                  K(3,2)=To*dZdS+mass*grav*cos(Thetao*pi/180.0);
                  K(3,3)=-(To*dXdS+Uo*Yv);

                  if((invflag=matrix_inv(M,Minv))!=0) nrerror("Mass Matrix Non-invertable!");
                  if((invflag=matrix_mult(Minv,D,Dp))!=0) nrerror("Matrix Multiplication
error!");
                  if((invflag=matrix_mult(Minv,K,Kp))!=0) nrerror("Matrix Multiplication
error!");

                  for(i=1; i<=3;i++)
                  {
                     for(j=1; j<=6;j++)
                     {
                        if(j==i+3) m[i][j]=1.0; else m[i][j]=0.0;
                     }
                  }
                  for(i=4; i<=6;i++)
                  {
                     for(j=1; j<=3;j++)
                     {
                        m[i][j]=-Kp(i-3,j);
                     }
                     for(j=4; j<=6;j++)
```

```
                        m[i][j]=-Dp(i-3,j-3);
                     }
                  }
                  balanc(m,6);
                  elmhes(m,6);
                  hqr(m,6,er,ei);
                  dest<<er[1]<<"\t"<<ei[1]<<"\n";
                  dest<<er[2]<<"\t\t"<<ei[2]<<"\n";
                  dest<<er[3]<<"\t\t\t"<<ei[3]<<"\n";
                  dest<<er[4]<<"\t\t\t\t"<<ei[4]<<"\n";
                  dest<<er[5]<<"\t\t\t\t\t"<<ei[5]<<"\n";
                  dest<<er[6]<<"\t\t\t\t\t\t"<<ei[6]<<"\n";
                  G(select,3)+=(G(select,2)-G(select,1))/res;
                  flag++;
               } while (flag<=res);
               G(select,3)=G(select,1);
               dest.close();
            } else
            {
               cout<<"\n\nControl gain parameters are not set yet.\n";
            }
*/          break;

         case 0:
            break;

         default:
            cout<<"\n\nYou have made an invalid selection\n\n";
            break;
      }

   //Loop back to do another calculation
   } while (choice!=0);


   delete[] m;
   delete[] rtr;
   delete[] rti;
   delete[] xc;
   delete[] result;

   exit(0);
}
```

JDD_MASS.H

```
#ifndef _JDD_MASS_H_
#define _JDD_MASS_H_

void balanc(double **a, int n);
void hqr(double **a, int n, double *wr, double *wi);
void zrhqr(double *a, int m, double *rtr, double *rti);
void nrerror(char error_text[]);

#endif
```

JDD_MASS.CPP

```
//Eigen Value Calculation for JDD_Method using the stiff string model
//but including effects of Cable drag and Cable Inertia

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <complex.h>
#include <float.h>
#include <except.h>
#include <dos.h>
#include <string.h>
#include <iostream.h>
#include <iomanip.h>
#include <fstream.h>
#include "matrix.h"
#include "jdd_mass.h"

const double pi=3.141592654;
int proceed=0;
const double res=20.0;
char *filename = "rootslcs.txt";
fstream dest;

//Function to set which gain will be varied and what the limits are
int GainSet(matrix &G)
{
   int test;
   proceed=0;
   do
   {
      cout<<"\n\nWhich control gain would you like to vary?:\n";
      cout<<"1.  GX\t\t";
      cout<<"2.  GXd\n";
      cout<<"3.  GZ\t\t";
      cout<<"4.  GZd\n";
      cout<<"5.  GM\t\t";
      cout<<"6.  GMd\n\n";
      cout<<"7.  GN\t\t";
      cout<<"8.  GNd\n";
      cout<<"9.  GL\t\t";
      cout<<"10.  GLd\n";
      cout<<"11.  GY\t\t";
      cout<<"12.  GYd\n";
      cout<<"\nPlease enter the number of your choice: ";
      cin>>test;
      cout<<"\n\n";

      switch(test)
```

```
         case 2:
         case 3:
         case 4:
         case 5:
         case 6:

            //Longitudinal Control Vector
            //Zero the control gain array
            matrix_zero(G);
            _fpreset();
            cout<<"\n\nPlease enter the lower limit of the chosen gain: ";
            cin>>G(test,1);
            cout<<"\nNow the upper limit of the chosen gain: ";
            cin>>G(test,2);
            G(test,3)=G(test,1);
            proceed=2;
            break;

         case 7:
         case 8:
         case 9:
         case 10:
         case 11:
         case 12:

            //Lateral Control Vector
            //Zero the control gain array
            cout<<"\n\nLateral option not implemented\n\n";
            proceed=0;
/*          matrix_zero(G);

            cout<<"\n\nPlease enter the lower limit of the chosen gain: ";
            cin>>G(test-6,1);
            cout<<"\nNow the upper limit of the chosen gain: ";
            cin>>G(test-6,2);
            G(test-6,3)=G(test-6,1);
            proceed=3;
            test-=6;
*/          break;

         default:
            cout<<"\n\nYou have made an invalid selection\n\n";
            proceed=0;
            break;
      }
   }while(proceed==0);

   return(test);
}


void main()
{
   int i,j,choice,select,invflag,flag;
   matrix G(6,3),PI(21);
   complex *xc;
   double *m,*rtr,*rti, *result;
   double mass,b,S,dens,AR,Xcg,Zcg,Xls,Zls,Xcp,Zcp,Xtp,Ztp,Xf,Ixz,Ixx,Iyy,Izz,Re;
   double Kvisc,Nfg,Azll,dEpsilondq,VH,CWo,Cmcpo,bf,altitude,Lth;
   double grav,Uo,Thetao,To,dZdS,dYdS,dXdS,dSigfdP,sweep,dSigfdr,dSiglsdr;
   double Xnp,Znp,taper,croot,ctip,cmean,cswp,B,Mach,Temp,Sside,dSiglsdP;
   double Nf,Nls,Nlsl,dSigmadB,Ki,Vol,Gamma,Sf,Zf,Zcpbody,KB,lbody,hmax,h1,h2,w1,w2;
   double lf,af,cf,abody,als,ils,ihs,CLo,CDo,CNbCL,Cnp1,Cnp2,CDpls,Cnr1,Cnr2;
   double KA,Klam,KH,dEpsilonda,Xt,Zt,lt,bt,ct,St,at,Xcv,atether,ath;
   double Cdof,Cybfs,Cybf,Cybbody,Cybls,Cyb,CLbf,CLbls,CLbbody,CLb,CNbbody;
double CLpf,CLpls,CLplsac,CLp,CLppf,CNpbody,CNpf,CNplsac,CNpls,CNp;
double Cyrfac,Cyrf,Cyrls,Cyr,CLrf,CLrCL,CLrCLo,CLrlsac,CLrls,CLr;
double CNrfac,CNrf,CNrlsac,CNrls,CNr,Yv,Yp,Yr,Lv,Lp,Lr,Nv,Np,Nr;
double Cxu,Czu,Cmu,Cxalpha,Czalpha,Cmalpha,Czalphals,Czqlsac,Czqls,Czqt,Czq;
double Cxalphals,Cxqlsac,Cxqls,Czadot,Czadotbody,Czadotls,Czadott;
double Cmadot,Cmadotbody,Cmadotls,Cmadott,Cxcpo,Czcpo,Cmqlsac,Cmqls,Cmqt;
double Cmq,Xu,Xw,Zu,Zw,Zwdot,Zq,Mu,Mw,Mwdot,Mq,densth,thCdx,thrad;

//Define all Kite parameters
//Total Kite
mass=50.0;                  //Aircraft mass (kg)
Xcg=1.5;                    //distance of CG from nose ref. pt.
Zcg=0.05;                   //distance +ve up of CG above nose ref. pt
Xcp=1.31;                   //distance of confluence pt. from nose ref. pt.
Zcp=-0.5;                   //distance of confluence pt. above nose ref. pt. (+ve above)
Xtp=Xcg-Xcp;                //distance of confluence pt. in front of CG
Ztp=Zcg-Zcp;                //distance of confluence pt. below CG
Xnp=1.70;                   //distance of neutral pt. from nose ref. pt.
Znp=0.15;                   //distance of neutral pt. above nose ref. pt. (+ve above)
Thetao=0.0;                 //Aircraft Equilibrium AOA (degrees)
To=163.3;                   //Cable tension at equilibrium (Newtons)
dZdS=0.945265405;           //SIN of longitudinal cable angle at cp
dYdS=0.0;                   //COS of lateral cable angle at cp
dXdS=0.326302486;           //COS of longitudinal cable angle at cp
CDo=0.1593875390;           //Coeff. of Drag for kite at ref. AOA
atether=80.0;               //Angle of tether to the ground (degs)
ath=atether*pi/180.0;       //and in radians
altitude=10000.0;           //altitude of kite (meters)
Lth=altitude/sin(ath);      //Length of tether
densth=0.0016;              //Mass/meter for the tether
thCdx=0.005;                //Cd for X-flow over tether
thrad=0.007;                //Tether Radius
Nfg=0.9;                    //Fudge factor for Czq calculation
Ixz=10.0;                   //
Ixx=1500.0;                 //
Iyy=1700.0;                 //
Izz=2000.0;                 //

//Wing
b=15.0;                     //Wing Span (meters)
croot=1.5;               //Wing root chord (meters)
ctip=1.5;                   //Wing tip chord (meters)
Xls=1.5;                 //distance of wing ac from nose ref. pt.
Zls=0.2;                 //distance of wing ac above nose ref. pt.
als=5.0;                 //Wing liftcurve slope
ils=7.0;                 //Incidence angle of wing (degrees)
Azll=15.0;               //Zero Lift Line angle for the airfoil (deg)
Gamma=0.0;               //Wing Dihedral angle in radians
sweep=0.0;               //Wing 1/4 chord sweep (degrees)
Nls=1.0;                 //ratio of velocity at ls with freestream velocity
Nlsl=0.95;               //ratio of lateral vel. at ls with freestream lat. vel.
dSiglsdP=0.0;            //dsigma_ls/dP_hat
dSiglsdr=0.0;        //Change in sidewash with yaw rate at wing
taper=ctip/croot;     //Taper ratio of wing
cmean=(ctip+croot)/2.0;     //Mean wing chord (meters)
S=cmean*b;               //Wing area (square meters)
AR=pow(b,2.0)/S;      //Aspect Ratio
CLo=als*(ils+Thetao+Azll)*pi/180.0;    //Coeff. of lift of the wing at ref. AOA


//Horizontal Stabilizer
Xt=6.0;                     //distance of horizontal stab. mac from nose ref. pt.
Zt=0.1;                     //distance of horizontal stab. mac above nose ref. pt.
at=4.0;                     //Horizontal Stab. lift curve slope (/rads)
bt=3.0;                     //Horizontal Stab. span
ct=0.5;                     //Horizontal Stab. chord
```

```
                                 [illegible]
St=bt*ct;                          //Horizontal Stab. Area
lt=Xt-Xcg;                      //Distance from wing mac to Horizontal Stab. mac
VH=St*lt/S/cmean;            //Horizontal Tail Volume Coefficient


//Vertical Stabilizer
Xf=6.0;                            //distance of vertical fin mac from nose ref. pt.
Zf=0.6;                            //distance of vertical fin cp above Nose ref. pt. (+ve above)
af=4.0;                            //Fin lift curve slope  (/rads)
cf=0.5;                            //Fin mean aerodynamic chord
bf=1.25;                        //Fin height
Cdof=0.0006;                    //Fin drag coeff. reference
Nf=0.95;                        //ratio of velocity at fin with freestream velocity
dSigmadB=0.0;                   //Change in side-wash with sideslipe angle
dSigfdr=0.0;                    //Change in sidewash with yaw rate at vertical fin
dSigfdP=0.0;                    //dsigma_fin/dP_hat
Sf=cf*bf;                          //Fin area
lf=Xf-Xcg;                      //Distance from wing mac to fin mac


//Fuselage
Xcv=1.3;                        //distance of volume center from nose ref. pt.
Ki=2.0;                            //2 for high wing
abody=0.0525;                   //Fuselage lateral liftcurve slope /rads
Vol=1.0;                        //Fuselage Volume
Zcpbody=0.1;                    //Distance of body cp above nose ref. pt.
Sside=1.0;                      //Area of fuselage side (m2)
lbody=6.0;                      //Total length of fuselage
hmax=0.5;                          //Fuselage Maximum height
h1=0.45;                        //Fuselage height at 1/4 length
h2=0.15;                        //Fuselage height at 3/4 length
w1=0.22;                        //Fuselage width at 1/4 length
w2=0.1;                            //Fuselage width at 3/4 length
KB=0.257*Xcg/lbody+0.00266667*pow(lbody/hmax,2.0)-0.06*lbody/hmax+0.30733;


//Atmospheric
grav=9.81;                      //gravity  (m/s2)
dens=0.38857;                   //Air density at altitude (kg/m3)
Kvisc=0.000037060;              //Kinematic Viscosity (m2/s)
Temp=220.0;                     //Temperature at given altitude
Uo=8.75;                        //Prevailing Wing velocity (m/s)
dEpsilondq=0.001;               //Change in downwash with change in pitch rate
Mach=Uo/pow((1.4*287.05*Temp),0.5);       //Mach number
Re=Uo*cmean/Kvisc;              //Reynold's number for the wing

//General
CNbCL=0.01;                     //CNbeta/pow(Coeff of lift,2)  ->  Etkin or NACA 1098
                                //could refine with an equation in future
CLppf=-0.58;                    //(CLp)planform  ->  Etkin or NACA 1098 (pg 20)
Cnp1=-0.08;                     // ->  Etkin or NACA 1098
Cnp2=8.5;                       // ->  Etkin or NACA 1098
CDpls=0.011;                    //Change in CDo for ls wrt AOA (/degree!!!)
CLrCLo=0.28;                    // -> Etkin and Reid pg 351
Cnr1=-0.005;                    // ->  Etkin or NACA 1098
Cnr2=-0.3;                      // ->  Etkin or NACA 1098
//Change in Downwash with change in alpha
   KA=1.0/AR-1.0/(1.0+pow(AR,1.7));
   Klam=(10.0-3.0*taper)/7.0;
   KH=(1.0-fabs((Zt-Zls)/b))/pow(2.0*lt/b,1.0/3.0);
   dEpsilonda=4.44*pow(KA*Klam*KH*pow(cos(sweep*pi/180.0),0.5),1.19);
CWo=mass*grav*2.0/dens/pow(Uo,2.0)/S;      //Coeff. of Weight
//Coefficient of Vertical tether force at the cp.
   Czcpo=(To*dZdS)/(0.5*dens*pow(Uo,2.0)*S);
//Coefficient of Horizontal tether force at the cp.
   Cxcpo=(To*dXdS)/(0.5*dens*pow(Uo,2.0)*S);
```

```
//Coefficient of tether Moment
   Cmcpo=(To*dXdS*(Zcg-Zcp)+To*dZdS*(Xcp-Xcg))/(0.5*dens*pow(Uo,2.0)*S*cmean);



//Calculate the Longitudinal non-dimensional stability derivatives
//U derivatives
//    X
Cxu=0.0;

//    Z
Czu=0.0;

//    M
Cmu=0.0;

//Alpha Derivatives
//    X
Cxalpha=CLo-CDpls*180.0/pi;

//    Z
Czalpha=-als-CDo;

//    M
Cmalpha=Cxalpha*(Zcg-Znp)/cmean-Czalpha*(Xcg-Xnp)/cmean;

//q derivatives
//    X
Cxalphals=(CLo-CDpls*180.0/pi);
Cxqlsac=Cxalphals;
Cxqls=-Cxalphals*(2.0*(Xcg-Xls)/cmean+dEpsilondq)*Nfg+Cxqlsac*Nfg;

//    Z
Czalphals=-(als+CDo);
Czqlsac=Czalphals;
Czqls=-Czalphals*(2.0*(Xcg-Xls)/cmean+dEpsilondq)*Nfg+Czqlsac*Nfg;
Czqt=-2.0*at*VH;
Czq=Czqls+Czqt;

//    M
Cmqlsac=-pi/4.0*cos(sweep*pi/180.0)*(pow((AR*tan(sweep*pi/180.0)),3)/3.0
   /(AR+6.0*cos(sweep*pi/180.0))+1.0);
Cmqls=Cxqls*(Zcg-Zls)/cmean-Czqls*(Xcg-Xls)/cmean+Cmqlsac;
Cmqt=-2.0*at*VH*lt/cmean;
Cmq=Cmqls+Cmqt;

//alpha_dot derivatives
//    Z
Czadotbody=-2.0*0.75*Vol/S/cmean;
Czadotls=0.0;
Czadott=-2.0*at*VH*dEpsilonda;
Czadot=Czadotbody+Czadotls+Czadott;

//    M
Cmadotbody=-(Xcg-Xcv)/cmean*Czadotbody;
Cmadotls=-(Xcg-Xls)/cmean*Czadotls;
Cmadott=-lt*Czadott;
Cmadot=Cmadotbody+Cmadotls+Cmadott;


//Calculate the Lateral non-dimensional stability derivatives

//Beta derivatives
//    Y
Cybfs=-(af+Cdof)*Sf/S;
Cybf=Nf*(1.0-dSigmadB)*Cybfs;
Cybbody=-Ki*abody*pow(Vol,(2.0/3.0))/S;
```

```
Cyb=Cybf+Cybbody+Cybls;

//     L
CLbf=-(Zcg-Zf)/b*Cybf;
CLbls=0.25*Gamma*CLo;
CLbbody=-(Zcg-Zcpbody)/b*Cybbody;
CLb=CLbf+CLbls+CLbbody;

//     N
CNbbody=-0.96*KB*Sside/S*lbody/b*pow(h1/h2,0.5)*pow(w2/w1,1.0/3.0);
CNbf=lf/b*Cybf;
CNblsac=CNbCL*pow(CLo,2.0);
CNbls=(Xcg-Xls)/b*Cybls+CNblsac;
CNb=CNbbody+CNbf+CNbls;

//P derivatives

//     Y
Cypbody=-2.0*(Zcg-Zcpbody)/b*Cybbody;
Cypf=Cybfs*(-2.2*(Zcg-Zf)/b+dSigfdP)*Nf;
Cyplsac=((AR+cos(sweep*pi/180.0))/(AR+4.0*cos(sweep*pi/180.0))*tan(sweep*pi/180.0)
                +1.0/AR)*CLo;
Cypls=-Cybls*(-2.0*(Zcg-Zls)/b+dSiglsdP)*Nls+Cyplsac;
Cyp=Cypbody+Cypf+Cypls;

//     L
CLpbody=-(Zcg-Zcpbody)/b*Cypbody;
CLpf=-1.1*(Zcg-Zf)/b*Cypf;
CLplsac=CLppf;
CLpls=-(Zcg-Zls)/b*Cypls+CLplsac;
CLp=CLpbody+CLpf+CLpls;

//     N
CNpbody=2.0*(Zcg-Zcpbody)/b*CNbbody;
CNpf=lf/b*Cypf;
CNplsac=(Cnp1*CLo+Cnp2*CDpls);
CNpls=(Xcg-Xls)/b*Cypls+CNplsac;
CNp=CNpbody+CNpf+CNpls;

//R derivatives

//     Y
Cyrfac=-Cybf*cf/b;
Cyrf=Cybfs*(2.0*lf/b+dSigfdr)*Nf+Cyrfac;
Cyrls=Cybls*(2.0*(Xcg-Xls)/b+dSiglsdr)*Nlsl;
Cyr=Cyrf+Cyrls;

//     L
CLrf=-(Zcg-Zf)/b*Cyrf;
cswp=cos(sweep*pi/180.0);
B=pow((1-pow(Mach*cswp,2.0)),0.5);
CLrCL=(1.0+AR*(1.0-pow(B,2.0))/(2.0*AR*(AR*B+2.0*cswp))+(AR*B+2.0*cswp)
   /(AR*B+4.0*cswp)*pow(tan(sweep*pi/180.0),2.0)/8.0)/(1.0+(AR+2.0*cswp)
   /(AR+4.0*cswp)*pow(tan(sweep*pi/180.0),2.0)/8.0)*CLrCLo;
CLrlsac=CLrCL*CLo;
CLrls=-(Zcg-Zls)/b*Cyrls+CLrlsac;
CLr=CLrf+CLrls;

//     N
CNrfac=-pi/4.0*pow(cf/b,2.0)*Sf/S;
CNrf=lf/b*Cyrf+CNrfac;
CNrlsac=(Cnr1*pow(CLo,2.0)+Cnr2*CDpls);
CNrls=(Xcg-Xls)/b*Cyrls+CNrlsac;
CNr=CNrf+CNrls;
```

```
//Longitudinal
//    X
Xu=CWo*dens*Uo*S*sin(Thetao*pi/180.0)-dens*Uo*S*Cxcpo+0.5*dens*Uo*S*Cxu;
Xw=0.5*dens*Uo*S*Cxalpha;

//    Z
Zu=-CWo*dens*Uo*S*cos(Thetao*pi/180.0)-Czcpo*dens*Uo*S+0.5*dens*Uo*S*Czu;
Zw=0.5*dens*Uo*S*Czalpha;
Zwdot=0.25*dens*S*cmean*Czadot;
Zq=0.25*dens*Uo*cmean*S*Czq;

//    M
Mu=-dens*Uo*cmean*S*Cmcpo+0.5*dens*Uo*S*cmean*Cmu;
Mw=0.5*dens*Uo*S*cmean*Cmalpha;
Mwdot=0.25*dens*S*pow(cmean,2.0)*Cmadot;
Mq=0.25*dens*Uo*pow(cmean,2.0)*S*Cmq;


//Lateral
//    Y
Yv=0.5*dens*Uo*S*Cyb;
Yp=0.25*dens*Uo*b*S*Cyp;
Yr=0.25*dens*Uo*b*S*Cyr;

//    L
Lv=0.5*dens*Uo*b*S*CLb;
Lp=0.25*dens*Uo*pow(b,2.0)*S*CLp;
Lr=0.25*dens*Uo*pow(b,2.0)*S*CLr;

//    N
Nv=0.5*dens*Uo*b*S*CNb;
Np=0.25*dens*Uo*pow(b,2.0)*S*CNp;
Nr=0.25*dens*Uo*pow(b,2.0)*S*CNr;

try {                                  // TEST FOR EXCEPTIONS.
   m = new double[5];
   rtr = new double[5];
   rti = new double[5];
   xc = new complex[5];
   result = new double[5];
   }
catch (xalloc) {
   cout << "Could not allocate.";
   exit(-1);
   }

cout<<"\n\nEigen Value Solver to generate Roots-Locus Plots";
cout<<"\nJDD Method - 'String Cable' with inertia and drag";

do
{
   cout<<"\n\nOptions:\n";
   cout<<"1.  Change Control gain selection and ranges.\n";
   cout<<"2.  Calculate the Longitudinal Stability Roots for the settings chosen.\n";
   cout<<"3.  Calculate the Lateral Stability Roots for the settings chosen.\n";
   cout<<"0.  Quit\n";
   cout<<"\nPlease enter the number of your choice: ";
   cin>>choice;
   cout<<"\n\n";

   switch(choice)
   {
      case 1:
         select=GainSet(G);
         break;
```

```
                if(proceed==choice)
                {
                   dest.open(filename,ios::out|ios::app);
                   dest<<"\n\n\nLongitudinal Stability Root Calculation";
                   dest<<"\n\n\nThe following stability roots have G"<<select<<" varying
from:\n";
                   dest<<G(select,1)<<"\t\n"<<G(select,2)<<"\nin "<<res<<" steps.\n\n";
                   flag=0;
                   do
                   {
                      PI(1)=densth*pow(Lth,2.0)*sin(ath)/3.0+mass*Lth*sin(ath);
                      PI(2)=-mass*Ztp;
                      PI(3)=(Zwdot-mass-densth*Lth/3.0)*Lth*cos(ath);
                      PI(4)=Xtp*(mass-Zwdot);
                      PI(5)=Mwdot*Lth*cos(ath)+densth*pow(Lth,2.0)/3.0
                              *(Ztp*sin(ath)+Xtp*cos(ath));
                      PI(6)=Iyy-Xtp*Mwdot;
                      PI(7)=Lth*(Xw*cos(ath)-Xu*sin(ath)-G(2,3)*sin(ath)+2.0/3.0
                              *thCdx*dens*thrad*Uo*Lth*pow(sin(ath),2.0));
                      PI(8)=Ztp*Xu-Xtp*Xw+Ztp*G(2,3);
                      PI(9)=Lth*(Zw*cos(ath)-Zu*sin(ath)+G(4,3)*cos(ath)-
                              2.0/3.0*thCdx*dens*thrad*Uo*Lth*sin(ath)*cos(ath));
                      PI(10)=Ztp*Zu-Xtp*Zw-mass*Uo-Zq-G(4,3)*Xtp;
                      PI(11)=Lth*(Mw*cos(ath)-Mu*sin(ath)+2.0/3.0*thCdx*dens*thrad
                              *Uo*Lth*sin(ath)*(Ztp*sin(ath)+Xtp*cos(ath)));
                      PI(12)=Ztp*Mu-Xtp*Mw-Mq-G(6,3);
                      PI(13)=To*sin(ath)-G(1,3)*Lth*sin(ath);
                      PI(14)=mass*grav*cos(Thetao*pi/180.0)+To*sin(ath)+Ztp*G(1,3);
                      PI(15)=-cos(ath);
                      PI(16)=-To*cos(ath)+G(3,3)*Lth*cos(ath);
                      PI(17)=mass*grav*sin(Thetao*pi/180.0)-To*cos(ath)-G(3,3)*Xtp;
                      PI(18)=-sin(ath);
                      PI(19)=To*(Xtp*cos(ath)+Ztp*sin(ath));
                      PI(20)=To*(Xtp*cos(ath)+Ztp*sin(ath))-G(5,3);
                      PI(21)=Xtp*sin(ath)-Ztp*cos(ath);

                      m[4]=PI(1)*(PI(4)*PI(21)-PI(6)*PI(18))-PI(2)*(PI(3)*PI(21)-
                            PI(5)*PI(18))+PI(15)*(PI(3)*PI(6)-PI(5)*PI(4));
                      m[3]=PI(1)*(PI(10)*PI(21)-PI(12)*PI(18))+PI(7)*(PI(4)*PI(21)-
                            PI(6)*PI(18))-PI(2)*(PI(9)*PI(21)-PI(11)*PI(18))-PI(8)*
                            (PI(3)*PI(21)-PI(5)*PI(18))+PI(15)*(PI(3)*PI(12)+
                            PI(6)*PI(9)-PI(5)*PI(10)-PI(4)*PI(11));
                      m[2]=PI(1)*(PI(17)*PI(21)-PI(18)*PI(20))+PI(7)*(PI(10)*PI(21)-
                            PI(12)*PI(18))+PI(13)*(PI(4)*PI(21)-PI(6)*PI(18))-PI(2)*
                            (PI(16)*PI(21)-PI(18)*PI(19))-PI(8)*(PI(9)*PI(21)-PI(11)*
                            PI(18))-PI(14)*(PI(3)*PI(21)-PI(5)*PI(18))+PI(15)*(PI(3)*
                            PI(20)+PI(9)*PI(12)+PI(6)*PI(16)-PI(5)*PI(17)-PI(10)*PI(11)-
                            PI(4)*PI(19));
                      m[1]=PI(7)*(PI(17)*PI(21)-PI(18)*PI(20))+PI(13)*(PI(10)*PI(21)-
                            PI(12)*PI(18))-PI(8)*(PI(16)*PI(21)-PI(18)*PI(19))-PI(14)*
                            (PI(9)*PI(21)-PI(11)*PI(18))+PI(15)*(PI(9)*PI(20)+PI(12)*
                            PI(16)-PI(11)*PI(17)-PI(10)*PI(19));
                      m[0]=PI(13)*(PI(17)*PI(21)-PI(18)*PI(20))-PI(14)*(PI(16)*PI(21)-
                            PI(18)*PI(19))+PI(15)*(PI(16)*PI(2)-PI(17)*PI(19));

                      zrhqr(m,4,rtr,rti);

                      for(i=1; i<=4;i++)
                      {
                         xc[i]=complex(rtr[i],rti[i]);
                         result[i]=abs(m[0]+xc[i]*(m[1]+xc[i]*(m[2]+xc[i]*(m[3]+xc[i]*m[4]))));
                      }

                      dest<<rtr[1]<<"\t"<<rti[1]<<"\t\t\t\t"<<result[1]<<"\n";
                      dest<<rtr[2]<<"\t\t"<<rti[2]<<"\t\t\t"<<result[2]<<"\n";
```

```
                  G(select,3)+=(G(select,2)-G(select,1))/res;
                  flag++;
               } while (flag<=res);
               G(select,3)=G(select,1);
               dest.close();
            } else
            {
               cout<<"\n\nControl gain parameters are not set yet.\n";
            }
            break;

         case 3:
            cout<<"\n\nThis option not implemented\n\n";
/*          if(proceed==choice)
            {
               dest.open(filename,ios::out|ios::app);
               dest<<"\n\n\nLateral Stability Root Calculation";
               dest<<"\n\n\nThe following stability roots have G"<<select<<" varying
from:\n";
               dest<<G(select,1)<<"\t\n"<<G(select,2)<<"\nin "<<res<<" steps.\n\n";
               flag=0;
               do
               {
                  M(1,1)=mass*Xtp;
                  M(1,2)=Ixz+mass*Xtp*Ztp;
                  M(1,3)=-(Izz+mass*pow(Xtp,2.0));
                  M(2,1)=-mass*Ztp;
                  M(2,2)=-(mass*pow(Ztp,2.0)+Ixx);
                  M(2,3)=Ixz+mass*Xtp*Ztp;
                  M(3,1)=-mass;
                  M(3,2)=-mass*Ztp;
                  M(3,3)=mass*Xtp;

                  D(1,1)=Nv-Xtp*Yv-Xtp*G(6,3);
                  D(1,2)=Ztp*Nv-Xtp*Ztp*Yv+Np-Xtp*Yp;
                  D(1,3)=-Xtp*Nv+pow(Xtp,2.0)*Yv+Nr-Xtp*Yr+G(2,3);
                  D(2,1)=Lv+Ztp*Yv+Ztp*G(6,3);
                  D(2,2)=Ztp*Lv+pow(Ztp,2.0)*Yv+Lp+Ztp*Yp+G(4,3);
                  D(2,3)=Lr+Ztp*Yr-Xtp*Lv-Xtp*Ztp*Yv;
                  D(3,1)=Yv+G(6,3);
                  D(3,2)=Ztp*Yv+Yp;
                  D(3,3)=Yr-Xtp*Yv;

                  K(1,1)=-Xtp*G(5,3);
                  K(1,2)=-Xtp*mass*grav*cos(Thetao*pi/180.0);
                  K(1,3)=Xtp*Yv*Uo-Nv*Uo+G(1,3);
                  K(2,1)=Ztp*G(5,3);
                  K(2,2)=Ztp*mass*grav*cos(Thetao*pi/180.0)+G(3,3);
                  K(2,3)=-(Uo*Lv+Ztp*Uo*Yv);
                  K(3,1)=-To*dYdS+G(5,3);
                  K(3,2)=To*dZdS+mass*grav*cos(Thetao*pi/180.0);
                  K(3,3)=-(To*dXdS+Uo*Yv);

                  if((invflag=matrix_inv(M,Minv))!=0) nrerror("Mass Matrix Non-invertable!");
                  if((invflag=matrix_mult(Minv,D,Dp))!=0) nrerror("Matrix Multiplication
error!");
                  if((invflag=matrix_mult(Minv,K,Kp))!=0) nrerror("Matrix Multiplication
error!");

                  for(i=1; i<=3;i++)
                  {
                     for(j=1; j<=6;j++)
                     {
                        if(j==i+3) m[i][j]=1.0; else m[i][j]=0.0;
                     }
```

```
                       for(i=4; i<=6;i++)
                       {
                          for(j=1; j<=3;j++)
                          {
                             m[i][j]=-Kp(i-3,j);
                          }
                          for(j=4; j<=6;j++)
                          {
                             m[i][j]=-Dp(i-3,j-3);
                          }
                       }
                       balanc(m,6);
                       elmhes(m,6);
                       hqr(m,6,er,ei);
                       dest<<er[1]<<"\t"<<ei[1]<<"\n";
                       dest<<er[2]<<"\t\t"<<ei[2]<<"\n";
                       dest<<er[3]<<"\t\t\t"<<ei[3]<<"\n";
                       dest<<er[4]<<"\t\t\t\t"<<ei[4]<<"\n";
                       dest<<er[5]<<"\t\t\t\t\t"<<ei[5]<<"\n";
                       dest<<er[6]<<"\t\t\t\t\t\t"<<ei[6]<<"\n";
                       G(select,3)+=(G(select,2)-G(select,1))/res;
                       flag++;
                    } while (flag<=res);
                    G(select,3)=G(select,1);
                    dest.close();
                 } else
                 {
                    cout<<"\n\nControl gain parameters are not set yet.\n";
                 }
*/               break;

              case 0:
                 break;

              default:
                 cout<<"\n\nYou have made an invalid selection\n\n";
                 break;
           }

     //Loop back to do another calculation
     } while (choice!=0);


     delete[] m;
     delete[] rtr;
     delete[] rti;
     delete[] xc;
     delete[] result;

     exit(0);
  }
```

Analyses:
- Trim-state Analysis
- Discrete-element Stability Analysis
- Time-step Integration Analysis

MATRIX.H

```
//*******************************************************************
//******         Matrix Class Declaration  Ver. 0.90          ******
//******                  March 4, 1997                       ******
//******                                                      ******
//******               USE AT YOUR OWN RISK!!!                ******
//******               USE AT YOUR OWN RISK!!!                ******
//******               USE AT YOUR OWN RISK!!!                ******
//******                                                      ******
//******    This file includes integer, double, and complex   ******
//******    matrix capabilities.  For examples on how to use  ******
//******    these classes see the example source files.       ******
//******                                                      ******
//******              by Joeleff Fitzsimmons                  ******
//******   Univ. of Toronto, Institute for Aerospace Studies  ******
//******            jfitzsim@utias.utoronto.ca                ******
//******    http://aerodyn.utias.utoronto.ca/html/index.htm   ******
//******                                                      ******
//******          Developed in Borland C++ Ver. 4.52          ******
//*******************************************************************

#include <complex.h>
#include <fstream.h>
#ifndef _MATRIX_H_
#define _MATRIX_H_

class matrixi
{
public:
   int m,n,init;            //m rows by n columns
   int *mtx_data;                    //Pointer to Matrix data
   matrixi(int r=0, int c=1);          //matrix constructor
   ~matrixi(void);                     //matrix destructor
   //Overload the () operator to ref. cells in a matrix
   int &operator()(int r, int c);
   //Overload the () operator to ref. cells in a vector
   int &operator()(int r);
   //Overload the = operator to copy matrices
   void operator=(matrixi &mtx);
};

void matrix_zero(matrixi &mtx);

class matrix
{
public:
   int m,n,init;            //m rows by n columns
   double *mtx_data;                 //Pointer to Matrix data
   matrix(int r=0, int c=1);         //matrix constructor
   ~matrix(void);                      //matrix destructor
   void Resize(int r=0, int c=1, int useprev=0);
                                       //Resize the matrix to new dimensions
                                       //and reset all the cells to zero
                                          //If useprev==1 use prev. info
   //Overload the () operator to ref. cells in a matrix
   double &operator()(int r, int c);
   //Overload the () operator to ref. cells in a vector
   //Overload the = operator to copy matrices
   void operator=(matrix &mtx);

   friend ostream& operator <<(ostream& os, matrix& MM);
   friend istream& operator >>(istream& is, matrix& MM);
};

void matrix_zero(matrix &mtx);

int matrix_I(matrix &mtx);

int matrix_sum(matrix &mtxl, matrix &mtxr, matrix &sum);

int matrix_sum(double s, matrix &mtx, matrix &sum);

int matrix_mult(matrix &mtxl, matrix &mtxr, matrix &prod);

int matrix_mult(double m, matrix &mtx, matrix &prod);

int matrix_inv(matrix &mtx, matrix &inv);

int matrix_trans(matrix &mtx, matrix &mtxt);

class matrixc
{
public:
   int m,n,init;           //m rows by n columns
   complex *mtx_data;                //Pointer to Matrix data
   matrixc(int r=0, int c=1);           //matrix constructor
   ~matrixc(void);                      //matrix destructor
   void Resize(int r=0, int c=1, int useprev=0);
                                        //Resize the matrix to new dimensions
         //and reset all the cells to zero
         //If useprev==1 use prev. info
   //Overload the () operator to ref. cells in a matrix
   complex &operator()(int r, int c);
   //Overload the () operator to ref. cells in a vector
   complex &operator()(int r);
   //Overload the * operator to multiply two complex matrices
   //matrixc &operator*(matrixc &mtx);
   //Overload the = operator to copy matrices
   void operator=(matrixc &mtx);
   //Overload the = operator to copy a double matrix to a complex matrix
   void operator=(matrix &mtx);
};

void matrix_zero(matrixc &mtx);

int matrix_I(matrixc &mtx);

int matrix_sum(matrix &mtxl, matrixc &mtxr, matrixc &sum);

int matrix_sum(matrixc &mtxl, matrix &mtxr, matrixc &sum);

int matrix_sum(matrixc &mtxl, matrixc &mtxr, matrixc &sum);

int matrix_sum(double s, matrixc &mtx, matrixc &sum);

int matrix_sum(complex s, matrixc &mtx, matrixc &sum);

int matrix_mult(matrix &mtxl, matrixc &mtxr, matrixc &prod);

int matrix_mult(matrixc &mtxl, matrix &mtxr, matrixc &prod);

int matrix_mult(matrixc &mtxl, matrixc &mtxr, matrixc &prod);
int matrix_mult(double m, matrixc &mtx, matrixc &prod);

int matrix_mult(complex m, matrix &mtx, matrixc &prod);

int matrix_mult(complex m, matrixc &mtx, matrixc &prod);

int matrix_inv(matrixc &mtx, matrixc &inv);

int matrix_trans(matrixc &mtx, matrixc &mtxt);

#endif
```

```
//*****************************************************************
//******          Matrix Class Declaration  Ver. 0.90       ******
//******                   March 4, 1997                    ******
//******                                                    ******
//******                USE AT YOUR OWN RISK!!!             ******
//******                USE AT YOUR OWN RISK!!!             ******
//******                USE AT YOUR OWN RISK!!!             ******
//******                                                    ******
//******    This file includes integer, double, and complex ******
//******    matrix capabilities.  For examples on how to use ******
//******    these classes see the example source files.     ******
//******                                                    ******
//******              by Joeleff Fitzsimmons                ******
//******   Univ. of Toronto, Institute for Aerospace Studies ******
//******             jfitzsim@utias.utoronto.ca             ******
//******    http://aerodyn.utias.utoronto.ca/html/index.htm ******
//******                                                    ******
//******           Developed in Borland C++ Ver. 4.52       ******
//*****************************************************************

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <float.h>
#include <complex.h>
#include <except.h>
#include <iostream.h>
#include <iomanip.h>
#include "matrix.h"

//*****************************************************************
//******              Integer Matrix operations             ******●
//*****************************************************************

//Initialize the memory space for the array of the specified size
matrixi::matrixi(int r, int c)
{
   init=0;
   if(r!=0)
   {
      try
      {
         m=r;n=c;
         mtx_data = new int[m*n];
      }
      catch (xalloc)
      {
         cout << "Could not allocate.";
         exit(-1);
      }

      //Set all cells in the matrix to zero
      for(int i=1; i<=m;i++)
      {
         for(int j=1; j<=n;j++)
         {
            mtx_data[(i-1)*n+j-1]=0;
         }
      }
      init=1;
   }
}

//Clean up after the matrix is not needed
matrixi::~matrixi(void)
```

```
{
   if(init)
      delete[] mtx_data;
}

//Overload the () operators to initialize the memory space for a matrix or
//to return a reference to the requested cell in a matrix
//If init=0 then the array hasn't been initialized yet otherwise return a value
int &matrixi::operator()(int r, int c)
{
   if(init==0)
   {
      try
      {
         m=r;n=c;
         mtx_data = new int[m*n];
      }
      catch (xalloc)
      {
         cout << "Could not allocate.";
         exit(-1);
      }

      //Set all cells in the matrix to zero
      for(int i=1; i<=m;i++)
      {
         for(int j=1; j<=n;j++)
         {
            mtx_data[(i-1)*n+j-1]=0;
         }
      }
      init=1;
      return mtx_data[0];
   }
   else
   {
      return mtx_data[(r-1)*n+c-1];
   }
}

//Overload the () operators to initialize the memory space for a vector or
//to return a reference to the requested cell in a vector
//If init=0 then the array hasn't been initialized yet otherwise return a value
int &matrixi::operator()(int r)
{
   if(init==0)
   {
      try
      {
         m=r;n=1;
         mtx_data = new int[m];
      }
      catch (xalloc)
      {
         cout << "Could not allocate.";
         exit(-1);
      }

      //Set all cells in the matrix to zero
      for(int i=1; i<=m;i++)
      {
         mtx_data[(i-1)*n]=0;
      }
      init=1;
      return mtx_data[0];
   }
   else
```

```
      return mtx_data[i-1];
   }
}

//Overload the = operator to copy two matrices
void matrixi::operator=(matrixi &mtx)
{
   if(m==mtx.m&&n==mtx.n)
   {
      for(int i=1;i<=m;i++)
      {
         for(int j=1;j<=n;j++)
         {
            mtx_data[(i-1)*n+j-1]=mtx.mtx_data[(i-1)*n+j-1];
         }
      }
   }
   else
   {
      exit(-1);
   }
}

//Zero all the elements in the matrix
void matrix_zero(matrixi &mtx)
{
   for(int i=1;i<=mtx.m;i++)
   {
      for(int j=1;j<=mtx.n;j++)
      {
         mtx(i,j)=0;
      }
   }
}

//*******************************************************************
//******                Double Matrix operations               ******
//*******************************************************************

//Initialize the memory space for the array of the specified size
matrix::matrix(int r, int c)
{
   init=0;
   if(r!=0)
   {
      try
      {
         m=r;n=c;
         mtx_data = new double[m*n];
      }
      catch (xalloc)
      {
         cout << "Could not allocate.";
         exit(-1);
      }

      //Set all cells in the matrix to zero
      for(int i=1; i<=m;i++)
      {
         for(int j=1; j<=n;j++)
         {
            mtx_data[(i-1)*n+j-1]=0.0;
         }
      }
      init=1;
   }
```

```
//Clean up after the matrix is not needed
matrix::~matrix(void)
{
   if(init)
      delete[] mtx_data;
}

//reset the matrix to new dimensions and initialize to zero
void
matrix::Resize(int r, int c, int useprev)
{
   int mp,np,initp,i;
   double *mtx_databak;
   initp=init;

   //If useprev==1 save the original info for later use
   if(useprev&&init==1)
   {
      try
      {
         mp=m;np=n;
         mtx_databak = new double[mp*np];
      }
      catch (xalloc)
      {
         cout << "Could not allocate.";
         exit(-1);
      }

      //copy original data to backup array
      for(i=0;i<m*n;i++)
      {
         mtx_databak[i] = mtx_data[i];
      }
   }

   //delete the previous array if it was there
   if(init)
      delete[] mtx_data;

   init=0;
   if(r!=0)
   {
      try
      {
         m=r;n=c;
         mtx_data = new double[m*n];
      }
      catch (xalloc)
      {
         cout << "Could not allocate.";
         exit(-1);
      }

      if(useprev&&initp==1)
      {
         //Set all cells in the matrix to prev. value if possible
         for(int i=1; i<=m;i++)
         {
            for(int j=1; j<=n;j++)
            {
               if(i<=mp&&j<=np)
                  mtx_data[(i-1)*n+j-1]=mtx_databak[(i-1)*n+j-1];
               else
                  mtx_data[(i-1)*n+j-1]=0.0;
            delete[] mtx_databak;
        }
        else
        {
            //Set all cells in the matrix to zero
            for(int i=1; i<=m;i++)
            {
                for(int j=1; j<=n;j++)
                {
                    mtx_data[(i-1)*n+j-1]=0.0;
                }
            }
        }
        init=1;
    }
}

//Overload the () operators to initialize the memory space for a matrix or
//to return a reference to the requested cell in a matrix
//If init=0 then the array hasn't been initialized yet otherwise return a value
double &matrix::operator()(int r, int c)
{
    if(init==0)
    {
        try
        {
            m=r;n=c;
            mtx_data = new double[m*n];
        }
        catch (xalloc)
        {
            cout << "Could not allocate.";
            exit(-1);
        }

        //Set all cells in the matrix to zero
        for(int i=1; i<=m;i++)
        {
            for(int j=1; j<=n;j++)
            {
                mtx_data[(i-1)*n+j-1]=0.0;
            }
        }
        init=1;
        return mtx_data[0];
    }
    else
    {
        return mtx_data[(r-1)*n+c-1];
    }
}

//Overload the () operators to initialize the memory space for a vector or
//to return a reference to the requested cell in a vector
//If init=0 then the array hasn't been initialized yet otherwise return a value
double &matrix::operator()(int r)
{
    if(init==0)
    {
        try
        {
            m=r;n=1;
            mtx_data = new double[m];
        }
        catch (xalloc)
```

```
            cout << "Could not allocate.";
            exit(-1);
         }

         //Set all cells in the matrix to zero
         for(int i=1; i<=m;i++)
         {
            mtx_data[(i-1)*n]=0.0;
         }
         init=1;
         return mtx_data[0];
      }
      else
      {
         return mtx_data[r-1];
      }
   }

   //Overload the = operator to copy two matrices
   void matrix::operator=(matrix &mtx)
   {
      if(m==mtx.m&&n==mtx.n)
      {
         for(int i=1;i<=m;i++)
         {
            for(int j=1;j<=n;j++)
            {
               if(fabs(mtx.mtx_data[(i-1)*n+j-1])>2.2e-16)
                  mtx_data[(i-1)*n+j-1]=mtx.mtx_data[(i-1)*n+j-1];
               else
                  mtx_data[(i-1)*n+j-1]=0.0;
            }
         }
      }
      else
      {
         exit(-1);
      }
   }

   ostream&
   operator <<(ostream& os, matrix& MM)
   {
      _fpreset();

      for(int i=1;i<=MM.m;i++)
      {
         for(int j=1;j<=MM.n;j++)
         {
            os << MM(i,j) << ",";
         }
         os << "\n";
      }
      os << "\n";

      // return the stream object
      return os;
   }

   istream&
   operator >>(istream& is, matrix& MM)
   {
      _fpreset();
      char c;
      for(int i=1;i<=MM.m;i++)
      {
         {
            is >> MM(i,j);
            is >> c;
         }
      }

     // return the stream object
     return is;
   }

   //Zero all the elements in the matrix
   void matrix_zero(matrix &mtx)
   {
      for(int i=1;i<=mtx.m;i++)
      {
         for(int j=1;j<=mtx.n;j++)
         {
            mtx(i,j)=0.0;
         }
      }
   }

   //Set the matrix equal to an Identity matrix
   //returns 0 if successful and 1 if the matrix is not a square matrix
   int matrix_I(matrix &mtx)
   {
      if(mtx.m==mtx.n)
      {
         for(int i=1;i<=mtx.m;i++)
         {
            for(int j=1;j<=mtx.n;j++)
            {
               mtx(i,j)=0.0;
            }
            mtx(i,i)=1.0;
         }
         return(0);
      }
      else
      {
         return(1);
      }
   }

   //Add two matrices together
   //returns 0 if successful and 1 if the matrix sizes are not compatible
   int matrix_sum(matrix &mtxl, matrix &mtxr, matrix &sum)
   {
      if(mtxl.m==mtxr.m&&mtxr.m==sum.m&&mtxl.n==mtxr.n&&mtxr.n==sum.n)
      {
         for(int i=1;i<=mtxl.m;i++)
         {
            for(int j=1;j<=mtxl.n;j++)
            {
               sum(i,j)=mtxl(i,j)+mtxr(i,j);
            }
         }
         return(0);
      }
      else
      {
         return(1);
      }
   }

   //Add a constant to a matrix
```

```
int matrix_sum(double s, matrix &mtx, matrix &sum)
{
   if(mtx.m==sum.m&&mtx.n==sum.n)
   {
      for(int i=1;i<=mtx.m;i++)
      {
         for(int j=1;j<=mtx.n;j++)
         {
            sum(i,j)=s+mtx(i,j);
         }
      }
      return(0);
   }
   else
   {
      return(1);
   }
}

//Multiply two matrices together
//returns 0 if successful and 2 if the matrix sizes are not compatible
int matrix_mult(matrix &mtxl, matrix &mtxr, matrix &prod)
{
   if(mtxl.n==mtxr.m&&mtxl.m==prod.m&&mtxr.n==prod.n)
   {
      // Zero Product matrix before calculating
      matrix_zero(prod);

      for(int i=1;i<=mtxl.m;i++)
      {
         for(int j=1;j<=mtxr.n;j++)
         {
            for(int k=1;k<=mtxl.n;k++)
            {
               prod(i,j)+=mtxl(i,k)*mtxr(k,j);
            }
         }
      }
      return(0);
   }
   else
   {
      return(2);
   }
}

//Multiply a matrix by a constant
//returns 0 if successful and 2 if the matrix sizes are not compatible
int matrix_mult(double m, matrix &mtx, matrix &prod)
{
   if(mtx.m==prod.m&&mtx.n==prod.n)
   {
      for(int i=1;i<=mtx.m;i++)
      {
         for(int j=1;j<=mtx.n;j++)
         {
            prod(i,j)=m*mtx(i,j);
         }
      }
      return(0);
   }
   else
   {
      return(2);
   }
}
```

```
//Calculate the inverse of a given matrix
//return F=0:     Inverse was successful
//return F=3:     Matrix is non-invertable
//return F=4:     The matrix sizes are incompatable
int matrix_inv(matrix &mtx, matrix &inv)
{
   int i,j,F,X,Y,K;
   double Cmax,test,L;
   F=0;        //Flag to monitor the matrix singularity
   X=Y=1;      //Start at top left of matrix
   if(mtx.m==mtx.n&&mtx.m==inv.m&&inv.m==inv.n)
   {
      //Create the Identity matrix
      matrix I(mtx.m,mtx.n);
      for(i=1;i<=I.m;i++)
      {
         I(i,i)=1.0;
      }


      //Copy elements from mtx to inv
      inv=mtx;

      while(X<=inv.m&&Y<=inv.n)
      {
         Cmax=0.0;K=X;
         for(i=X;i<=inv.m;i++)
         {
            test=fabs(inv(i,Y));
            if(test>Cmax)
            {
               Cmax=test;
               K=i;
            }
         }

         if(Cmax<.00001)
         {
            F=3;/*D=0.0;*/Y++;
            break;
         }

         for(j=1;j<=inv.m;j++)
         {
            test=inv(X,j);
            inv(X,j)=inv(K,j);
            inv(K,j)=test;
            if(F==0)
            {
               test=I(X,j);
               I(X,j)=I(K,j);
               I(K,j)=test;
            }
         }

         L=inv(X,Y);
         for(j=1;j<=inv.n;j++)
         {
            inv(X,j)=inv(X,j)/L;
            if(F==0) I(X,j)=I(X,j)/L;
         }

         for(i=1;i<=inv.m;i++)
         {
            L=inv(i,Y);
            for(j=1;j<=inv.n;j++)
```

```
               if(F==0 && i!=X) I(i,j)=I(i,j)-L*I(X,j);
            }
         }
         ++X;++Y;
      }

      //Copy Inverse matrix from I to inv
      inv=I;
      return F;
   }
   else
   {
      F=4;
      return F;
   }
}

//Calculate the transform of a matrix
//returns 0 if successful and 1 if the matrix sizes are not compatible
int matrix_trans(matrix &mtx, matrix &mtxt)
{
   if(mtx.m==mtxt.n&&mtx.n==mtxt.m)
   {
      for(int i=1;i<=mtx.m;i++)
      {
         for(int j=1;j<=mtx.n;j++)
         {
            mtxt(j,i)=mtx(i,j);
         }
      }
   }
   else
   {
      return 5;
   }
   return 0;
}

//*********************************************************************
//******                 Complex Matrix operations               ******
//*********************************************************************

//Initialize the memory space for the array of the specified size
matrixc::matrixc(int r, int c)
{
   init=0;
   if(r!=0)
   {
      try
      {
         m=r;n=c;
         mtx_data = new complex[m*n];
      }
      catch (xalloc)
      {
         cout << "Could not allocate.";
         exit(-1);
      }

      //Set all cells in the matrix to zero
      for(int i=1; i<=m;i++)
      {
         for(int j=1; j<=n;j++)
         {
            mtx_data[(i-1)*n+j-1]=complex(0.0,0.0);
```

```
        ,
        init=1;
    }
}

//Clean up after the matrix is not needed
matrixc::~matrixc(void)
{
    if(init)
        delete[] mtx_data;
}

//reset the matrix to new dimensions and initialize to zero
void
matrixc::Resize(int r, int c, int useprev)
{
    int mp,np,initp,i;
    complex *mtx_databak;
    initp=init;

    //If useprev==1 save the original info for later use
    if(useprev&&init==1)
    {
        try
        {
            mp=m;np=n;
            mtx_databak = new complex[mp*np];
        }
        catch (xalloc)
        {
            cout << "Could not allocate.";
            exit(-1);
        }

        //copy original data to backup array
        for(i=0;i<m*n;i++)
        {
            mtx_databak[i] = mtx_data[i];
        }
    }

    //delete the previous array if it was there
    if(init)
        delete[] mtx_data;

    init=0;
    if(r!=0)
    {
        try
        {
            m=r;n=c;
            mtx_data = new complex[m*n];
        }
        catch (xalloc)
        {
            cout << "Could not allocate.";
            exit(-1);
        }

        if(useprev&&initp==1)
        {
            //Set all cells in the matrix to prev. value if possible
            for(int i=1; i<=m;i++)
            {
                for(int j=1; j<=n;j++)
                {
                  mtx_data[(i-1)*n+j-1]=mtx_databak[(i-1)*n+j-1];
               else
                  mtx_data[(i-1)*n+j-1]=complex(0.0,0.0);
            }
         }
         delete[] mtx_databak;
      }
      else
      {
         //Set all cells in the matrix to zero
         for(int i=1; i<=m;i++)
         {
            for(int j=1; j<=n;j++)
            {
               mtx_data[(i-1)*n+j-1]=complex(0.0,0.0);
            }
         }
      }
      init=1;
   }
}

//Overload the () operators to initialize the memory space for a matrix or
//to return a reference to the requested cell in a matrix
//If init=0 then the array hasn't been initialized yet otherwise return a value
complex &matrixc::operator()(int r, int c)
{
   if(init==0)
   {
      try
      {
         m=r;n=c;
         mtx_data = new complex[m*n];
      }
      catch (xalloc)
      {
         cout << "Could not allocate.";
         exit(-1);
      }

      //Set all cells in the matrix to zero
      for(int i=1; i<=m;i++)
      {
         for(int j=1; j<=n;j++)
         {
            mtx_data[(i-1)*n+j-1]=complex(0.0,0.0);
         }
      }
      init=1;
      return mtx_data[0];
   }
   else
   {
      return mtx_data[(r-1)*n+c-1];
   }
}

//Overload the () operators to initialize the memory space for a vector or
//to return a reference to the requested cell in a vector
//If init=0 then the array hasn't been initialized yet otherwise return a value
complex &matrixc::operator()(int r)
{
   if(init==0)
   {
      try
      {
            mtx_data = new complex[m];
         }
         catch (xalloc)
         {
            cout << "Could not allocate.";
            exit(-1);
         }

         //Set all cells in the matrix to zero
         for(int i=1; i<=m;i++)
         {
            mtx_data[(i-1)*n]=complex(0.0,0.0);
         }
         init=1;
         return mtx_data[0];
      }
      else
      {
         return mtx_data[r-1];
      }
   }

   //Overload the = operator to copy a double matrix to a complex matrix
   void matrixc::operator=(matrix &mtx)
   {
      if(m==mtx.m&&n==mtx.n)
      {
         for(int i=1;i<=m;i++)
         {
            for(int j=1;j<=n;j++)
            {
               if(fabs(mtx.mtx_data[(i-1)*n+j-1])>1.0e-14)
                  mtx_data[(i-1)*n+j-1]=complex(mtx.mtx_data[(i-1)*n+j-1],0.0);
               else
                  mtx_data[(i-1)*n+j-1]=complex(0.0,0.0);
            }
         }
      }
      else
      {
         exit(-1);
      }
   }

   //Overload the = operator to copy two complex matrices
   void matrixc::operator=(matrixc &mtx)
   {
      if(m==mtx.m&&n==mtx.n)
      {
         for(int i=1;i<=m;i++)
         {
            for(int j=1;j<=n;j++)
            {
               if(abs(mtx.mtx_data[(i-1)*n+j-1])>1.0e-14)
                  mtx_data[(i-1)*n+j-1]=mtx.mtx_data[(i-1)*n+j-1];
               else
                  mtx_data[(i-1)*n+j-1]=complex(0.0,0.0);
            }
         }
      }
      else
      {
         exit(-1);
      }
   }
```

```
{
   for(int i=1;i<=mtx.m;i++)
   {
      for(int j=1;j<=mtx.n;j++)
      {
         mtx(i,j)=complex(0.0,0.0);
      }
   }
}

//Set the matrix equal to an Identity matrix
//returns 0 if successful and 1 if the matrix is not a square matrix
int matrix_I(matrixc &mtx)
{
   if(mtx.m==mtx.n)
   {
      for(int i=1;i<=mtx.m;i++)
      {
         for(int j=1;j<=mtx.n;j++)
         {
            mtx(i,j)=complex(0.0,0.0);
         }
         mtx(i,i)=complex(1.0,0.0);
      }
      return(0);
   }
   else
   {
      return(1);
   }
}

//Add two matrices (a double and a complex) together
//returns 0 if successful and 1 if the matrix sizes are not compatible
int matrix_sum(matrix &mtxl, matrixc &mtxr, matrixc &sum)
{
   if(mtxl.m==mtxr.m&&mtxr.m==sum.m&&mtxl.n==mtxr.n&&mtxr.n==sum.n)
   {
      for(int i=1;i<=mtxl.m;i++)
      {
         for(int j=1;j<=mtxl.n;j++)
         {
            sum(i,j)=complex(mtxl(i,j),0.0)+mtxr(i,j);
         }
      }
      return(0);
   }
   else
   {
      return(1);
   }
}

//Add two matrices (a complex and a double) together
//returns 0 if successful and 1 if the matrix sizes are not compatible
int matrix_sum(matrixc &mtxl, matrix &mtxr, matrixc &sum)
{
   if(mtxl.m==mtxr.m&&mtxr.m==sum.m&&mtxl.n==mtxr.n&&mtxr.n==sum.n)
   {
      for(int i=1;i<=mtxl.m;i++)
      {
         for(int j=1;j<=mtxl.n;j++)
         {
            sum(i,j)=mtxl(i,j)+complex(mtxr(i,j),0.0);
         }
      return(0);
   }
   else
   {
      return(1);
   }
}

//Add two Complex matrices together
//returns 0 if successful and 1 if the matrix sizes are not compatible
int matrix_sum(matrixc &mtxl, matrixc &mtxr, matrixc &sum)
{
   if(mtxl.m==mtxr.m&&mtxr.m==sum.m&&mtxl.n==mtxr.n&&mtxr.n==sum.n)
   {
      for(int i=1;i<=mtxl.m;i++)
      {
         for(int j=1;j<=mtxl.n;j++)
         {
            sum(i,j)=mtxl(i,j)+mtxr(i,j);
         }
      }
      return(0);
   }
   else
   {
      return(1);
   }
}

//Add a double constant to a complex matrix
//returns 0 if successful and 1 if the matrix sizes are not compatible
int matrix_sum(double s, matrixc &mtx, matrixc &sum)
{
   if(mtx.m==sum.m&&mtx.n==sum.n)
   {
      for(int i=1;i<=mtx.m;i++)
      {
         for(int j=1;j<=mtx.n;j++)
         {
            sum(i,j)=complex(s,0.0)+mtx(i,j);
         }
      }
      return(0);
   }
   else
   {
      return(1);
   }
}

//Add a complex constant to a complex matrix
//returns 0 if successful and 1 if the matrix sizes are not compatible
int matrix_sum(complex s, matrixc &mtx, matrixc &sum)
{
   if(mtx.m==sum.m&&mtx.n==sum.n)
   {
      for(int i=1;i<=mtx.m;i++)
      {
         for(int j=1;j<=mtx.n;j++)
         {
            sum(i,j)=s+mtx(i,j);
         }
      }
      return(0);
   }
   else
```

```
      return(1);
   }
}

//Multiply two matrices together (one double and one complex)
//returns 0 if successful and 2 if the matrix sizes are not compatible
int matrix_mult(matrix &mtxl, matrixc &mtxr, matrixc &prod)
{
   if(mtxl.n==mtxr.m&&mtxl.m==prod.m&&mtxr.n==prod.n)
   {
      // Zero Product matrix before calculating
      matrix_zero(prod);

      for(int i=1;i<=mtxl.m;i++)
      {
         for(int j=1;j<=mtxr.n;j++)
         {
            for(int k=1;k<=mtxl.n;k++)
            {
               prod(i,j)+=complex(mtxl(i,k),0.0)*mtxr(k,j);
            }
         }
      }
      return(0);
   }
   else
   {
      return(2);
   }
}

//Multiply two matrices together (one complex and one double)
//returns 0 if successful and 2 if the matrix sizes are not compatible
int matrix_mult(matrixc &mtxl, matrix &mtxr, matrixc &prod)
{
   if(mtxl.n==mtxr.m&&mtxl.m==prod.m&&mtxr.n==prod.n)
   {
      // Zero Product matrix before calculating
      matrix_zero(prod);

      for(int i=1;i<=mtxl.m;i++)
      {
         for(int j=1;j<=mtxr.n;j++)
         {
            for(int k=1;k<=mtxl.n;k++)
            {
               prod(i,j)+=mtxl(i,k)*complex(mtxr(k,j),0.0);
            }
         }
      }
      return(0);
   }
   else
   {
      return(2);
   }
}

//Multiply two matrices together (both complex)
//returns 0 if successful and 2 if the matrix sizes are not compatible
int matrix_mult(matrixc &mtxl, matrixc &mtxr, matrixc &prod)
{
   if(mtxl.n==mtxr.m&&mtxl.m==prod.m&&mtxr.n==prod.n)
   {
      // Zero Product matrix before calculating
      matrix_zero(prod);
      for(int i=1;i<=mtxl.m;i++)
      {
         for(int j=1;j<=mtxr.n;j++)
         {
            for(int k=1;k<=mtxl.n;k++)
            {
               prod(i,j)+=mtxl(i,k)*mtxr(k,j);
            }
         }
      }
      return(0);
   }
   else
   {
      return(2);
   }
}

//Multiply a complex matrix by a double constant
//returns 0 if successful and 2 if the matrix sizes are not compatible
int matrix_mult(double m, matrixc &mtx, matrixc &prod)
{
   if(mtx.m==prod.m&&mtx.n==prod.n)
   {
      for(int i=1;i<=mtx.m;i++)
      {
         for(int j=1;j<=mtx.n;j++)
         {
            prod(i,j)=complex(m,0.0)*mtx(i,j);
         }
      }
      return(0);
   }
   else
   {
      return(2);
   }
}

//Multiply a double matrix by a complex constant
//returns 0 if successful and 2 if the matrix sizes are not compatible
int matrix_mult(complex m, matrix &mtx, matrixc &prod)
{
   if(mtx.m==prod.m&&mtx.n==prod.n)
   {
      for(int i=1;i<=mtx.m;i++)
      {
         for(int j=1;j<=mtx.n;j++)
         {
            prod(i,j)=m*complex(mtx(i,j),0.0);
         }
      }
      return(0);
   }
   else
   {
      return(2);
   }
}

//Multiply a complex matrix by a complex constant
//returns 0 if successful and 2 if the matrix sizes are not compatible
int matrix_mult(complex m, matrixc &mtx, matrixc &prod)
{
   if(mtx.m==prod.m&&mtx.n==prod.n)
   {
         for(int j=1;j<=mtx.n;j++)
         {
            prod(i,j)=m*mtx(i,j);
         }
      }
      return(0);
   }
   else
   {
      return(2);
   }
}

//Calculate the inverse of a given matrixc
//return F=0:     Inverse was successful
//return F=3:     Matrix is non-invertable
//return F=4:     The matrix sizes are incompatable
int matrix_inv(matrixc &mtx, matrixc &inv)
{
   int i,j,F,X,Y,K;
   complex Cmax,test,L;
   F=0;        //Flag to monitor the matrix singularity
   X=Y=1;      //Start at top left of matrix
   if(mtx.m==mtx.n&&mtx.m==inv.m&&inv.m==inv.n)
   {
      //Create the Identity matrix
      matrixc I(mtx.m,mtx.n);
      for(i=1;i<=I.m;i++)
      {
         I(i,i)=complex(1.0,0.0);
      }

      //Copy elements from mtx to inv
      inv=mtx;

      while(X<=inv.m&&Y<=inv.n)
      {
         Cmax=complex(0.0,0.0);K=X;
         for(i=X;i<=inv.m;i++)
         {
            test=complex(abs(inv(i,Y)),0.0);
            if(abs(test)>abs(Cmax))
            {
               Cmax=test;
               K=i;
            }
         }

         if(abs(Cmax)<0.00001)
         {
            F=3;/*D=0.0;*/Y++;
            break;
         }

         for(j=1;j<=inv.m;j++)
         {
            test=inv(X,j);
            inv(X,j)=inv(K,j);
            inv(K,j)=test;
            if(F==0)
            {
               test=I(X,j);
               I(X,j)=I(K,j);
               I(K,j)=test;
            }
            L=inv(X,Y);
            for(j=1;j<=inv.n;j++)
            {
               inv(X,j)=inv(X,j)/L;
               if(F==0) I(X,j)=I(X,j)/L;
            }

            for(i=1;i<=inv.m;i++)
            {
               L=inv(i,Y);
               for(j=1;j<=inv.n;j++)
               {
                  if(i!=X) inv(i,j)=inv(i,j)-L*inv(X,j);
                  if(F==0 && i!=X) I(i,j)=I(i,j)-L*I(X,j);
               }
            }
            ++X;++Y;
         }

         //Copy Inverse matrix from I to inv
         inv=I;
         return F;
      }
      else
      {
         F=4;
         return F;
      }
   }

   //Calculate the transform of a complex matrix
   //returns 0 if successful and 5 if the matrix sizes are not compatible
   int matrix_trans(matrixc &mtx, matrixc &mtxt)
   {
      if(mtx.m==mtxt.n&&mtx.n==mtxt.m)
      {
         for(int i=1;i<=mtx.m;i++)
         {
            for(int j=1;j<=mtx.n;j++)
            {
               mtxt(j,i)=mtx(i,j);
            }
         }
         return (0);
      }
      else
      {
         return (5);
      }
   }
```

```
#define SWAP(a,b) temp=(a);(a)=(b);(b)=temp;
#define M 7
#define NSTACK 100
#include "matrix.h"

void sort3(matrix& arr, matrix& brr, matrix& crr)
{
   int i,ir=arr.m,j,k,l=1;
   int jstack=0;
   matrixi istack;
   double a,b,c,temp;

   istack(NSTACK);
   for (;;) {
      if (ir-l < M) {
         for (j=l+1;j<=ir;j++) {
            a=arr(j);
            b=brr(j);
            c=crr(j);
            for (i=j-1;i>=l;i--) {
               if (arr(i) <= a) break;
               arr(i+1)=arr(i);
               brr(i+1)=brr(i);
               crr(i+1)=crr(i);
            }
            arr(i+1)=a;
            brr(i+1)=b;
            crr(i+1)=c;
         }
         if (!jstack) {
            return;
         }
         ir=istack(jstack);
         l=istack(jstack-1);
         jstack -= 2;
      } else {
         k=(l+ir) >> 1;
         SWAP(arr(k),arr(l+1))
         SWAP(brr(k),brr(l+1))
         SWAP(crr(k),crr(l+1))
         if (arr(l) > arr(ir)) {
            SWAP(arr(l),arr(ir))
            SWAP(brr(l),brr(ir))
            SWAP(crr(l),crr(ir))
         }
         if (arr(l+1) > arr(ir)) {
            SWAP(arr(l+1),arr(ir))
            SWAP(brr(l+1),brr(ir))
            SWAP(crr(l+1),crr(ir))
         }
         if (arr(l) > arr(l+1)) {
            SWAP(arr(l),arr(l+1))
            SWAP(brr(l),brr(l+1))
            SWAP(crr(l),crr(l+1))
         }
         i=l+1;
         j=ir;
         a=arr(l+1);
         b=brr(l+1);
         c=crr(l+1);
         for (;;) {
            do i++; while (arr(i) < a);
            do j--; while (arr(j) > a);
            if (j < i) break;
            SWAP(arr(i),arr(j))
         }
         arr(l+1)=arr(j);
         arr(j)=a;
         brr(l+1)=brr(j);
         brr(j)=b;
         crr(l+1)=crr(j);
         crr(j)=c;
         jstack += 2;
   //    if (jstack > NSTACK) nrerror("NSTACK too small in sort2.");
         if (ir-i+1 >= j-l) {
            istack(jstack)=ir;
            istack(jstack-1)=i;
            ir=j-1;
         } else {
            istack(jstack)=j-1;
            istack(jstack-1)=l;
            l=i;
         }
      }
   }
}
#undef M
#undef NSTACK
#undef SWAP
```

```
#ifndef _RGG_H_
#define _RGG_H_

int rgg(matrix& a,matrix& b,matrix& alfr,matrix& alfi,matrix& beta,
      const int matz,matrix& z);

#endif
```

RGGPOOL.CPP

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "matrix.h"

double sign(double x, double y);

double epslon (double x);

void qzhes(matrix& a,matrix& b,const int matz,matrix& z);

int qzit(matrix& a,matrix& b,double eps1,int matz,matrix& z);

void qzval(matrix& a,matrix& b,matrix& alfr,matrix& alfi,
            matrix& beta,int matz,matrix& z);

void qzvec(matrix& a,matrix& b,matrix& alfr,matrix& alfi,
            matrix& beta,matrix& z);

double
sign(double x, double y)
{
   double sn;

   if(y>=0.0)
      sn=fabs(x);
   else
      sn=-fabs(x);

   return sn;
}

/*
c     estimate unit roundoff in quantities of size x.
c
      double precision a,b,c,eps
c
c     this program should function properly on all systems
c     satisfying the following two assumptions,
c        1.  the base used in representing floating point
c            numbers is not a power of three.
c        2.  the quantity  a  in statement 10 is represented to
c            the accuracy used in floating point variables
c            that are stored in memory.
c     the statement number 10 and the go to 10 are intended to
c     force optimizing compilers to generate code satisfying
c     assumption 2.
c     under these assumptions, it should be true that,
c            a  is not exactly equal to four-thirds,
c            b  has a zero for its last bit or digit,
c            c  is not exactly equal to one,
c            eps  measures the separation of 1.0 from
```

```
c      the developers of eispack would appreciate being informed
c      about any systems where these assumptions do not hold.
c
c      this version dated 12/7/97
c      -> converted to Borland C++ by Joeleff Fitzsimmons, UTIAS
*/

double
epslon (double x)
{
   double a,b,c,eps;
   eps=0.0;
   a = 4.0/3.0;
   while(eps==0.0)
   {
      b = a - 1.0;
      c = b + b + b;
      eps = fabs(c-1.0);
   }
   return eps*fabs(x);
}

/*
     this subroutine is the first step of the qz algorithm
     for solving generalized matrix eigenvalue problems,
     siam j. numer. anal. 10, 241-256(1973) by moler and stewart.

     this subroutine accepts a pair of real general matrices and
     reduces one of them to upper hessenberg form and the other
     to upper triangular form using orthogonal transformations.
     it is usually followed by  qzit,  qzval  and, possibly,  qzvec.

     on input

        nm must be set to the row dimension of two-dimensional
          array parameters as declared in the calling program
          dimension statement.

        n is the order of the matrices.

        a contains a real general matrix.

        b contains a real general matrix.

        matz should be set to true if the right hand transformations
          are to be accumulated for later use in computing
          eigenvectors, and to false otherwise.

     on output

        a has been reduced to upper hessenberg form.  the elements
          below the first subdiagonal have been set to zero.

        b has been reduced to upper triangular form.  the elements
          below the main diagonal have been set to zero.

        z contains the product of the right hand transformations if
          matz has been set to true  otherwise, z is not referenced.

     questions and comments should be directed to burton s. garbow,
     mathematics and computer science div, argonne national laboratory

     this version dated 12/7/97
      -> converted to Borland C++ by Joeleff Fitzsimmons, UTIAS
*/
```

```
qzhes(matrix& a,matrix& b,const int matz,matrix& z)
{
   int i,j,k,l,lb,l1,nk1,nm1,nm2,n;
   double  r,s,t,u1,u2,v1,v2,rho;

   n=a.n;
   //.......... initialize z ..........
   if (matz)
   {
      for(j=1;j<=n;j++)
      {
         for(i=1;i<=n;i++)
         {
            z(i,j) = 0.0;
         }
         z(j,j) = 1.0;
      }
   }

   //.......... reduce b to upper triangular form ..........
   if (n > 1)
   {
      nm1 = n - 1;

      for(l=1;l<=nm1;l++)
      {
         l1 = l + 1;
         s = 0.0;

         for(i=l1;i<=n;i++)
            s = s + fabs(b(i,l));

         if (s == 0.0) continue;

         s = s + fabs(b(l,l));
         r = 0.0;

         for(i=l;i<=n;i++)
         {
            b(i,l) = b(i,l) / s;
            r = r + pow(b(i,l),2.0);
         }

         r = sign(sqrt(r),b(l,l));
         b(l,l) = b(l,l) + r;
         rho = r * b(l,l);

         for(j=l1;j<=n;j++)
         {
            t = 0.0;

            for(i=l;i<=n;i++)
               t = t + b(i,l) * b(i,j);

            t = -t / rho;

            for(i=l;i<=n;i++)
               b(i,j) = b(i,j) + t * b(i,l);
         }

         for(j=l;j<=n;j++)
         {
            t = 0.0;

            for(i=l;i<=n;i++)
               t = t + b(i,l) * a(i,j);
         t = -t / rho;

         for(i=1;i<=n;i++)
            a(i,j) = a(i,j) + t * b(i,l);
      }

      b(l,l) = -s * r;

      for(i=l1;i<=n;i++)
         b(i,l) = 0.0;
   }

   //.......... reduce a to upper hessenberg form, while
   //           keeping b triangular ..........
   if (n!=2)
   {
      nm2 = n - 2;

      for(k=1;k<=nm2;k++)
      {
         nk1 = nm1 - k;
         //.......... for l=n-1 step -1 until k+1 do -- ..........
         for(lb=1;lb<=nk1;lb++)
         {
            l = n - lb;
            l1 = l + 1;
            //.......... zero a(l+1,k) ..........
            s = fabs(a(l,k)) + fabs(a(l1,k));
            if (s==0.0) continue;
            u1 = a(l,k) / s;
            u2 = a(l1,k) / s;
            r = sign(sqrt(u1*u1+u2*u2),u1);
            v1 =  -(u1 + r) / r;
            v2 = -u2 / r;
            u2 = v2 / v1;

            for(j=k;j<=n;j++)
            {
               t = a(l,j) + u2 * a(l1,j);
               a(l,j) = a(l,j) + t * v1;
               a(l1,j) = a(l1,j) + t * v2;
            }

            a(l1,k) = 0.0;

            for(j=l;j<=n;j++)
            {
               t = b(l,j) + u2 * b(l1,j);
               b(l,j) = b(l,j) + t * v1;
               b(l1,j) = b(l1,j) + t * v2;
            }

            //.......... zero b(l+1,l) ..........
            s = fabs(b(l1,l1)) + fabs(b(l1,l));
            if (s==0.0) continue;
            u1 = b(l1,l1) / s;
            u2 = b(l1,l) / s;
            r = sign(sqrt(u1*u1+u2*u2),u1);
            v1 =  -(u1 + r) / r;
            v2 = -u2 / r;
            u2 = v2 / v1;

            for(i=1;i<=l1;i++)
            {
               t = b(i,l1) + u2 * b(i,l);
               b(i,l1) = b(i,l1) + t * v1;
               }

               b(l1,l) = 0.0;

               for(i=1;i<=n;i++)
               {
                  t = a(i,l1) + u2 • a(i,l);
                  a(i,l1) = a(i,l1) + t * v1;
                  a(i,l) = a(i,l) + t • v2;
               }

               if (!matz) continue;

               for(i=1;i<=n;i++)
               {
                  t = z(i,l1) + u2 • z(i,l);
                  z(i,l1) = z(i,l1) + t * v1;
                  z(i,l) = z(i,l) + t * v2;
               }

            }
         }
      }
   }
}

/*   this subroutine is the second step of the qz algorithm
     for solving generalized matrix eigenvalue problems,
     siam j. numer. anal. 10, 241-256(1973) by moler and stewart,
     as modified in technical note nasa tn d-7305(1973) by ward.

     this subroutine accepts a pair of real matrices, one of them
     in upper hessenberg form and the other in upper triangular form.
     it reduces the hessenberg matrix to quasi-triangular form using
     orthogonal transformations while maintaining the triangular form
     of the other matrix.  it is usually preceded by  qzhes  and
     followed by  qzval  and, possibly,  qzvec.

     on input

        nm must be set to the row dimension of two-dimensional
          array parameters as declared in the calling program
          dimension statement.

        n is the order of the matrices.

        a contains a real upper hessenberg matrix.

        b contains a real upper triangular matrix.

        eps1 is a tolerance used to determine negligible elements.
          eps1 = 0.0 (or negative) may be input, in which case an
          element will be neglected only if it is less than roundoff
          error times the norm of its matrix.  if the input eps1 is
          positive, then an element will be considered negligible
          if it is less than eps1 times the norm of its matrix.  a
          positive value of eps1 may result in faster execution,
          but less accurate results.

        matz should be set to .true. if the right hand transformations
          are to be accumulated for later use in computing
          eigenvectors, and to .false. otherwise.

        z contains, if matz has been set to .true., the
          transformation matrix produced in the reduction
          by  qzhes, if performed, or else the identity matrix.
      on output

         a has been reduced to quasi-triangular form.  the elements
           below the first subdiagonal are still zero and no two
           consecutive subdiagonal elements are nonzero.

         b is still in upper triangular form, although its elements
           have been altered.  the location b(n,1) is used to store
           eps1 times the norm of b for later use by  qzval  and  qzvec.

         z contains the product of the right hand transformations
           (for both steps) if matz has been set to .true..

         ierr is set to
           zero       for normal return,
           j          if the limit of 30*n iterations is exhausted
                      while the j-th eigenvalue is being sought.

      questions and comments should be directed to burton s. garbow,
      mathematics and computer science div, argonne national laboratory

      this version dated 12/7/97
       -> converted to Borland C++ by Joeleff Fitzsimmons, UTIAS
*/

int
qzit(matrix& a,matrix& b,double eps1,int matz,matrix& z)
{
   int i,j,k,l,n,en,k1,k2,ld,l1,l1,na,ish,itn,its,km1,
         lm1,enm2,ierr,lor1,enorn,notlas;
   double r,s,t,a1,a2,a3,ep,sh,u1,u2,u3,v1,v2,v3,ani,a11,
         a12,a21,a22,a33,a34,a43,a44,bni,b11,b12,b22,b33,b34,
         b44,epsa,epsb,anorm,bnorm;

   n=a.n;
   ierr = 0;
   //.......... compute epsa,epsb ..........
   anorm = 0.0;
   bnorm = 0.0;

   for(i=1;i<=n;i++)
   {
      ani = 0.0;
      if (i!=1) ani = fabs(a(i,i-1));
      bni = 0.0;

      for(j=i;j<=n;j++)
      {
         ani = ani + fabs(a(i,j));
         bni = bni + fabs(b(i,j));
      }

      if (ani>anorm) anorm = ani;
      if (bni>bnorm) bnorm = bni;
   }

   if (anorm==0.0) anorm = 1.0;
   if (bnorm==0.0) bnorm = 1.0;
   ep = eps1;
   if (ep<=0.0)
   {
      //.......... use roundoff level if eps1 is zero ..........
      ep = epslon(1.0);
   }
   epsa = ep * anorm;
```

```
//.......... reduce a to quasi-triangular form, while
//            keeping b triangular ..........
lor1 = 1;
enorn = n;
en = n;
itn = 30*n;
//.......... begin qz step ..........
I60:
if (en<=2) goto I1001;
if (!matz) enorn = en;
its = 0;
na = en - 1;
enm2 = na - 1;

I70:
ish = 2;
//.......... check for convergence or reducibility.
//            for l=en step -1 until 1 do -- ..........
for(ll=1;ll<=en;ll++)
{
   lm1 = en - ll;
   l = lm1 + 1;
   if (l==1) goto I95;
   if (fabs(a(l,lm1))<=epsa) goto I90;
}

I90:
a(l,lm1) = 0.0;
if (l<na) goto I95;
//.......... 1-by-1 or 2-by-2 block isolated ..........
en = lm1;
goto I60;

//.......... check for small top of b ..........
I95:
ld = l;

I100:
l1 = l + 1;
b11 = b(l,l);
if (fabs(b11)>epsb) goto I120;
b(l,l) = 0.0;
s = fabs(a(l,l)) + fabs(a(l1,l));
u1 = a(l,l) / s;
u2 = a(l1,l) / s;
r = sign(sqrt(u1*u1+u2*u2),u1);
v1 = -(u1 + r) / r;
v2 = -u2 / r;
u2 = v2 / v1;

for(j=l;j<=enorn;j++)
{
   t = a(l,j) + u2 * a(l1,j);
   a(l,j) = a(l,j) + t * v1;
   a(l1,j) = a(l1,j) + t * v2;
   t = b(l,j) + u2 * b(l1,j);
   b(l,j) = b(l,j) + t * v1;
   b(l1,j) = b(l1,j) + t * v2;
}

if (l!=1) a(l,lm1) = -a(l,lm1);
lm1 = l;
l = l1;
goto I90;
```

```
a11 = a(l,l) / b11;
a21 = a(l1,l) / b11;
if (ish==1) goto I140;
//.......... iteration strategy ..........
if (itn==0) goto I1000;
if (its==10) goto I155;
//.......... determine type of shift ..........
b22 = b(l1,l1);
if (fabs(b22)<epsb) b22 = epsb;
b33 = b(na,na);
if (fabs(b33)<epsb) b33 = epsb;
b44 = b(en,en);
if (fabs(b44)<epsb) b44 = epsb;
a33 = a(na,na) / b33;
a34 = a(na,en) / b44;
a43 = a(en,na) / b33;
a44 = a(en,en) / b44;
b34 = b(na,en) / b44;
t = 0.5 * (a43 * b34 - a33 - a44);
r = t * t + a34 * a43 - a33 * a44;
if (r<0.0) goto I150;
//.......... determine single shift zeroth column of a ..........
ish = 1;
r = sqrt(r);
sh = -t + r;
s = -t - r;
if (fabs(s-a44)<fabs(sh-a44)) sh = s;
//.......... look for two consecutive small
//           sub-diagonal elements of a.
//           for l=en-2 step -1 until ld do -- ..........
for(ll=ld;ll<=enm2;ll++)
{
   l = enm2 + ld - ll;
   if (l==ld) goto I140;
   lm1 = l - 1;
   l1 = l + 1;
   t = a(l,l);
   if (fabs(b(l,l))>epsb) t = t - sh * b(l,l);
   if (fabs(a(l,lm1))<=fabs(t/a(l1,l)) * epsa) goto I100;
}

I140:
a1 = a11 - sh;
a2 = a21;
if (l!=ld) a(l,lm1) = -a(l,lm1);
goto I160;

//.......... determine double shift zeroth column of a ..........
I150:
a12 = a(l,l1) / b22;
a22 = a(l1,l1) / b22;
b12 = b(l,l1) / b22;
a1 = ((a33 - a11) * (a44 - a11) - a34 * a43 + a43 * b34 * a11)
      / a21 + a12 - a11 * b12;
a2 = (a22 - a11) - a21 * b12 - (a33 - a11) - (a44 - a11)
      + a43 * b34;
a3 = a(l1+1,l1) / b22;
goto I160;

//.......... ad hoc shift ..........
I155:
a1 = 0.0;
a2 = 1.0;
a3 = 1.1605;
I160:
its = its + 1;
```

```
[illegible]
//.......... main loop ..........
for(k=1;k<=na;k++)
{
   notlas = k!=na&&ish==2;
   k1 = k + 1;
   k2 = k + 2;
   km1 = max(k-1,1);
   l1 = min(en,k1+ish);
   if (notlas) goto I190;
   //.......... zero a(k+1,k-1) ..........
   if (k!=l)
   {
      a1 = a(k,km1);
      a2 = a(k1,km1);
   }
   s = fabs(a1) + fabs(a2);
   if (s==0.0) goto I70;
   u1 = a1 / s;
   u2 = a2 / s;
   r = sign(sqrt(u1*u1+u2*u2),u1);
   v1 = -(u1 + r) / r;
   v2 = -u2 / r;
   u2 = v2 / v1;

   for(j=km1;j<=enorn;j++)
   {
      t = a(k,j) + u2 * a(k1,j);
      a(k,j) = a(k,j) + t * v1;
      a(k1,j) = a(k1,j) + t * v2;
      t = b(k,j) + u2 * b(k1,j);
      b(k,j) = b(k,j) + t * v1;
      b(k1,j) = b(k1,j) + t * v2;
   }

   if (k!=l) a(k1,km1) = 0.0;
   goto I240;

   //.......... zero a(k+1,k-1) and a(k+2,k-1) ..........
   I190:
   if (k!=l)
   {
      a1 = a(k,km1);
      a2 = a(k1,km1);
      a3 = a(k2,km1);
   }
   s = fabs(a1) + fabs(a2) + fabs(a3);
   if (s==0.0) continue;
   u1 = a1 / s;
   u2 = a2 / s;
   u3 = a3 / s;
   r = sign(sqrt(u1*u1+u2*u2+u3*u3),u1);
   v1 = -(u1 + r) / r;
   v2 = -u2 / r;
   v3 = -u3 / r;
   u2 = v2 / v1;
   u3 = v3 / v1;

   for(j=km1;j<=enorn;j++)
   {
      t = a(k,j) + u2 * a(k1,j) + u3 * a(k2,j);
      a(k,j) = a(k,j) + t * v1;
      a(k1,j) = a(k1,j) + t * v2;
      a(k2,j) = a(k2,j) + t * v3;
      t = b(k,j) + u2 * b(k1,j) + u3 * b(k2,j);
      b(k,j) = b(k,j) + t * v1;
}

if (k!=1)
{
   a(k1,km1) = 0.0;
   a(k2,km1) = 0.0;
}
//.......... zero b(k+2,k+1) and b(k+2,k) ..........
s = fabs(b(k2,k2)) + fabs(b(k2,k1)) + fabs(b(k2,k));
if (s==0.0) goto I240;
u1 = b(k2,k2) / s;
u2 = b(k2,k1) / s;
u3 = b(k2,k) / s;
r = sign(sqrt(u1*u1+u2*u2+u3*u3),u1);
v1 = -(u1 + r) / r;
v2 = -u2 / r;
v3 = -u3 / r;
u2 = v2 / v1;
u3 = v3 / v1;

for(i=lor1;i<=l1;i++)
{
   t = a(i,k2) + u2 * a(i,k1) + u3 * a(i,k);
   a(i,k2) = a(i,k2) + t * v1;
   a(i,k1) = a(i,k1) + t * v2;
   a(i,k) = a(i,k) + t * v3;
   t = b(i,k2) + u2 * b(i,k1) + u3 * b(i,k);
   b(i,k2) = b(i,k2) + t * v1;
   b(i,k1) = b(i,k1) + t * v2;
   b(i,k) = b(i,k) + t * v3;
}

b(k2,k) = 0.0;
b(k2,k1) = 0.0;
if (!matz) goto I240;

for(i=1;i<=n;i++)
{
   t = z(i,k2) + u2 * z(i,k1) + u3 * z(i,k);
   z(i,k2) = z(i,k2) + t * v1;
   z(i,k1) = z(i,k1) + t * v2;
   z(i,k) = z(i,k) + t * v3;
}

//.......... zero b(k+1,k) ..........
I240:
s = fabs(b(k1,k1)) + fabs(b(k1,k));
if (s==0.0) continue;
u1 = b(k1,k1) / s;
u2 = b(k1,k) / s;
r = sign(sqrt(u1*u1+u2*u2),u1);
v1 = -(u1 + r) / r;
v2 = -u2 / r;
u2 = v2 / v1;

for(i=lor1;i<=l1;i++)
{
   t = a(i,k1) + u2 * a(i,k);
   a(i,k1) = a(i,k1) + t * v1;
   a(i,k) = a(i,k) + t * v2;
   t = b(i,k1) + u2 * b(i,k);
   b(i,k1) = b(i,k1) + t * v1;
   b(i,k) = b(i,k) + t * v2;
}
```

```
         if (!matz) continue;

         for(i=1;i<=n;i++)
         {
            t = z(i,k1) + u2 * z(i,k);
            z(i,k1) = z(i,k1) + t * v1;
            z(i,k) = z(i,k) + t * v2;
         }

      }

      //.......... end qz step ..........
      goto I70;
      //.......... set error -- all eigenvalues have not
      //           converged after 30*n iterations ..........
      I1000:
      ierr = en;
      //.......... save epsb for use by qzval and qzvec ..........
      I1001:
      if (n>1) b(n,1) = epsb;
      return ierr;
   }

   /*   this subroutine is the third step of the qz algorithm
        for solving generalized matrix eigenvalue problems,
        siam j. numer. anal. 10, 241-256(1973) by moler and stewart.

        this subroutine accepts a pair of real matrices, one of them
        in quasi-triangular form and the other in upper triangular form.
        it reduces the quasi-triangular matrix further, so that any
        remaining 2-by-2 blocks correspond to pairs of complex
        eigenvalues, and returns quantities whose ratios give the
        generalized eigenvalues.  it is usually preceded by  qzhes
        and  qzit  and may be followed by  qzvec.

        on input

           nm must be set to the row dimension of two-dimensional
             array parameters as declared in the calling program
             dimension statement.

           n is the order of the matrices.

           a contains a real upper quasi-triangular matrix.

           b contains a real upper triangular matrix.  in addition,
             location b(n,1) contains the tolerance quantity (epsb)
             computed and saved in  qzit.

           matz should be set to .true. if the right hand transformations
             are to be accumulated for later use in computing
             eigenvectors, and to .false. otherwise.

           z contains, if matz has been set to .true., the
             transformation matrix produced in the reductions by qzhes
             and qzit, if performed, or else the identity matrix.
             if matz has been set to .false., z is not referenced.

        on output

           a has been reduced further to a quasi-triangular matrix
             in which all nonzero subdiagonal elements correspond to
             pairs of complex eigenvalues.

           b is still in upper triangular form, although its elements
             have been altered.  b(n,1) is unaltered.
        alfr and alfi contain the real and imaginary parts of the
          diagonal elements of the triangular matrix that would be
          obtained if a were reduced completely to triangular form
          by unitary transformations.  non-zero values of alfi occur
          in pairs, the first member positive and the second negative.

        beta contains the diagonal elements of the corresponding b,
          normalized to be real and non-negative.  the generalized
          eigenvalues are then the ratios ((alfr+i*alfi)/beta).

        z contains the product of the right hand transformations
          (for all three steps) if matz has been set to .true.

     questions and comments should be directed to burton s. garbow,
     mathematics and computer science div, argonne national laboratory

     this version dated 12/7/97
      -> converted to Borland C++ by Joeleff Fitzsimmons, UTIAS
*/

void
qzval(matrix& a,matrix& b,matrix& alfr,matrix& alfi,matrix& beta,
      int matz,matrix& z)
{
   int i,j,en,na,nn,isw;
   double n,c,d,e,r,s,t,an,a1,a2,bn,cq,cz,di,dr,ei,ti,tr,u1,
         u2,v1,v2,a1i,a11,a12,a2i,a21,a22,b11,b12,b22,sqi,sqr,
         ssi,ssr,szi,szr,a11i,a11r,a12i,a12r,a22i,a22r,epsb;
   n=a.n;
   epsb = b(n,1);
   isw = 1;
   //......... find eigenvalues of quasi-triangular matrices.
   //          for en=n step -1 until 1 do -- .........
   for(nn=1;nn<=n;nn++)
   {
      en = n + 1 - nn;
      na = en - 1;
      if (isw==2) goto I505;
      if (en==1) goto I410;
      if (a(en,na)!=0.0) goto I420;

      //......... 1-by-1 block, one real root .........
      I410:
      alfr(en) = a(en,en);
      if (b(en,en)<0.0) alfr(en) = -alfr(en);
      beta(en) = fabs(b(en,en));
      alfi(en) = 0.0;
      continue;

      //......... 2-by-2 block .........
      I420:
      if (fabs(b(na,na))<=epsb) goto I455;
      if (fabs(b(en,en))>epsb) goto I430;
      a1 = a(en,en);
      a2 = a(en,na);
      bn = 0.0;
      goto I435;
      I430:
      an = fabs(a(na,na)) + fabs(a(na,en)) + fabs(a(en,na))+fabs(a(en,en));
      bn = fabs(b(na,na)) + fabs(b(na,en)) + fabs(b(en,en));
      a11 = a(na,na) / an;
      a12 = a(na,en) / an;
      a21 = a(en,na) / an;
      a22 = a(en,en) / an;
      b11 = b(na,na) / bn;
      b12 = b(na,en) / bn;
```

```
e = a11 / b11;
ei = a22 / b22;
s = a21 / (b11 * b22);
t = (a22 - e * b22) / b22;
if (fabs(e)<=fabs(ei)) goto I431;
e = ei;
t = (a11 - e * b11) / b11;

I431:
c = 0.5 * (t - s * b12);
d = c * c + s * (a12 - e * b12);
if (d<0.0) goto I480;

//.......... two real roots.
//           zero both a(en,na) and b(en,na) ..........
e = e + (c + sign(sqrt(d),c));
a11 = a11 - e * b11;
a12 = a12 - e * b12;
a22 = a22 - e * b22;
if (fabs(a11)+fabs(a12)<fabs(a21)+fabs(a22)) goto I432;
a1 = a12;
a2 = a11;
goto I435;

I432:
a1 = a22;
a2 = a21;

//.......... choose and apply real z ..........
I435:
s = fabs(a1) + fabs(a2);
u1 = a1 / s;
u2 = a2 / s;
r = sign(sqrt(u1*u1+u2*u2),u1);
v1 = -(u1 + r) / r;
v2 = -u2 / r;
u2 = v2 / v1;

for(i=1;i<=en;i++)
{
   t = a(i,en) + u2 * a(i,na);
   a(i,en) = a(i,en) + t * v1;
   a(i,na) = a(i,na) + t * v2;
   t = b(i,en) + u2 * b(i,na);
   b(i,en) = b(i,en) + t * v1;
   b(i,na) = b(i,na) + t * v2;
}

if (!matz) goto I450;

for(i=1;i<=n;i++)
{
   t = z(i,en) + u2 * z(i,na);
   z(i,en) = z(i,en) + t * v1;
   z(i,na) = z(i,na) + t * v2;
}

I450:
if (bn==0.0) goto I475;
if (an<fabs(e) * bn) goto I455;
a1 = b(na,na);
a2 = b(en,na);
goto I460;

I455:
a1 = a(na,na);
```

```
//......... choose and apply real q .........
I460:
s = fabs(a1) + fabs(a2);
if (s==0.0) goto I475;
u1 = a1 / s;
u2 = a2 / s;
r = sign(sqrt(u1*u1+u2*u2),u1);
v1 = -(u1 + r) / r;
v2 = -u2 / r;
u2 = v2 / v1;

for(j=na;j<=n;j++)
{
   t = a(na,j) + u2 * a(en,j);
   a(na,j) = a(na,j) + t * v1;
   a(en,j) = a(en,j) + t * v2;
   t = b(na,j) + u2 * b(en,j);
   b(na,j) = b(na,j) + t * v1;
   b(en,j) = b(en,j) + t * v2;
}

I475:
a(en,na) = 0.0;
b(en,na) = 0.0;
alfr(na) = a(na,na);
alfr(en) = a(en,en);
if (b(na,na)<0.0) alfr(na) = -alfr(na);
if (b(en,en)<0.0) alfr(en) = -alfr(en);
beta(na) = fabs(b(na,na));
beta(en) = fabs(b(en,en));
alfi(en) = 0.0;
alfi(na) = 0.0;
goto I505;

//......... two complex roots .........
I480:
e = e + c;
ei = sqrt(-d);
a11r = a11 - e * b11;
a11i = ei * b11;
a12r = a12 - e * b12;
a12i = ei * b12;
a22r = a22 - e * b22;
a22i = ei * b22;
if (fabs(a11r) + fabs(a11i) + fabs(a12r) + fabs(a12i) <
      fabs(a21) + fabs(a22r) + fabs(a22i)) goto I482;
a1 = a12r;
a1i = a12i;
a2 = -a11r;
a2i = -a11i;
goto I485;

I482:
a1 = a22r;
a1i = a22i;
a2 = -a21;
a2i = 0.0;

//......... choose complex z .........
I485:
cz = sqrt(a1*a1+a1i*a1i);
if (cz==0.0) goto I487;
szr = (a1 * a2 + a1i * a2i) / cz;
szi = (a1 * a2i - a1i * a2) / cz;
r = sqrt(cz*cz+szr*szr+szi*szi);
```

```
szr = szr / r;
szi = szi / r;
goto I490;

I487:
szr = 1.0;
szi = 0.0;

I490:
if (an<(fabs(e) + ei) * bn) goto I492;
a1 = cz * b11 + szr * b12;
a1i = szi * b12;
a2 = szr * b22;
a2i = szi * b22;
goto I495;

I492:
a1 = cz * a11 + szr * a12;
a1i = szi * a12;
a2 = cz * a21 + szr * a22;
a2i = szi * a22;

//......... choose complex q .........
I495:
cq = sqrt(a1*a1+a1i*a1i);
if (cq==0.0) goto I497;
sqr = (a1 * a2 + a1i * a2i) / cq;
sqi = (a1 * a2i - a1i * a2) / cq;
r = sqrt(cq*cq+sqr*sqr+sqi*sqi);
cq = cq / r;
sqr = sqr / r;
sqi = sqi / r;
goto I500;

I497:
sqr = 1.0;
sqi = 0.0;

//......... compute diagonal elements that would result
//          if transformations were applied .........
I500:
ssr = sqr * szr + sqi * szi;
ssi = sqr * szi - sqi * szr;
i = 1;
tr = cq * cz * a11 + cq * szr * a12 + sqr * cz * a21 + ssr * a22;
ti = cq * szi * a12 - sqi * cz * a21 + ssi * a22;
dr = cq * cz * b11 + cq * szr * b12 + ssr * b22;
di = cq * szi * b12 + ssi * b22;
goto I503;

I502:
i = 2;
tr = ssr * a11 - sqr * cz * a12 - cq * szr * a21 + cq * cz * a22;
ti = -ssi * a11 - sqi * cz * a12 + cq * szi * a21;
dr = ssr * b11 - sqr * cz * b12 + cq * cz * b22;
di = -ssi * b11 - sqi * cz * b12;

I503:
t = ti * dr - tr * di;
j = na;
if (t<0.0) j = en;
r = sqrt(dr*dr+di*di);
beta(j) = bn * r;
alfr(j) = an * (tr * dr + ti * di) / r;
alfi(j) = an * t / r;
if (i==1) goto I502;
```

```
      isw = 3 - isw;
    }
    b(n,1) = epsb;
  }

  /*   this subroutine is the optional fourth step of the qz algorithm
  c      for solving generalized matrix eigenvalue problems,
  c      siam j. numer. anal. 10, 241-256(1973) by moler and stewart.
  c
  c      this subroutine accepts a pair of real matrices, one of them in
  c      quasi-triangular form (in which each 2-by-2 block corresponds to
  c      a pair of complex eigenvalues) and the other in upper triangular
  c      form.  it computes the eigenvectors of the triangular problem and
  c      transforms the results back to the original coordinate system.
  c      it is usually preceded by  qzhes,  qzit, and  qzval.
  c
  c      on input
  c
  c         nm must be set to the row dimension of two-dimensional
  c           array parameters as declared in the calling program
  c           dimension statement.
  c
  c         n is the order of the matrices.
  c
  c         a contains a real upper quasi-triangular matrix.
  c
  c         b contains a real upper triangular matrix.  in addition,
  c           location b(n,1) contains the tolerance quantity (epsb)
  c           computed and saved in  qzit.
  c
  c         alfr, alfi, and beta  are vectors with components whose
  c           ratios ((alfr+i*alfi)/beta) are the generalized
  c           eigenvalues.  they are usually obtained from  qzval.
  c
  c         z contains the transformation matrix produced in the
  c           reductions by  qzhes,  qzit, and  qzval, if performed.
  c           if the eigenvectors of the triangular problem are
  c           desired, z must contain the identity matrix.
  c
  c      on output
  c
  c         a is unaltered.  its subdiagonal elements provide information
  c            about the storage of the complex eigenvectors.
  c
  c         b has been destroyed.
  c
  c         alfr, alfi, and beta are unaltered.
  c
  c         z contains the real and imaginary parts of the eigenvectors.
  c           if alfi(i) .eq. 0.0, the i-th eigenvalue is real and
  c             the i-th column of z contains its eigenvector.
  c           if alfi(i) .ne. 0.0, the i-th eigenvalue is complex.
  c             if alfi(i) .gt. 0.0, the eigenvalue is the first of
  c               a complex pair and the i-th and (i+1)-th columns
  c               of z contain its eigenvector.
  c             if alfi(i) .lt. 0.0, the eigenvalue is the second of
  c               a complex pair and the (i-1)-th and i-th columns
  c               of z contain the conjugate of its eigenvector.
  c           each eigenvector is normalized so that the modulus
  c           of its largest component is 1.0 .
  c
  c      questions and comments should be directed to burton s. garbow,
  c      mathematics and computer science div, argonne national laboratory
  c
  c      this version dated 12/7/97
```

```
void
qzvec(matrix& a,matrix& b,matrix& alfr,matrix& alfi,matrix& beta,matrix& z)
{
   int i,j,k,m,n,en,ii,jj,na,nn,isw,enm2;
   double d,q,r,s,t,w,x,y,di,dr,ra,rr,sa,ti,tr,t1,t2,w1,x1,
            zz,z1,alfm,almi,almr,betm,epsb;
   n=a.n;
   epsb = b(n,1);
   isw = 1;
   //......... for en=n step -1 until 1 do -- ..........
   for(nn=1;nn<=n;nn++)
   {
      en = n + 1 - nn;
      na = en - 1;
      if (isw==2) goto I795;
      if (alfi(en)!=0.0) goto I710;

      //.......... real vector ..........
      m = en;
      b(en,en) = 1.0;
      if (na==0) continue;
      alfm = alfr(m);
      betm = beta(m);

      //.......... for i=en-1 step -1 until 1 do -- ..........
      for(ii=1;ii<=na;ii++)
      {
         i = en - ii;
         w = betm * a(i,i) - alfm * b(i,i);
         r = 0.0;

         for(j=m;j<=en;j++)
            r = r + (betm * a(i,j) - alfm * b(i,j)) * b(j,en);

         if (i==1||isw==2) goto I630;
         if (betm * a(i,i-1)==0.0) goto I630;
         zz = w;
         s = r;
         goto I690;

         I630:
         m = i;
         if (isw==2) goto I640;
         //.......... real 1-by-1 block ..........
         t = w;
         if (w==0.0) t = epsb;
         b(i,en) = -r / t;
         continue;

         //.......... real 2-by-2 block ..........
         I640:
         x = betm * a(i,i+1) - alfm * b(i,i+1);
         y = betm * a(i+1,i);
         q = w * zz - x * y;
         t = (x * s - zz * r) / q;
         b(i,en) = t;
         if (fabs(x)<=fabs(zz)) goto I650;
         b(i+1,en) = (-r - w * t) / x;
         goto I690;

         I650:
         b(i+1,en) = (-s - y * t) / zz;

         I690:
}

//.......... end real vector ..........
continue;

//.......... complex vector ..........
I710:
m = na;
almr = alfr(m);
almi = alfi(m);
betm = beta(m);
//.......... last vector component chosen imaginary so that
//           eigenvector matrix is triangular ..........
y = betm * a(en,na);
b(na,na) = -almi * b(en,en) / y;
b(na,en) = (almr * b(en,en) - betm * a(en,en)) / y;
b(en,na) = 0.0;
b(en,en) = 1.0;
enm2 = na - 1;
if (enm2==0) goto I795;

//.......... for i=en-2 step -1 until 1 do -- ..........
for(ii=1;ii<=enm2;ii++)
{
   i = na - ii;
   w = betm * a(i,i) - almr * b(i,i);
   w1 = -almi * b(i,i);
   ra = 0.0;
   sa = 0.0;

   for(j=m;j<=en;j++)
   {
      x = betm * a(i,j) - almr * b(i,j);
      x1 = -almi * b(i,j);
      ra = ra + x * b(j,na) - x1 * b(j,en);
      sa = sa + x * b(j,en) + x1 * b(j,na);
   }
   if (i==1||isw==2) goto I770;
   if (betm * a(i,i-1)==0.0) goto I770;
   zz = w;
   z1 = w1;
   r = ra;
   s = sa;
   isw = 2;
   continue;

   I770:
   m = i;
   if (isw==2) goto I780;

   //.......... complex 1-by-1 block ..........
   tr = -ra;
   ti = -sa;

   I773:
   dr = w;
   di = w1;

   //.......... complex divide (t1,t2) = (tr,ti) / (dr,di) ..........
   I775:
   if (fabs(di)>fabs(dr)) goto I777;
   rr = di / dr;
   d = dr + di * rr;
   t1 = (tr + ti * rr) / d;
   t2 = (ti - tr * rr) / d;
   if(isw==1) goto I787;
```

```
      I777:
      rr = dr / di;
      d = dr * rr + di;
      t1 = (tr * rr + ti) / d;
      t2 = (ti * rr - tr) / d;
      if(isw==1) goto I787;
      else if(isw==2) goto I782;

      //.......... complex 2-by-2 block ..........
      I780:
      x = betm * a(i,i+1) - almr * b(i,i+1);
      x1 = -almi * b(i,i+1);
      y = betm * a(i+1,i);
      tr = y * ra - w * r + w1 * s;
      ti = y * sa - w * s - w1 * r;
      dr = w * zz - w1 * z1 - x * y;
      di = w * z1 + w1 * zz - x1 * y;
      if (dr==0.0&&di==0.0) dr = epsb;
      goto I775;

      I782:
      b(i+1,na) = t1;
      b(i+1,en) = t2;
      isw = 1;
      if (fabs(y)>(fabs(w) + fabs(w1))) goto I785;
      tr = -ra - x * b(i+1,na) + x1 * b(i+1,en);
      ti = -sa - x * b(i+1,en) - x1 * b(i+1,na);
      goto I773;

      I785:
      t1 = (-r - zz * b(i+1,na) + z1 * b(i+1,en)) / y;
      t2 = (-s - zz * b(i+1,en) - z1 * b(i+1,na)) / y;

      I787:
      b(i,na) = t1;
      b(i,en) = t2;
   }

   //.......... end complex vector ..........
   I795:
   isw = 3 - isw;
}

//.......... end back substitution.
//           transform to original coordinate system.
//           for j=n step -1 until 1 do -- ..........
for(jj=1;jj<=n;jj++)
{
   j = n + 1 - jj;

   for(i=1;i<=n;i++)
   {
      zz = 0.0;

      for(k=1;k<=j;k++)
         zz = zz + z(i,k) * b(k,j);

      z(i,j) = zz;
   }
}

//.......... normalize so that modulus of largest
//           component of each vector is 1.
//           (isw is 1 initially from before) ..........
for(j=1;j<=n;j++)
```

```
      if (isw==2) goto I920;
      if (alfi(j)!=0.0) goto I945;

      for(i=1;i<=n;i++)
         if (fabs(z(i,j))>d) d = fabs(z(i,j));

      for(i=1;i<=n;i++)
         z(i,j) = z(i,j) / d;

      continue;

      I920:
      for(i=1;i<=n;i++)
      {
         r = fabs(z(i,j-1)) + fabs(z(i,j));
         if (r!=0.0) r = r * sqrt(pow(z(i,j-1)/r,2.0)+pow(z(i,j)/r,2.0));
         if (r>d) d = r;
      }

      for(i=1;i<=n;i++)
      {
         z(i,j-1) = z(i,j-1) / d;
         z(i,j) = z(i,j) / d;
      }

      I945:
      isw = 3 - isw;
   }
}

/*
   this subroutine calls the recommended sequence of
   subroutines from the eigensystem subroutine package (eispack)
   to find the eigenvalues and eigenvectors (if desired)
   for the real general generalized eigenproblem  ax = (lambda)bx.

   on input

      nm  must be set to the row dimension of the two-dimensional
      array parameters as declared in the calling program
      dimension statement.

      n  is the order of the matrices  a  and  b.

      a  contains a real general matrix.

      b  contains a real general matrix.

      matz  is an integer variable set equal to zero if
      only eigenvalues are desired.  otherwise it is set to
      any non-zero integer for both eigenvalues and eigenvectors.

   on output

      alfr  and  alfi  contain the real and imaginary parts,
      respectively, of the numerators of the eigenvalues.

      beta  contains the denominators of the eigenvalues,
      which are thus given by the ratios  (alfr+i*alfi)/beta.
      complex conjugate pairs of eigenvalues appear consecutively
      with the eigenvalue having the positive imaginary part first.

      z  contains the real and imaginary parts of the eigenvectors
      if matz is not zero.  if the j-th eigenvalue is real, the
      j-th column of  z  contains its eigenvector.  if the j-th
      j-th and (j+1)-th columns of  z  contain the real and
      imaginary parts of its eigenvector.  the conjugate of this
      vector is the eigenvector for the conjugate eigenvalue.

      ierr  is an integer output variable set equal to an error
         completion code described in the documentation for qzit.
         the normal completion code is zero.

   questions and comments should be directed to burton s. garbow,
   mathematics and computer science div, argonne national laboratory

     this version dated 12/7/97
      -> converted to Borland C++ by Joeleff Fitzsimmons, UTIAS
*/

int
rgg(matrix& a,matrix& b,matrix& alfr,matrix& alfi,matrix& beta,
      const int matz,matrix& z)
{
   int ierr,tf;

   if (a.n > a.m) ierr = 10 * a.n;
   else
   {
      if (matz == 0)
      {
         //......... find eigenvalues only .........
         tf = 0;
         qzhes(a,b,tf,z);
         ierr=qzit(a,b,0.0,tf,z);
         qzval(a,b,alfr,alfi,beta,tf,z);
      }else{
         //......... find both eigenvalues and eigenvectors .........
         tf = 1;
         qzhes(a,b,tf,z);
         ierr=qzit(a,b,0.0,tf,z);
         qzval(a,b,alfr,alfi,beta,tf,z);
         if (ierr == 0)
            qzvec(a,b,alfr,alfi,beta,z);
      }
   }
   return ierr;
}
```

```
#ifndef _ALTITDEF_H_
#define _ALTITDEF_H_

#include <classlib/arrays.h>

struct tagALTIT {
   double Alt, Dens, KinVisc, Wind;
};
typedef struct tagALTIT ALTIT;

class Altitude : public tagALTIT
{
public:
   Altitude() {Alt = 0.0; Dens = 0.0; KinVisc = 0.0; Wind = 0.0;}
   Altitude( double A, double D=0.0, double K=0.0, double W=0.0)
         {Alt=A; Dens=D; KinVisc=K; Wind=W;}
   Altitude(const Altitude far& A)
         {Alt=A.Alt; Dens=A.Dens; KinVisc=A.KinVisc; Wind=A.Wind;}
   void Set( double A, double D=0.0, double K=0.0, double W=0.0)
         {Alt=A; Dens=D; KinVisc=K; Wind=W;}

   // Copy functions/operators
   void  operator=(const Altitude far& A)
         {Alt=A.Alt; Dens=A.Dens; KinVisc=A.KinVisc; Wind=A.Wind;}
   int   operator ==(const Altitude far& A) {return Alt == A.Alt;}
   int   operator <(const Altitude far& A) {return Alt < A.Alt;}

};
inline ostream& operator <<(ostream& os, const Altitude& A)
  { return os << '(' << A.Alt << ',' << A.Dens << ',' << A.KinVisc
      << ',' << A.Wind << ')'; }
inline istream& operator >>(istream& is, Altitude& A)
  { char c; return is >> c >> A.Alt >> c >> A.Dens >> c >> A.KinVisc
      >> c >> A.Wind >> c; }

typedef TArray<Altitude> AltitudeArray;
typedef TArrayIterator<Altitude> AltitudeArrayIterator;

class AltitudeData : public AltitudeArray
{
public:
   AltitudeData() : AltitudeArray(10, 0, 10) {}
   ~AltitudeData() {}
   void GetData(Altitude& A, AltitudeData& AD);
   double Q(double Alt);
   // The == operator must be defined for the container class, even if unused
   bool operator ==(const AltitudeData& other) const
   {
      return &other == this;
   }

   // Save the minimum and maximum altitudes
   double MinAlt,MaxAlt;

   // Save the number of entries
   int numEntries;

   friend ostream& operator <<(ostream& os, const AltitudeData& table);
   friend istream& operator >>(istream& is, AltitudeData& table);

};

#endif
```

```
//
//Altitude and AltitudeData Class Definitions
//

#include <classlib/arrays.h>
#include <fstream.h>
#include <math.h>
#include <float.h>
#include "altitdef.h"

void
AltitudeData::GetData(Altitude& A, AltitudeData& AD)
{
   AltitudeArrayIterator i(AD);
   int j=0;
   while((i.Current()).Alt<A.Alt&&(i.Current()).Alt!=0.0)
   {
      i++;
      j++;
   }
   if((i.Current()).Alt==A.Alt)
      A=i.Current();
   else if((i.Current()).Alt==0.0){
      i.Restart(j-1, j);
      A=i.Current();
   }else{
      double lrr;
      Altitude lower,upper;
      upper=i.Current();
      i.Restart(j-1, j);
      lower=i.Current();
      lrr=(A.Alt-lower.Alt)/(upper.Alt-lower.Alt);
      A.Dens=lrr*(upper.Dens-lower.Dens)+lower.Dens;
      A.KinVisc=lrr*(upper.KinVisc-lower.KinVisc)+lower.KinVisc;
      A.Wind=lrr*(upper.Wind-lower.Wind)+lower.Wind;
   }
}

double
AltitudeData::Q(double Alt)
{
   AltitudeArrayIterator i(*this);
   double dens,wind;
   int j=0;
   while((i.Current()).Alt<Alt&&(i.Current()).Alt!=0.0)
   {
      i++;
      j++;
   }
   if((i.Current()).Alt==Alt)
   {
      dens=(i.Current()).Dens;
      wind=(i.Current()).Wind;
   }
   else if((i.Current()).Alt==0.0)
   {
      i.Restart(j-1, j);
      dens=(i.Current()).Dens;
      wind=(i.Current()).Wind;
   }
   else
   {
      double lrr;
      Altitude lower,upper;
      upper=i.Current();
      lower=i.Current();
      lrr=(Alt-lower.Alt)/(upper.Alt-lower.Alt);
      dens=lrr*(upper.Dens-lower.Dens)+lower.Dens;
      wind=lrr*(upper.Wind-lower.Wind)+lower.Wind;
   }
   return (0.5*dens*pow(wind,2.0));
}

ostream&
operator <<(ostream& os, const AltitudeData& table)
{
//  _fpreset();
  // Write the number of entries in the table
  os << table.numEntries;

  // Get an iterator for the array of points
  AltitudeArrayIterator j(table);

  // While the iterator is valid (i.e. we haven't run out of entries)
  while(j)
    // Write the entry from the iterator and increment the array.
    os << "\n" << j++;

  // return the stream object
  return os;
}

istream&
operator >>(istream& is, AltitudeData& table)
{
  int i;
  _fpreset();

  is >> i;
  table.numEntries=i;

  while (i--) {
    Altitude A;
    is >> A;
    table.Add(A);
    if(i==table.numEntries-1)
      table.MinAlt=A.Alt;
    if(!i)
      table.MaxAlt=A.Alt;
  }

  // return the stream object
  return is;
}
```

```
//
// Kite Data Class Definition
//
#ifndef _SUPERKITE_H_
#define _SUPERKITE_H_
#include <math.h>
#include <complex.h>
#include <classlib/arrays.h>
#include "matrix.h"
#include "altitdef.h"

void sort3(matrix& arr, matrix& brr, matrix& crr);

class ControlGain {
   public:
      ControlGain()
      {
         Ccx=Ccy=Ccz=Ccl=Ccm=Ccn=0.0;
      }
      void operator=(ControlGain &CG);
      void Zero(void);

      friend ostream& operator <<(ostream& os, ControlGain& CG);
      friend istream& operator >>(istream& is, ControlGain& CG);

      double Ccx,Ccy,Ccz,Ccl,Ccm,Ccn;
};

typedef TArray<double> TProfileVector;
typedef TArrayIterator <double> TViter;

class KiteData
{
public:
   KiteData();
   ~KiteData();
   char* FileForm(void);
   void FileForm(char* FF);
   void KiteAlt(double A);
   double WindFunc(double alt);
   double KiteAlt(void);
   void CalculateNonDims(AltitudeData& AltData);
   void KiteCValues(void);
   void KiteqRN(const Altitude& A);
   void SetControl(const ControlGain& CG);
   void GetControl(ControlGain& CG);
   void SpecWind(AltitudeData& AD);
   double KiteTrimAngle(void);
   void KiteTetherProfile(void);
   int KiteSRoots(ControlGain& CG, AltitudeData& AltData,
                  bool CheckEV, bool MLOut, int matz=1);
   void TimeStepAnalysis(ControlGain& CG, AltitudeData& AltData, bool TSOut,
                  bool KCMOut = false);
   int iswindfile(void);

   friend ostream& operator <<(ostream& os, KiteData& KD);
   friend istream& operator >>(istream& is, KiteData& KD);

   //eq. relating tension and element drag to new segment angle
   //eq. = 0 when correct angle is input
   inline double tangle(double angle) {return (2.0*Tx*sin(angle*pi/180.0)+
      (tmperl*teleml-2.0*Ty)*cos(angle*pi/180.0)+tCDcyl*q*tdia*teleml*
      pow(sin(angle*pi/180.0),2.0));}

   //Returns 0.5 if i==j else returns 1.0
```

```
    [illegible]
    //Returns 1.0 if i==j else returns 0.0
    inline double delta(int i,int j) {return (i==j ? 1.0 : 0.0);}

    //Super Kite Specified Configuration
    TProfileVector *Profile;
    int N,aoaexceed,aoaset,aoasetomax,EiglgValid,EigltValid,IWF;
    double pi,g,Nc,Clmax,aoamax,aoalim,tGammallim,deltaht;
    double mass,e,Xcg,Zcg,Xcplg,Zcplg,Ixz,Ixx,Iyy,Izz,b,croot,ctip,awing,Cmac,
           Thick,Dih,LEswp,Xcplt,Zcplt;
    double QCswp,iwing,azll,Xac,Zac,hacwb,Eo,Ki,dSiglsdP,CLp_plan,dSiglsdr;
    double bht,cht,aht,Thickht,Xht,Zht,iht,azllht,Netaht,bvt,cvt,avt,Tbar,
           bvt2,cvt2,avt2,Gammabar;
    double Thickvt,Xvt,Zvt,Netavt,dSigfdP,dSigfdr,dSigmadB,AOA,rcl,rcd,
           ralt,rsalt,Thickvt2,Xvt2,Zvt2,Netavt2;
    double Xcv,Volfs,fsw,fsh,fsqw,fsqh,fstqw,fstqh,cfs,lfs,sigfs_B,afs,Zfscp;
    double tdens,tethE,tdia,tdiao,teleml,tCDcyl,tCDax,Tx,Ty,tmperl,windvel,
           ttlen,VH,VH2,bht2,cht2,aht2,Thickht2,Xht2,Zht2,iht2,azllht2,Netaht2;
    double *Lgalfr,*Lgalfi,*Lgbeta,*Ltalfr,*Ltalfi,*Ltbeta,*WFA;
    double Cxu,Czu,Cmu,CLb,CNb,Cyp,CLp,CNp,CLr,Cyr,CNr,Cmq,Czq,Czadot,Cmadot,Cyb;
    double Cxalpha,Cxqls,Czalpha,Cmalpha;
    complex *Lgcheck,*Ltcheck;
    matrix AA,BB,Lgz,Ltz;
    matrixc Lgmodestretch,Lgmodeangle,Lgmodes,Ltmodes;
    //Time Step Variables
    matrix LgTS,LtTS;
    double XTS,ZTS,MTS,XTSDur,ZTSDur,MTSDur,TimeStep,TotalTime;
    double YTS,LTS,NTS,YTSDur,LTSDur,NTSDur;
    int XTSSet,ZTSSet,MTSSet,YTSSet,LTSSet,NTSSet;

 private:
    char FileName_Formula[150];

    //Super Kite Calculated Values
    double Alttd,W,cmean,tratio,S,AR,fratio,lht,Sht,fratioht,ARht,
           lvt,Svt,lvt2,Svt2,fratiovt2,fratiovt;
    double Sfs,fratiofs,K2_K1,ea,ea2,Kac,Kfs,Kht,Kht2,KB,RNfs,RNw,RNht,RNvt,q,
           CMfs,CLac,Xnp,Znp,lht2,Sht2,fratioht2,ARht2,RNht2,Kvt,Kvt2,RNvt2;
    double CLht,CLht2,CDfs,CDac,CDht,CDht2,CDvt,CDvt2,Temp,dEpsilondq,Nfg,Nls,
           Nls1,CLo,CNbCL,CLppf,Cnp1;
    double Cnp2,CDpls,CLrCLo,Cnr1,Cnr2,KA,Klam,KH,dEpsilonda,CWo,Mach,
           CNrfac2,CNrf2,KH2,dEpsilonda2,Czadott2;
    double Cdof,Cybfs,Cybf,Cybbody,Cybls,CLbf,CLbls,CLbbody,CNbbody;
    double CNbf,CNblsac,CNbls,Cypbody,Cypf,Cyplsac,Cypls,CLpbody;
    double CLpf,CLpls,CLplsac,CNpbody,CNpf,CNplsac,CNpls,CLbf2,CNbf2;
    double Cyrfac,Cyrf,Cyrls,CLrf,CLrCL,CLrlsac,CLrls,cswp,B,Cmcpo;
    double CNrfac,CNrf,CNrlsac,CNrls,Yv,Yp,Yr,Lv,Lp,Lr,Nv,Np,Nr,CNpf2,Cyrf2;
    double Czalphals,Czqlsac,Czqls,Czqt,Cyrfac2;
    double Cxalphals,Cxqlsac,Czadotbody,Czadotls,Czadott,CLrf2;
    double Cmadotbody,Cmadotls,Cmadott,Cxcpo,Czcpo,Cmqlsac,Cmqls,Cmqt;
    double Xu,Xw,Zu,Zw,Zwdot,Zq,Mu,Mw,Mwdot,Mq,Cybfs2,Cybf2,Cypf2,CLpf2;
    double CGx,CGy,CGz,CGl,CGm,CGn;
 };

 class VarParam {
    public:
       VarParam();
       ~VarParam();

       int LgActive,LtActive,NSteps;
       double LgStart,LgInc,LtStart,LtInc;
 };

 class KiteQuery {
   public:
       KiteQuery();
       ~KiteQuery();
```

```
        void GetVarParam(VarParam& VP, ControlGain& VCG, KiteData& KD);
        void SetArraySize(double N);
        void ResetKiteData(KiteData& KD);
        void operator=(KiteQuery &KQ);

        int LgActive,LtActive;
        double StorageB,StorageX;
        matrixi Steps;
        matrix Start,End;
        matrixc LgRoots,LtRoots;
        char LgParam[50], LtParam[50];
};

class Scale {
    public:
        Scale();
        void SetScale(double& ymax, double& xmax, double& xmin);

        bool YMaxSet,XMaxSet,XMinSet;
        double YMax,XMax,XMin;
};

#endif
```

```
//
//Super Kite and Environment Initialization
//

#include <stdio.h>
#include <stdlib.h>
#include <cstring.h>
#include <math.h>
#include <classlib/arrays.h>
#include <float.h>
#include <fstream.h>
#include "matrix.h"
#include "rggpool.h"
#include "suprkite.h"

//Overload the = operator to copy two ControlGain objects
void
ControlGain::operator=(ControlGain &CG)
{
   Ccx=CG.Ccx;
   Ccy=CG.Ccy;
   Ccz=CG.Ccz;
   Ccl=CG.Ccl;
   Ccm=CG.Ccm;
   Ccn=CG.Ccn;
}

// Set all of the control gains to zero
void
ControlGain::Zero(void)
{
   Ccx=0.0;
   Ccy=0.0;
   Ccz=0.0;
   Ccl=0.0;
   Ccm=0.0;
   Ccn=0.0;
}

ostream&
operator <<(ostream& os, ControlGain& CG)
{
   _fpreset();
   os << CG.Ccx << "\n";
   os << CG.Ccy << "\n";
   os << CG.Ccz << "\n";
   os << CG.Ccl << "\n";
   os << CG.Ccm << "\n";
   os << CG.Ccn << "\n";

   // return the stream object
   return os;
}

istream&
operator >>(istream& is, ControlGain& CG)
{
   _fpreset();

   is >> CG.Ccx;
   is >> CG.Ccy;
   is >> CG.Ccz;
   is >> CG.Ccl;
   is >> CG.Ccm;
   is >> CG.Ccn;
```

```
  // return the stream object
  return is;
}

VarParam::VarParam()
{
   NSteps=1;
   LgActive=1;
   LtActive=4;
   LgStart=LgInc=LtStart=LtInc=0.0;
}

VarParam::~VarParam()
{

}

KiteQuery::KiteQuery()
{
   StorageB=0.0;
   StorageX=0.0;
   LgActive=1;
   LtActive=4;
   Start(8);End(8);Steps(8);
   strcpy(LgParam,"");
   strcpy(LtParam,"");
   for(int i=1;i<=6;i++)
   {
      Start(i)=-10.0;
      End(i)=10.0;
      Steps(i)=20;
   }
   Start(7)=0.2;
   End(7)=0.3;
   Steps(7)=20;
   Start(8)=-0.5;
   End(8)=0.5;
   Steps(8)=20;

}

KiteQuery::~KiteQuery()
{

}

//Overload the = operator to copy two ControlGain objects
void KiteQuery::operator=(KiteQuery &KQ)
{
   LgActive=KQ.LgActive;
   LtActive=KQ.LtActive;
   for(int i=1;i<=8;i++)
   {
      Start(i)=KQ.Start(i);
      End(i)=KQ.End(i);
      Steps(i)=KQ.Steps(i);
   }
   strcpy(LgParam,KQ.LgParam);
   strcpy(LtParam,KQ.LtParam);
}

void
KiteQuery::SetControl(ControlGain& CG, KiteData& KD)
{
   int i;
   double origcg;
   for(i=1;i<=8;i++)
   {
      switch(i)
      {
         case 1:
            origcg=CG.Ccx;
            break;
         case 2:
            origcg=CG.Ccz;
            break;
         case 3:
            origcg=CG.Ccm;
            break;
         case 4:
            origcg=CG.Ccy;
            break;
         case 5:
            origcg=CG.Ccl;
            break;
         case 6:
            origcg=CG.Ccn;
            break;
         case 7:
            Start(7)=KD.bvt2-0.3;
            if(Start(7)<0.0) Start(7)=0.0;
            End(7)=KD.bvt2+0.3;
            break;
         case 8:
            Start(8)=KD.Xvt2-0.5;
            End(8)=KD.Xvt2+0.5;
            break;
      }

      if(i<=6)
      {
         Start(i)=origcg-10.0;
         End(i)=origcg+10.0;
      }
      Steps(i)=20;
   }
}

void
KiteQuery::GetVarParam(VarParam& VP, ControlGain& VCG, KiteData& KD)
{
   VP.LgActive=LgActive;
   VP.LtActive=LtActive;
   StorageB=KD.bvt2;
   StorageX=KD.Xvt2;

   switch(LgActive)
   {
      case 1:
         VP.LgStart=VCG.Ccx=Start(1);
         VP.NSteps=Steps(1);
         VP.LgInc=(End(1)-Start(1))/VP.NSteps;
         break;
      case 2:
         VP.LgStart=VCG.Ccz=Start(2);
         VP.NSteps=Steps(2);
         VP.LgInc=(End(2)-Start(2))/VP.NSteps;
         break;
      case 3:
         VP.LgStart=VCG.Ccm=Start(3);
         VP.NSteps=Steps(3);
         VP.LgInc=(End(3)-Start(3))/VP.NSteps;
```

```
    }

    switch(LtActive)
    {
       case 4:
          VP.LtStart=VCG.Ccy=Start(4);
          VP.LtInc=(End(4)-Start(4))/VP.NSteps;
          break;
       case 5:
          VP.LtStart=VCG.Ccl=Start(5);
          VP.LtInc=(End(5)-Start(5))/VP.NSteps;
          break;
       case 6:
          VP.LtStart=VCG.Ccn=Start(6);
          VP.LtInc=(End(6)-Start(6))/VP.NSteps;
          break;
       case 7:
          VP.LtStart=KD.bvt2=Start(7);
          VP.LtInc=(End(7)-Start(7))/VP.NSteps;
          break;
       case 8:
          VP.LtStart=KD.Xvt2=Start(8);
          VP.LtInc=(End(8)-Start(8))/VP.NSteps;
          break;
    }
}

void
KiteQuery::SetArraySize(double N)
{
    LgRoots.Resize(N*4+2,Steps(LgActive)+1);
    LtRoots.Resize(N*2+4,Steps(LgActive)+1);
}

void
KiteQuery::ResetKiteData(KiteData& KD)
{
    KD.bvt2=StorageB;
    KD.Xvt2=StorageX;
}

KiteData::KiteData()
{
    Profile        = new TProfileVector(10,0,100);
    IWF=1;
    pi=3.14159265359;
    g=9.81;

    strcpy(FileName_Formula,"wind10_0.wdf");

    //Initialize the Wind Data Function Array
    WFA = new double[13];
    Lgalfr = new double[1];
    Lgalfi = new double[1];
    Ltalfr = new double[1];
    Ltalfi = new double[1];
    Lgcheck = new complex[1];
    Ltcheck = new complex[1];

    //Specify the eigen value solution invalid
    EiglgValid=0;
    EigltValid=0;

    //Specify not to force the AOA and specify horiz. tail aoa
    aoaset=0;
    aoasetomax=0;
```

```
//Set default values for the Super Kite
//All values are defined using Metric as standard

AOA=0.0;                    //Kite Trim Angle of Attack
Alttd=ralt=5100.0;          //Kite Trim Altitude
mass = 70.0;                //Kite mass
e = 0.95;                   //Kite efficiency factor
Xcg = 1.50;                 //For-Aft CG location (from nose)
Zcg = 0.05;                 //Vertical CG location (from nose)
Xcplg = 1.39;               //For-Aft Confluence point location (long)
Zcplg = -0.50;              //Vertical Confluence point location (long)
Xcplt = 1.39;               //For-Aft Confluence point location (lat)
Zcplt = -0.50;              //Vertical Confluence point location (lat)
b = 15.00;                  //Wing span (m)
croot = 1.50;               //Wing root chord (m)
ctip = 1.50;                //Wing tip chord (m)
awing = 5.00;               //Wing lift curve slope (/rad)
Clmax = 2.0;                //Wing Stall Coeff. of lift
Cmac = -0.10;               //Wing coeff. of moment about the AC
Thick = 13.0;               //Wing airfoil thickness (% of chord)
Dih = 5.0;                  //Wing dihedral angle (degrees)
LEswp = 0.0;                //Wing LE sweep angle (degrees)
QCswp = 0.0;                //Wing 1/4 chord sweep angle (degrees)
iwing = 8.0;                //Wing incidence angle (degrees)
azll = 13.0;                //Wing zero lift AOA (degrees)
Xac = 1.50;                 //Wing AC location aft of nose (m)
Zac = 0.20;                 //Wing AC Vertical location (m)
hacwb = 0.30;               //Position of AC on Wing
Eo = 1.0;                   //Zero lift downwash angle
dSiglsdP = 0.0;             //Side wash due to rolling of lifting surface(ls)
CLp_plan = 0.0;             //Roll damping effect of planform
dSiglsdr = 0.0;             //side wash due to yawing of lifting surface
bht = 3.00;                 //Horz. Stab. span (m)
cht = 0.50;                 //Horz. Stab. Chord (m)
aht = 4.00;                 //Horz. Stab. lift curve slope (/rads)
Thickht = 10.0;             //Horz. Stab. airfoil max thickness (% chord)
Xht = 6.00;                 //Horz. Stab. for-aft location from nose (m)
Zht = 0.10;                 //Horz. Stab. vertical location (m)
iht = 0.0;                  //Horz. Stab. incidence angle (deg)
azllht = 0.0;               //Horz. Stab. zero lift AOA (deg)
Netaht = 0.98;              //Horz. Stab. dynamic pressure effic.
bht2 = 0.00;                //Horz. Stab. 2 span (m)
cht2 = 0.50;                //Horz. Stab. 2 Chord (m)
aht2 = 4.00;                //Horz. Stab. 2 lift curve slope (/rads)
Thickht2 = 10.0;            //Horz. Stab. 2 airfoil max thickness (% chord)
Xht2 = 6.00;                //Horz. Stab. 2 for-aft location from nose (m)
Zht2 = 0.10;                //Horz. Stab. 2 vertical location (m)
iht2 = 0.0;                 //Horz. Stab. 2 incidence angle (deg)
azllht2 = 0.0;              //Horz. Stab. 2 zero lift AOA (deg)
Netaht2 = 0.98;             //Horz. Stab. 2 dynamic pressure effic.
bvt = 1.25;                 //Vert. Stab. total height (m)
cvt = 0.50;                 //Vert. Stab. mean chord (m)
avt = 4.00;                 //Vert. Stab. lift curve slope (/rad)
Thickvt = 8.0;              //Vert. Stab. airfoil max. thickness (% chord)
Xvt = 6.00;                 //Vert. Stab. mean 1/4 chord for-aft dist. (m)
Zvt = 0.60;                 //Vert. Stab. AC location Vertically (m)
Netavt = 1.00;              //Vert. Stab. dynamic pressure effic.
bvt2 = 0.0;                 //Vert. Stab. 2 total height (m)
cvt2 = 0.75;                //Vert. Stab. 2 mean chord (m)
avt2 = 4.00;                //Vert. Stab. 2 lift curve slope (/rad)
Thickvt2 = 8.0;             //Vert. Stab. 2 airfoil max. thickness (% chord)
Xvt2 = 6.00;                //Vert. Stab. 2 mean 1/4 chord for-aft dist. (m)
Zvt2 = 0.60;                //Vert. Stab. 2 AC location Vertically (m)
Netavt2 = 1.00;             //Vert. Stab. 2 dynamic pressure effic.
dSigfdP = 0.0;              //Side wash due to rolling of vertical fin
```

```
   dsigmadB = 0.0;             //Vert. Stab. Side wash factor
   Volfs = 1.00;               //Volume of fuselage (m^3)
   fsw = 0.25;                 //Fuselage max. width (m)
   fsh = 0.50;                 //Fuselage max. height (m)
   fsqw = 0.22;                //Fuselage width at 1/4 length (m)
   fsqh = 0.45;                //Fuselage height at 1/4 length (m)
   fstqw = 0.10;               //Fuselage width at 3/4 length (m)
   fstqh = 0.15;               //Fuselage height at 3/4 length (m)
   cfs = 2.00;                 //Characteristic fuselage length (m)
   lfs = 6.00;                 //Fuselage total length (m)
   sigfs_B = 0.0;              //Fuselage side wash factor
   afs = 0.0525;               //Lateral fuselage lift curve slope (/rad)
   Zfscp = 0.10;               //Fuselage lateral center of pressure height (m)
   tdens = 1010.0;             //tether density
   tethE = 10.0e9;             //tether Elastic Modulus
   tdia = 0.0014;              //tether diameter @ Confluence point
   tdiao = 0.0014;             //tether diameter @ ground
   teleml = 10.0;              //tether incremental length
   tCDcyl = 0.8;               //tether coeff. of drag in x-flow direction
   tCDax = 0.1;                //tether coeff. of drag in axial direction
   tGammallim = 0.0;           //min. allowable tether angle for trim state
   Xcv = 1.3;                  //Distance of fuse volume center from nose ref pt
   Ixz = 10.0;
   Ixx = 1500.0;
   Iyy = 1700.0;
   Izz = 2000.0;
   Nc = 1.0;                   //Internal tether damping (N*s/m)
   rcl = 0.0;                  //Resulting total Cl
   rcd = 0.0;                  //Resulting total Cd
   ralt = 0.0;                 //Resulting calculated altitude
   rsalt = 0.0;                //Resulting Specified altitude
   aoalim=0.0;                 //The AOA limit if specified
   CGx=CGy=CGz=CGl=CGm=CGn=0.0;
   TimeStep=0.5;
   TotalTime=10.0;
   XTS=ZTS=MTS=XTSDur=ZTSDur=MTSDur=0.0;
   YTS=LTS=NTS=YTSDur=LTSDur=NTSDur=0.0;
   XTSSet=ZTSSet=MTSSet=YTSSet=LTSSet=NTSSet=0;
}

KiteData::~KiteData()
{
   delete Profile;
   delete[] WFA;
   delete[] Lgalfr;
   delete[] Lgalfi;
   delete[] Ltalfr;
   delete[] Ltalfi;
   delete[] Lgcheck;
}

void
KiteData::KiteCValues(void)
{
   double Mo=0.1;                    // Approximate Mach number
   W         =g*mass;                // Weight of tharwp
   cmean     =(croot+ctip)/2.0;      // Mean chord of wing
   tratio    =ctip/croot;            // Taper ratio of wing
   S         =cmean*b;               // Wing area
   AR        =pow(b,2.0)/S;          // Wing Aspect Ratio
   fratio    =Thick/100.0;           // Fineness ratio of wing
   lht       =Xht-Xcg;               // Horizontal tail moment arm
   Sht       =bht*cht;               // Area of Horizontal tail
   fratioht =Thickht/100.0;          // Fineness ratio of Horiz. Stab.
   if(Sht) ARht=pow(bht,2.0)/Sht;    // Horizontal tail Aspect Ratio
   else ARht=0.0;
```

```
                                     // 2 Horizontal tail moment arm
Sht2      =bht2*cht2;                // 2 Area of Horizontal tail
fratioht2=Thickht2/100.0;            // 2 Fineness ratio of Horiz. Stab.
                                     // 2 Horizontal tail Aspect Ratio
if(Sht2) ARht2=pow(bht2,2.0)/Sht2;
else ARht2=0.0;
VH2=Sht2*lht2/S/cmean;               // 2 Horizontal Tail Volume Coefficient
Svt       =bvt*cvt;                  // Vertical tail area
lvt       =Xcg-Xvt;                  // Distance from wing mac to fin mac
fratiovt =Thickvt/100.0;             // Fineness ratio of Vert. Fin
Svt2      =bvt2*cvt2;                // 2 Vertical tail area
lvt2      =Xcg-Xvt2;                 // 2 Distance from wing mac to fin mac
fratiovt2=Thickvt2/100.0;            // 2 Fineness ratio of Vert. Fin
Sfs       =pow(Volfs,(2.0/3.0));     // Surface area of Fuse. based on Volume
fratiofs =cfs/fsw;                   // Fineness ratio of fuselage
if(fratiofs>11.0){
   K2_K1 = 1.0;
}else{
   K2_K1 = -1.822885+3.93388535*fratiofs-4.3548924*pow(fratiofs,2.0)+
           3.2814978*pow(fratiofs,3.0)-1.56429315*pow(fratiofs,4.0)+
           0.4801706513*pow(fratiofs,5.0)-0.0973870078*pow(fratiofs,6.0)+
           0.0131973844*pow(fratiofs,7.0)-0.0011830478889*
           pow(fratiofs,8.0)+0.00006736317705*pow(fratiofs,9.0)-
           0.00000220699477*pow(fratiofs,10.0)+0.00000003168005374*
           pow(fratiofs,11.0);
}

//downwash wrt alph (only if aft of the lifting surface)
if(Xht>Xac)
{
   ea     = 4.44*pow(((1.0/AR-1.0/(1.0+pow(AR,1.7)))*(10.0-3.0*tratio)/7.0*
           ((1.0-fabs((Zac-Zht)/b))/pow(2.0*(Xht-Xac)/b,(1.0/3.0)))*
           pow(cos(QCswp*pi/180.0),0.5)),1.19);
}else{
   ea     = 0.0;
}

//downwash wrt alph for 2nd surface (only if aft of the lifting surface)
if(Xht2>Xac)
{
   ea2    = 4.44*pow(((1.0/AR-1.0/(1.0+pow(AR,1.7)))*(10.0-3.0*tratio)/7.0*
           ((1.0-fabs((Zac-Zht2)/b))/pow(2.0*(Xht2-Xac)/b,(1.0/3.0)))*
           pow(cos(QCswp*pi/180.0),0.5)),1.19);
}else{
   ea2    = 0.0;
}

Kfs       = 3.105-0.83831*fratiofs+0.142014*pow(fratiofs,2.0)-1.10278e-2*
           pow(fratiofs,3.0)+3.19444e-4*pow(fratiofs,4.0);
Kac   = 1.0+((2.0-pow(Mo,2.0))*cos(QCswp*pi/180.0))/pow(1.0-pow(Mo,2.0)*
       pow(cos(QCswp*pi/180.0),2.0),0.5)*fratio+100.0*pow(fratio,4.0);
Kht   = 1.0+((2.0-pow(Mo,2.0))*cos(QCswp*pi/180.0))/pow(1.0-pow(Mo,2.0)*
       pow(cos(QCswp*pi/180.0),2.0),0.5)*fratioht+100.0*pow(fratioht,4.0);
Kht2  = 1.0+((2.0-pow(Mo,2.0))*cos(QCswp*pi/180.0))/pow(1.0-pow(Mo,2.0)*
       pow(cos(QCswp*pi/180.0),2.0),0.5)*fratioht2+100.0*pow(fratioht2,4.0);
Kvt   = 1.0+((2.0-pow(Mo,2.0))*cos(QCswp*pi/180.0))/pow(1.0-pow(Mo,2.0)*
       pow(cos(QCswp*pi/180.0),2.0),0.5)*fratiovt+100.0*pow(fratiovt,4.0);
Kvt2  = 1.0+((2.0-pow(Mo,2.0))*cos(QCswp*pi/180.0))/pow(1.0-pow(Mo,2.0)*
       pow(cos(QCswp*pi/180.0),2.0),0.5)*fratiovt2+100.0*pow(fratiovt2,4.0);

//Interference factor for wing in High or Low position
if(Zac>=0.0)
   Ki = 4.0*Zac/fsh;
else
   Ki = -3.0*Zac/fsh;
   //Change in Downwash with change in alpha (only if aft of lifting surface)
   if(lht>0.0)
   {
      KA=1.0/AR-1.0/(1.0+pow(AR,1.7));
      Klam=(10.0-3.0*tratio)/7.0;
      KH=(1.0-fabs((Zht-Zac)/b))/pow(2.0*lht/b,1.0/3.0);
      dEpsilonda=4.44*pow(KA*Klam*KH*pow(cos(QCswp*pi/180.0),0.5),1.19);
   }else{
      dEpsilonda=0.0;
   }

   //Change in Downwash with change in alpha for 2nd horiz. surface
   // (only if aft of lifting surface)
   if(lht2>0.0)
   {
      KA=1.0/AR-1.0/(1.0+pow(AR,1.7));
      Klam=(10.0-3.0*tratio)/7.0;
      KH2=(1.0-fabs((Zht2-Zac)/b))/pow(2.0*lht2/b,1.0/3.0);
      dEpsilonda2=4.44*pow(KA*Klam*KH*pow(cos(QCswp*pi/180.0),0.5),1.19);
   }else{
      dEpsilonda2=0.0;
   }

   //Calculate the steady state untethered Neutral point
   Xnp=cmean*VH*aht/awing*(1.0-dEpsilonda)+cmean*VH2*aht2/awing*
         (1.0-dEpsilonda2)+Xac;
   Znp=Zac/2.0;
}

void
KiteData::KiteAlt(double A)
{
   Alttd=A;
}

double
KiteData::KiteAlt(void)
{
   return Alttd;
}

void
KiteData::SpecWind(AltitudeData& AD)
{
   AltitudeArrayIterator i(AD);
   int j=0;
   double tempAlt,FminAlt,FmaxAlt,MinAlt,MaxAlt;
   FminAlt=WFA[1];
   FmaxAlt=WFA[2];
   MinAlt=AD.MinAlt;
   MaxAlt=AD.MaxAlt;
   while((i.Current()).Alt!=0.0)
   {
      tempAlt=(i.Current()).Alt;
      Altitude A;
      A=i.Current();
      if(tempAlt<FminAlt)
      {
         A.Wind=(tempAlt-MinAlt)/(FminAlt-MinAlt)*WindFunc(FminAlt/1000.0);
         AD.SetData(j,A);
      }
      else if(tempAlt>FmaxAlt)
      {
         A.Wind=(MaxAlt-tempAlt)/(MaxAlt-FmaxAlt)*WindFunc(FminAlt/1000.0);
         AD.SetData(j,A);
      else
      {
         A.Wind=WindFunc(tempAlt/1000.0);
         AD.SetData(j,A);
      }
      j++;
      i++;
   }
}

void
KiteData::KiteqRN(const Altitude& A)
{
    Alttd=A.Alt;
    windvel=A.Wind;
    q=0.5*A.Dens*pow(A.Wind,2.0);
    RNfs=A.Wind*cfs/A.KinVisc;
    RNw=A.Wind*cmean/A.KinVisc;
    RNht=A.Wind*Netaht*cht/A.KinVisc;
    RNht2=A.Wind*Netaht2*cht2/A.KinVisc;
    RNvt=A.Wind*Netavt*cvt/A.KinVisc;
    RNvt2=A.Wind*Netavt2*cvt2/A.KinVisc;
}

void
KiteData::SetControl(const ControlGain& CG)
{
   CGx=CG.Ccx;
   CGy=CG.Ccy;
   CGz=CG.Ccz;
   CGl=CG.Ccl;
   CGm=CG.Ccm;
   CGn=CG.Ccn;
}

void
KiteData::GetControl(ControlGain& CG)
{
   CG.Ccx=CGx;
   CG.Ccy=CGy;
   CG.Ccz=CGz;
   CG.Ccl=CGl;
   CG.Ccm=CGm;
   CG.Ccn=CGn;
}

double
KiteData::KiteTrimAngle(void)
{
   int i,j;
   double aa,bb,cc,det,aoa1,aoa2,x1,x2,f,fmid,dx,xmid,rtb,xacc,Alt;
   double ccp,origaoamax;
   Alt=0.0;
   deltaht=0.0;
   aoaexceed=0;
   if(aoasetomax==-1) aoasetomax=0;

   //Calculate the maximum allowable AOA before the Wing stall AOA is exceeded
   //If the AOA is specified, check to make sure it is valid
   //
   aoamax=origaoamax=Clmax/(awing*pi/180.0)-iwing-azll;
   if(aoaset&&!aoasetomax&&aoalim<=aoamax)
      aoamax=aoalim;
   else if(aoaset&&!aoasetomax&&aoalim>aoamax)
      aoasetomax=-1;          //indicate that aoalim was > the allowable aoamax
```

```
//Check to see if we are to force the AOA to a particular value (aoaset)
//and then calculate the new elevator AOA
//
if(!aoaset)
{
   //
   //Calculate trim AOA at given altitude
   //
   aa=pow(awing/180.0,2.0)*pi*q*S*(Zac-Zcplg)/(AR*e);
   bb=(2.0*K2_K1*Volfs*q+awing*q*S*(Xcplg-Xac)+aht*(1.0-ea)*q*Netaht*Sht*
       (Xcplg-Xht)+aht2*(1.0-ea2)*q*Netaht2*Sht2*(Xcplg-Xht2))*pi/180.0+
      pow(awing/180.0,2.0)*pi*q*S*(Zac-Zcplg)*(2.0*azll+2.0*iwing)/(AR*e);
   ccp=(aht*(azllht+iht-Eo-ea*(azll+iwing))*Netaht*Sht*(Xcplg-Xht)
      +aht2*(azllht2+iht2-Eo-ea2*(azll+iwing))*Netaht2*Sht2*(Xcplg-Xht2))
      *q*pi/180.0;
   cc=Cmac*q*S*cmean+awing*(azll+iwing)*q*S*(Xcplg-Xac)*pi/180.0+ccp+
      2.0*Kfs*pow(10.0,(-0.21*log10(RNfs)-0.97))*pow(Sfs,2.0)*q*(Zcg-Zcplg)
      /S+2.0*Kac*pow(10.0,(-0.21*log10(RNw)-0.97))*S*q*(Zac-Zcplg)+pow(awing
      *(azll+iwing)/180.0,2.0)*pi*q*S*(Zac-Zcplg)/(AR*e)+W*(Xcg-Xcplg);
   det=pow(bb,2.0)-4.0*aa*cc;
   if(det<0.0) AOA=-99.0;
   else{
      aoa1=(-bb+pow(det,0.5))/(2.0*aa);
      aoa2=(-bb-pow(det,0.5))/(2.0*aa);
      if(fabs(aoa1)<fabs(aoa2))
      {
         AOA=aoa1;
         if(aoa1>aoamax)
            aoaexceed=1;
      }else
      {
         AOA=aoa2;
         if(aoa2>aoamax)
            aoaexceed=1;
      }
   }
}
//Specify the trim AOA and determine the elevator aoa
else
{
   AOA=aoamax;
   aoamax=origaoamax;
   deltaht=-1.0/aht*(2.0*K2_K1*Volfs*q*AOA+Cmac*q*S*cmean*180.0/pi+awing*
       (azll+iwing+AOA)*q*S*(Xcplg-Xac)+aht*AOA*(1.0-ea)*q*Netaht*Sht*(Xcplg-
      Xht)+aht*(azllht+iht-Eo-ea*(azll+iwing))*q*Netaht*Sht*(Xcplg-Xht));
   deltaht+=-1.0/aht*(2.0*Kfs*pow(10.0,(-0.21*log10(RNfs)-0.97))*pow(Sfs,
      2.0)*q*(Zcg-Zcplg)*180.0/(S*pi)+2.0*Kac*pow(10.0,(-0.21*log10(RNw)-0.97
      ))*S*q*(Zac-Zcplg)*180.0/pi+pow(awing,2.0)/(180.0*AR*e)*pow(azll+iwing+
      AOA,2.0)*q*S*(Zac-Zcplg)+W*(Xcg-Xcplg)*180.0/pi);
}

//
//Calculate the Cm,Cl, and Cd for the kite
//
CMfs=2.0*K2_K1*Volfs*AOA*pi/(180.0*Sfs*cfs);
CLac=awing*pi/180.0*(AOA+iwing+azll);
CLht=aht*pi/180.0*(azllht+deltaht+iht+AOA-ea*(azll+iwing+AOA));
CLht2=aht2*pi/180.0*(azllht2+iht2+AOA-ea2*(azll+iwing+AOA));
CDfs=2.0*Kfs*pow(10.0,(-0.21*log10(RNfs)-0.97))*Sfs/S;
CDac=2.0*Kac*pow(10.0,(-0.21*log10(RNw)-0.97))+pow(CLac,2.0)/(pi*AR*e);
CDht=2.0*Kht*pow(10.0,(-0.21*log10(RNht)-0.97))*Sht/S+pow(CLht,2.0)/
      (pi*AR*e);
CDht2=2.0*Kht2*pow(10.0,(-0.21*log10(RNht2)-0.97))*Sht2/S+pow(CLht2,2.0)/
      (pi*AR*e);
CDvt=2.0*Kht*pow(10.0,(-0.21*log10(RNvt)-0.97))*Svt/S;
CDvt2=2.0*Kht2*pow(10.0,(-0.21*log10(RNvt)-0.97))*Svt/S;
```

```
   rcl=CLac+CLht*Netaht*Sht/S+CLht2*Netaht2*Sht2/S;
   //Total Cd for the Super Kite
   rcd=CDfs+CDac+CDht*Netaht+CDht2*Netaht2+CDvt*Netavt+CDvt2*Netavt2;

   //
   //Calculate the actual attainable altitude for the given trim AOA
   //
   //Net vertical force
   Ty=CLac*q*S+CLht*q*Netaht*Sht+CLht2*q*Netaht2*Sht2-W;
   //Net horizontal force
   Tx=(CDfs+CDac+CDht*Netaht+CDvt*Netavt+CDht2*Netaht2+CDvt2*Netavt2)*q*S;
   tmperl=tdens*pi*pow(tdia/2.0,2.0);     //Mass per unit length of the tether
   xacc=10e-10;                           //Solver accuracy
   x1=-15.0;                              //Lower angle limit
   x2=90.0;                               //Upper angle limit
   ttlen=0.0;                             //initialize the total tether length
   rtb=90.0;
   i=0;
   while(Ty>0.0 && rtb>=tGammallim)
   {
      if(AOA==-99.0) Ty=-99.0;
      else
      {
         j=1;
         i++;
         ttlen+=teleml;
         f=tangle(x1);
         rtb = f < 0.0 ? (dx=x2-x1,x1) : (dx=x1-x2,x2);
         while(j!=0)
         {
            fmid=tangle(xmid=rtb+(dx *=0.5));
            if(fmid<=0.0) rtb=xmid;
            if(fabs(dx)<xacc || fmid == 0.0) j=0;
         }
         Tx=Tx+tCDax*q*tdia*teleml*pow(cos(rtb*pi/180.0),3.0)+tCDcyl*q*tdia*
            teleml*pow(sin(rtb*pi/180.0),3.0);
         Ty=Ty+tCDax*q*tdia*teleml*pow(cos(rtb*pi/180.0),2.0)*sin(rtb*pi/180.0)
            -tCDcyl*q*tdia*teleml*pow(sin(rtb*pi/180.0),2.0)*cos(rtb*pi/180.0)-
            tmperl*teleml*g;
         Alt+=teleml*sin(rtb*pi/180.0);
      }
   }
   if(i>1)
   {
      ttlen=ttlen-teleml;                  //Save the total tether length
      ralt=Alt-teleml*sin(rtb*pi/180.0);   //Save the last calculated Altitude
   }
   rsalt=Alttd;                            //Save the last specified Altitude
   return (Alttd = ralt);
}

void
KiteData::KiteTetherProfile(void)
{
   int j;
   double x1,x2,f,fmid,dx,xmid,rtb,xacc;

   N=0;                                         //Number of elements
   Profile->Flush();                            //Empty the profile array

   //
   //Calculate the actual attainable altitude for the given trim AOA
   //
   //Net vertical force
   Ty=CLac*q*S+CLht*q*Netaht*Sht+CLht2*q*Netaht2*Sht2-W;
   Tx=(CDfs+CDac+CDht*Netaht+CDvt*Netavt+CDht2*Netaht2+CDvt2*Netavt2)*q*S;
   Tbar=pow(pow(Tx,2.0)+pow(Ty,2.0),0.5);       //SS Net Tension at the CP
   Gammabar=atan(Ty/Tx);                        //SS Net Tension angle (radians)
   tmperl=tdens*pi*pow(tdia/2.0,2.0);           //Mass/unit lgth of the tether
   xacc=10e-10;                                 //Solver accuracy
   x1=-15.0;                                    //Lower angle limit
   x2=90.0;                                     //Upper angle limit
   rtb=90.0;
   while(Ty>0.0 && rtb>=tGammallim)
   {
      if(AOA==-99.0) Ty=-99.0;
      else
      {
         j=1;
         f=tangle(x1);
         rtb = f < 0.0 ? (dx=x2-x1,x1) : (dx=x1-x2,x2);
         while(j!=0)
         {
            fmid=tangle(xmid=rtb+(dx *=0.5));
            if(fmid<=0.0) rtb=xmid;
            if(fabs(dx)<xacc || fmid == 0.0) j=0;
         }
         Tx=Tx+tCDax*q*tdia*teleml*pow(cos(rtb*pi/180.0),3.0)+tCDcyl*q*
            tdia*teleml*pow(sin(rtb*pi/180.0),3.0);
         Ty=Ty+tCDax*q*tdia*teleml*pow(cos(rtb*pi/180.0),2.0)*sin(rtb*pi/180.0)
            -tCDcyl*q*tdia*teleml*pow(sin(rtb*pi/180.0),2.0)*cos(rtb*pi/180.0)-
            tmperl*teleml*g;
         Profile->Add(rtb*pi/180.0);
         N++;
      }
   }
   if(N>1)
      N--;
}

double
KiteData::WindFunc(double alt)
{
   return (WFA[3]+alt*(WFA[4]+alt*(WFA[5]+alt*(WFA[6]+alt*(WFA[7]+alt*
         (WFA[8]+alt*(WFA[9]+alt*(WFA[10]+alt*(WFA[11]+alt*WFA[12])))))))));
}

void
KiteData::CalculateNonDims(AltitudeData& AltData)
{
   //Create temporary Altitude object and init. with current kite altitude
   Altitude A(Alttd);

   //Get the data assoc. with the altitude from the datafile
   AltData.GetData(A, AltData);

   //General
   Temp=200.0;           //Approx. Temperature at altitude (deg K)
   dEpsilondq=0.001;     //Change in downwash with change in pitch rate
   Nfg=0.9;              //Fudge factor for Czq calculation
   Nls=1.0;              //ratio of velocity at ls with freestream velocity
   Nlsl=0.95;            //ratio of lateral vel. at ls with freestream lat. vel.
   //Coeff. of lift of the wing at ref. AOA
   CLo = awing*(iwing+AOA+azll)*pi/180.0;
   CNbCL=0.01;           //CNbeta/pow(Coeff of lift,2)  ->  Etkin or NACA 1098
                         //could refine with an equation in future
   CLppf=-0.58;          //(CLp)planform  ->  Etkin or NACA 1098 (pg 20)
   Cnp1=-0.08;           // ->  Etkin or NACA 1098
   Cnp2=8.5;             // ->  Etkin or NACA 1098
   CDpls=0.011;          //Change in CDo for ls wrt AOA (/degree!!!)
   CLrCLo=0.28;          // -> Etkin and Reid pg 351
```

```
Cmi2=-0.3;            // -> Etkin or NACA 1098
Mach=A.Wind/pow((1.4*287.05*Temp),0.5);            //Mach number
//Coefficient of Vertical tether force at the cp.
   Czcpo=(Tbar*sin(Gammabar))/(0.5*A.Dens*pow(A.Wind,2.0)*S);
//Coefficient of Horizontal tether force at the cp.
   Cxcpo=(Tbar*cos(Gammabar))/(0.5*A.Dens*pow(A.Wind,2.0)*S);
//Coefficient of tether Moment
   Cmcpo=(Tbar*cos(Gammabar)*(Zcg-Zcplg)+Tbar*sin(Gammabar)*(Xcplg-Xcg))
             /(0.5*A.Dens*pow(A.Wind,2.0)*S*cmean);

//Calculate the Longitudinal non-dimensional stability derivatives
//U derivatives
//    X
Cxu=0.0;

//    Z
Czu=0.0;

//    M
Cmu=0.0;

//Alpha Derivatives
//    X
Cxalpha=rcl-2.0*CLo*awing/(pi*AR*e);

//    Z
Czalpha=-awing-rcd;

//    M
Cmalpha=Cxalpha*(Zcg-Znp)/cmean-Czalpha*(Xcg-Xnp)/cmean;

//q derivatives
//    X
Cxalphals=(CLo-CDpls*180.0/pi);
Cxqlsac=Cxalphals;
Cxqls=-Cxalphals*(2.0*(Xcg-Xac)/cmean+dEpsilondq)*Nfg+Cxqlsac*Nfg;

//    Z
Czalphals=-(awing+CDac);
Czqlsac=Czalphals;
Czqls=-Czalphals*(2.0*(Xcg-Xac)/cmean+dEpsilondq)*Nfg+Czqlsac*Nfg;
Czqt=-2.0*(aht*VH+aht2*VH2);
Czq=Czqls+Czqt;

//    M
Cmqlsac=-pi/4.0*cos(QCswp*pi/180.0)*(pow((AR*tan(QCswp*pi/180.0)),3.0)/3.0
   /(AR+6.0*cos(QCswp*pi/180.0))+1.0);
Cmqls=Cxqls*(Zcg-Zac)/cmean-Czqls*(Xcg-Xac)/cmean+Cmqlsac;
Cmqt=-2.0/cmean*(aht*VH*lht+aht2*VH2*lht2);
Cmq=Cmqls+Cmqt;

//alpha_dot derivatives
//    Z
Czadotbody=-2.0*0.75*Volfs/S/cmean;
Czadotls=0.0;
Czadott=-2.0*aht*VH*dEpsilonda;
Czadott2=-2.0*aht2*VH2*dEpsilonda2;
Czadot=Czadotbody+Czadotls+Czadott+Czadott2;

//    M
Cmadotbody=-(Xcg-Xcv)/cmean*Czadotbody;
Cmadotls=-(Xcg-Xac)/cmean*Czadotls;
Cmadott=-lht*Czadott-lht2*Czadott2;
Cmadot=Cmadotbody+Cmadotls+Cmadott;

//Calculate the Lateral non-dimensional stability derivatives
```

```
//Beta derivatives
//    Y
Cybfs=-(avt+CDvt)*Svt/S;
Cybf=Netavt*(1.0-dSigmadB)*Cybfs;
Cybfs2=-(avt2+CDvt2)*Svt2/S;
Cybf2=Netavt2*(1.0-dSigmadB)*Cybfs2;
Cybbody=-Ki*afs*pow(Volfs,(2.0/3.0))/S;
Cybls=-0.75*awing*pow(sin(Dih*pi/180.0),2.0);
Cyb=Cybf+Cybf2+Cybbody+Cybls;

//    L
CLbf=-(Zcg-Zvt)/b*Cybf;
CLbf2=-(Zcg-Zvt2)/b*Cybf2;
CLbls=-0.25*Dih*pi/180.0*CLo;
CLbbody=-(Zcg-Zfscp)/b*Cybbody;
CLb=CLbf+CLbf2+CLbls+CLbbody;

//    N
CNbbody=-0.96*KB*Sfs/S*lfs/b*pow(fsqh/fstqh,0.5)*pow(fstqw/fsqw,1.0/3.0);
CNbf=lvt/b*Cybf;
CNbf2=lvt2/b*Cybf2;
CNblsac=CNbCL*pow(CLo,2.0);
CNbls=(Xcg-Xac)/b*Cybls+CNblsac;
CNb=CNbbody+CNbf+CNbf2+CNbls;

//P derivatives

//    Y
Cypbody=-2.0*(Zcg-Zfscp)/b*Cybbody;
Cypf=Cybfs*(-2.2*(Zcg-Zvt)/b+dSigfdP)*Netavt;
Cypf2=Cybfs2*(-2.2*(Zcg-Zvt2)/b+dSigfdP)*Netavt2;
Cyplsac=((AR+cos(QCswp*pi/180.0))/(AR+4.0*cos(QCswp*pi/180.0))*
            tan(QCswp*pi/180.0)+1.0/AR)*CLo;
Cypls=-Cybls*(-2.0*(Zcg-Zac)/b+dSiglsdP)*Nls+Cyplsac;
Cyp=Cypbody+Cypf+Cypf2+Cypls;

//    L
CLpbody=-(Zcg-Zfscp)/b*Cypbody;
CLpf=-1.1*(Zcg-Zvt)/b*Cypf;
CLpf2=-1.1*(Zcg-Zvt2)/b*Cypf2;
CLplsac=CLppf;
CLpls=-(Zcg-Zac)/b*Cypls+CLplsac;
CLp=CLpbody+CLpf+CLpf2+CLpls;

//    N
CNpbody=2.0*(Zcg-Zfscp)/b*CNbbody;
CNpf=lvt/b*Cypf;
CNpf2=lvt2/b*Cypf2;
CNplsac=(Cnp1*CLo+Cnp2*CDpls);
CNpls=(Xcg-Xac)/b*Cypls+CNplsac;
CNp=CNpbody+CNpf+CNpf2+CNpls;

//R derivatives

//    Y
Cyrfac=-Cybf*cvt/b;
Cyrf=Cybfs*(2.0*lvt/b+dSigfdr)*Netavt+Cyrfac;
Cyrfac2=-Cybf2*cvt2/b;
Cyrf2=Cybfs2*(2.0*lvt2/b+dSigfdr)*Netavt2+Cyrfac2;
Cyrls=Cybls*(2.0*(Xcg-Xac)/b+dSiglsdr)*Nlsl;
Cyr=Cyrf+Cyrf2+Cyrls;

//    L
CLrf=-(Zcg-Zvt)/b*Cyrf;
CLrf2=-(Zcg-Zvt2)/b*Cyrf2;
cswp=cos(QCswp*pi/180.0);
```

```
   CLrCL=(1.0+AR*(1.0-pow(B,2.0))/(2.0*B*(AR*B+2.0*cswp))+(AR*B+2.0*cswp)
      /(AR*B+4.0*cswp)*pow(tan(QCswp*pi/180.0),2.0)/8.0)/(1.0+(AR+2.0*cswp)
      /(AR+4.0*cswp)*pow(tan(QCswp*pi/180.0),2.0)/8.0)*CLrCLo;
   CLrlsac=CLrCL*CLo;
   CLrls=-(Zcg-Zac)/b*Cyrls+CLrlsac;
   CLr=CLrf+CLrf2+CLrls;

   //    N
   CNrfac=-pi/4.0*pow(cvt/b,2.0)*Svt/S;
   CNrf=lvt/b*Cyrf+CNrfac;
   CNrfac2=-pi/4.0*pow(cvt2/b,2.0)*Svt2/S;
   CNrf2=lvt2/b*Cyrf2+CNrfac2;
   CNrlsac=(Cnr1*pow(CLo,2.0)+Cnr2*CDpls);
   CNrls=(Xcg-Xac)/b*Cyrls+CNrlsac;
   CNr=CNrf+CNrf2+CNrls;
}

int
KiteData::iswindfile(void)
{
   return IWF;
}

char*
KiteData::FileForm(void)
{
   return FileName_Formula;
}

void
KiteData::FileForm(char* FF)
{
   strcpy(FileName_Formula,FF);
}

ostream&
operator <<(ostream& os, KiteData& KD)
{
   _fpreset();
   os << string(KD.FileName_Formula) << "\n";
   os << KD.IWF << "\n";
   os << KD.AOA << "\n";
   os << KD.Alttd << "\n";
   os << KD.mass << "\n";
   os << KD.e << "\n";
   os << KD.Xcg << "\n";
   os << KD.Zcg << "\n";
   os << KD.Xcplg << "\n";
   os << KD.Zcplg << "\n";
   os << KD.b << "\n";
   os << KD.croot << "\n";
   os << KD.ctip << "\n";
   os << KD.awing << "\n";
   os << KD.Cmac << "\n";
   os << KD.Thick << "\n";
   os << KD.Dih << "\n";
   os << KD.LEswp << "\n";
   os << KD.QCswp << "\n";
   os << KD.iwing << "\n";
   os << KD.azll << "\n";
   os << KD.Xac << "\n";
   os << KD.Zac << "\n";
   os << KD.hacwb << "\n";
   os << KD.Eo << "\n";
   os << KD.dSiglsdP << "\n";
   os << KD.CLp_plan << "\n";
os << KD.cht << "\n";
os << KD.aht << "\n";
os << KD.Thickht << "\n";
os << KD.Xht << "\n";
os << KD.Zht << "\n";
os << KD.iht << "\n";
os << KD.azllht << "\n";
os << KD.Netaht << "\n";
os << KD.bvt << "\n";
os << KD.cvt << "\n";
os << KD.avt << "\n";
os << KD.Thickvt << "\n";
os << KD.Xvt << "\n";
os << KD.Zvt << "\n";
os << KD.Netavt << "\n";
os << KD.dSigfdP << "\n";
os << KD.dSigfdr << "\n";
os << KD.dSigmadB << "\n";
os << KD.Volfs << "\n";
os << KD.fsw << "\n";
os << KD.fsh << "\n";
os << KD.fsqw << "\n";
os << KD.fsqh << "\n";
os << KD.fstqw << "\n";
os << KD.fstqh << "\n";
os << KD.cfs << "\n";
os << KD.lfs << "\n";
os << KD.sigfs_B << "\n";
os << KD.afs << "\n";
os << KD.Zfscp << "\n";
os << KD.tdens << "\n";
os << KD.tdia << "\n";
os << KD.teleml << "\n";
os << KD.tCDcyl << "\n";
os << KD.tCDax << "\n";
os << KD.Xcv << "\n";
os << KD.Ixz << "\n";
os << KD.Ixx << "\n";
os << KD.Iyy << "\n";
os << KD.Izz << "\n";
os << KD.Nc << "\n";
os << KD.tdiao << "\n";
os << KD.tethE << "\n";
os << KD.Clmax << "\n";
os << KD.tGammallim << "\n";
os << KD.aoalim << "\n";
os << KD.aoaset << "\n";
os << KD.aoasetomax << "\n";
os << KD.Xcplt << "\n";
os << KD.Zcplt << "\n";
os << KD.bht2 << "\n";
os << KD.cht2 << "\n";
os << KD.aht2 << "\n";
os << KD.Thickht2 << "\n";
os << KD.Xht2 << "\n";
os << KD.Zht2 << "\n";
os << KD.iht2 << "\n";
os << KD.azllht2 << "\n";
os << KD.Netaht2 << "\n";
os << KD.bvt2 << "\n";
os << KD.cvt2 << "\n";
os << KD.avt2 << "\n";
os << KD.Thickvt2 << "\n";
os << KD.Xvt2 << "\n";
os << KD.Zvt2 << "\n";
os << KD.CGy << "\n";
os << KD.CGz << "\n";
os << KD.CGl << "\n";
os << KD.CGm << "\n";
os << KD.CGn << "\n";

int i,j;
if(!KD.IWF)
{
   for(i=0;i<13;i++)
      os << KD.WFA[i] << "\n";
}

os << "\n\nProfile data - element angles (rads) from kite to grnd\n";
TViter iterProf(*KD.Profile);
iterProf.Restart();
int upperlimit = KD.Profile->GetItemsInContainer()-2;
for(i=0;i<=upperlimit;i++)
{
   os << iterProf.Current() << "\n";
   iterProf++;
}


int N=KD.N;
double checkmax,checkmin;
complex checksum;
if(N<=1)
   N=0;
if(KD.EiglgValid)
{
   os << "\n\nEigen values and vectors - Longitudinal Analysis\n\n\n";
   os << "Eigen values (real on top, imag. on bottom)\n";
   for(i=0;i<N*4+6;i++)
   {
      if(KD.Lgalfi[i]>=0.0)
         os << KD.Lgalfr[i] << "\t";
      if(KD.Lgalfi[i]>0.0)
         os << "\t";
   }
   os << "\n";
   for(i=0;i<N*4+6;i++)
   {
      if(KD.Lgalfi[i]>=0.0)
         os << KD.Lgalfi[i] << "\t";
      if(KD.Lgalfi[i]>0.0)
         os << "\t";
   }
   os << "\n\nEigen vectors -- Body:\n";

   for(i=1;i<=4;i++)
   {
      for(j=5;j<=N*4+6;j++)
      {
         os << KD.Lgz(i,j) << "\t";
      }
      os << "\n";
   }
   os << "\nElement stretch:\n";
   for(i=N*2+7;i<=N*4+6;i+=2)
   {
      for(j=5;j<=N*4+6;j++)
      {
         os << KD.Lgz(i,j) << "\t";
      }
}
os << "\nChange in long. element angle:\n";
for(i=N*2+8;i<=N*4+6;i+=2)
{
   for(j=5;j<=N*4+6;j++)
   {
      os << KD.Lgz(i,j) << "\t";
   }
   os << "\n";
}

//*****************************************************************
//Print the normalized mode shapes as calculated
//Stretch Only
os << "\n\n ***********************************************************";
os << "****************************";
os << "\n Mode shapes normalized wrt total motion at confluence point";
os << " due to element stretch ONLY\n\n";
for(i=1;i<=4;i++)
{
   for(j=1;j<=KD.Lgmodestretch.n;j++)
   {
      os << real(KD.Lgmodestretch(i,j)) << "\t";
      if(fabs(imag(KD.Lgmodestretch(1,j)))>1.0e-10)
         os << imag(KD.Lgmodestretch(i,j)) << "\t";
   }
   os << "\n";
}
os << "\nTether motion (from grnd up) - X direction:\n";
for(i=1;i<=N;i++)
{
   for(j=1;j<=KD.Lgmodestretch.n;j++)
   {
      os << real(KD.Lgmodestretch(3+2*i,j)) << "\t";
      if(fabs(imag(KD.Lgmodestretch(1,j)))>1.0e-10)
         os << imag(KD.Lgmodestretch(3+2*i,j)) << "\t";
   }
   os << "\n";
}
os << "\nTether motion (from grnd up) - Z direction:\n";
for(i=1;i<=N;i++)
{
   for(j=1;j<=KD.Lgmodestretch.n;j++)
   {
      os << real(KD.Lgmodestretch(4+2*i,j)) << "\t";
      if(fabs(imag(KD.Lgmodestretch(1,j)))>1.0e-10)
         os << imag(KD.Lgmodestretch(4+2*i,j)) << "\t";
   }
   os << "\n";
}
os << "\nTether motion (from grnd up) - Total Magnitude:\n";
for(i=1;i<=N;i++)
{
   for(j=1;j<=KD.Lgmodestretch.n;j++)
   {
      os << real(pow(pow(KD.Lgmodestretch(3+2*i,j),2.0)+pow(
         KD.Lgmodestretch(4+2*i,j),2.0),0.5)) << "\t";
      if(fabs(imag(KD.Lgmodestretch(1,j)))>1.0e-10)
         os << imag(pow(pow(KD.Lgmodestretch(3+2*i,j),2.0)+
            pow(KD.Lgmodestretch(4+2*i,j),2.0),0.5)) << "\t";
   }
   os << "\n";
}

//*****************************************************************
//Print the normalized mode shapes as calculated
```

```
os << "\n\n *************************************************************";
os << "***********************************";
os << "\n Mode shapes normalized wrt total motion at confluence point";
os << " due to element angle change ONLY\n\n";
for(i=1;i<=4;i++)
{
   for(j=1;j<=KD.Lgmodeangle.n;j++)
   {
      os << real(KD.Lgmodeangle(i,j)) << "\t";
      if(fabs(imag(KD.Lgmodeangle(1,j)))>1.0e-10)
         os << imag(KD.Lgmodeangle(i,j)) << "\t";
   }
   os << "\n";
}
os << "\nTether motion (from grnd up) - X direction:\n";
for(i=1;i<=N;i++)
{
   for(j=1;j<=KD.Lgmodeangle.n;j++)
   {
      os << real(KD.Lgmodeangle(3+2*i,j)) << "\t";
      if(fabs(imag(KD.Lgmodeangle(1,j)))>1.0e-10)
         os << imag(KD.Lgmodeangle(3+2*i,j)) << "\t";
   }
   os << "\n";
}
os << "\nTether motion (from grnd up) - Z direction:\n";
for(i=1;i<=N;i++)
{
   for(j=1;j<=KD.Lgmodeangle.n;j++)
   {
      os << real(KD.Lgmodeangle(4+2*i,j)) << "\t";
      if(fabs(imag(KD.Lgmodeangle(1,j)))>1.0e-10)
         os << imag(KD.Lgmodeangle(4+2*i,j)) << "\t";
   }
   os << "\n";
}
os << "\nTether motion (from grnd up) - Total Magnitude:\n";
for(i=1;i<=N;i++)
{
   for(j=1;j<=KD.Lgmodeangle.n;j++)
   {
      os << real(pow(pow(KD.Lgmodeangle(3+2*i,j),2.0)+pow(
         KD.Lgmodeangle(4+2*i,j),2.0),0.5)) << "\t";
      if(fabs(imag(KD.Lgmodeangle(1,j)))>1.0e-10)
         os << imag(pow(pow(KD.Lgmodeangle(3+2*i,j),2.0)+pow(
            KD.Lgmodeangle(4+2*i,j),2.0),0.5)) << "\t";
   }
   os << "\n";
}

//********************************************************************
//Print the normalized mode shapes as calculated
//Both stretch and angle change
os << "\n\n *************************************************************";
os << "***********************************************";
os << "\n Mode shapes normalized wrt total motion at confluence point";
os << " due to both element stretch and angle change\n\n";
for(i=1;i<=4;i++)
{
   for(j=1;j<=KD.Lgmodes.n;j++)
   {
      os << real(KD.Lgmodes(i,j)) << "\t";
      if(fabs(imag(KD.Lgmodes(1,j)))>1.0e-10)
         os << imag(KD.Lgmodes(i,j)) << "\t";
   }
   os << "\n";
```

```
for(i=1;i<=N;i++)
{
   for(j=1;j<=KD.Lgmodes.n;j++)
   {
      os << real(KD.Lgmodes(3+2*i,j)) << "\t";
      if(fabs(imag(KD.Lgmodes(1,j)))>1.0e-10)
         os << imag(KD.Lgmodes(3+2*i,j)) << "\t";
   }
   os << "\n";
}
os << "\nTether motion (from grnd up) - Z direction:\n";
for(i=1;i<=N;i++)
{
   for(j=1;j<=KD.Lgmodes.n;j++)
   {
      os << real(KD.Lgmodes(4+2*i,j)) << "\t";
      if(fabs(imag(KD.Lgmodes(1,j)))>1.0e-10)
         os << imag(KD.Lgmodes(4+2*i,j)) << "\t";
   }
   os << "\n";
}
os << "\nTether motion (from grnd up) - Total Magnitude:\n";
for(i=1;i<=N;i++)
{
   for(j=1;j<=KD.Lgmodes.n;j++)
   {
      os << real(pow(pow(KD.Lgmodes(3+2*i,j),2.0)+pow(KD.Lgmodes(4+2*i,j)
         ,2.0),0.5)) << "\t";
      if(fabs(imag(KD.Lgmodes(1,j)))>1.0e-10)
         os << imag(pow(pow(KD.Lgmodes(3+2*i,j),2.0)+pow(KD.Lgmodes
            (4+2*i,j),2.0),0.5)) << "\t";
   }
   os << "\n";
}

if(real(KD.Lgcheck[0])!=100.0)
{
   checkmax=checkmin=abs(KD.Lgcheck[5]);
   checksum=complex(0.0,0.0);
   os << "\n\nResult matrix of eigen check - Longitudinal Analysis\n\n";
   for(i=0;i<N*4+6;i++)
   {
      for(j=0;j<N*4+6;j++)
      {
         if(abs(KD.Lgcheck[i*(N*4+6)+j])>1.0e-10)
            os << KD.Lgcheck[i*(N*4+6)+j] << "\t";
         else
            os << "(0.0,0.0)\t";
         if(real(KD.Lgcheck[i*(N*4+6)+j])!=0.0&&
                                     imag(KD.Lgcheck[i*(N*4+6)+j])!=-1.0)
         {
            checksum+=KD.Lgcheck[i*(N*4+6)+j];
            if(abs(KD.Lgcheck[i*(N*4+6)+j])>checkmax)
               checkmax=abs(KD.Lgcheck[i*(N*4+6)+j]);
            if(abs(KD.Lgcheck[i*(N*4+6)+j])<checkmin)
               checkmin=abs(KD.Lgcheck[i*(N*4+6)+j]);
         }
      }
      os << "\n";
   }
   os << "\nThe average of all of the elements in the above matrix is:\n";
   os << checksum/(16.0*N*(N+2.0)+12.0) << "\n";
   os << "\nThe max and min of all of the elements in the above";
   os << " matrix is:\n";
   os << checkmax << "\t" << checkmin << "\n";
```

```
}

if(KD.EigltValid)
{
   os << "\n\n\nEigen values and vectors - Lateral Analysis\n\n\n";
   os << "Eigen values (real on top, imag. on bottom)\n";
   for(i=0;i<N*2+6;i++)
   {
      if(KD.Ltalfi[i]>=0.0)
         os << KD.Ltalfr[i] << "\t";
      if(KD.Ltalfi[i]>0.0)
         os << "\t";
   }
   os << "\n";
   for(i=0;i<N*2+6;i++)
   {
      if(KD.Ltalfi[i]>=0.0)
         os << KD.Ltalfi[i] << "\t";
      if(KD.Ltalfi[i]>0.0)
         os << "\t";
   }
   os << "\n\nEigen vectors -- Body:\n";

   for(i=1;i<=5;i++)
   {
      for(j=3;j<=N*2+6;j++)
      {
         os << KD.Ltz(i,j) << "\t";
      }
      os << "\n";
   }
   os << "\nChange in lateral element angle:\n";
   for(i=N+7;i<=N*2+6;i++)
   {
      for(j=3;j<=N*2+6;j++)
      {
         os << KD.Ltz(i,j) << "\t";
      }
      os << "\n";
   }

   //Print the mode shapes as calculated
   os << "\n\n Mode shapes normalized wrt motion at confluence point\n\n";
   for(i=1;i<=5;i++)
   {
      for(j=1;j<=KD.Ltmodes.n;j++)
      {
         os << real(KD.Ltmodes(i,j)) << "\t";
         if(fabs(imag(KD.Ltmodes(1,j)))>1.0e-10)
            os << imag(KD.Ltmodes(i,j)) << "\t";
      }
      os << "\n";
   }
   os << "\nTether motion (from grnd up) - Y direction:\n";
   for(i=1;i<=N;i++)
   {
      for(j=1;j<=KD.Ltmodes.n;j++)
      {
         os << real(KD.Ltmodes(5+i,j)) << "\t";
         if(fabs(imag(KD.Ltmodes(1,j)))>1.0e-10)
            os << imag(KD.Ltmodes(5+i,j)) << "\t";
      }
      os << "\n";
   }

   if(real(KD.Ltcheck[0])!=100.0)
```

```
         checkmax=checkmin=abs(KD.Ltcheck[5]);
         checksum=complex(0.0,0.0);
         os << "\n\nResult matrix of eigen check - Lateral Analysis\n\n";
         for(i=0;i<N*2+6;i++)
         {
            for(j=0;j<N*2+6;j++)
            {
               if(abs(KD.Ltcheck[i*(N*2+6)+j])>1.0e-10)
                  os << KD.Ltcheck[i*(N*2+6)+j] << "\t";
               else
                  os << "(0.0,0.0)\t";
               if(real(KD.Ltcheck[i*(N*2+6)+j])!=0.0&&
                                          imag(KD.Ltcheck[i*(N*2+6)+j])!=-1.0)
               {
                  checksum+=KD.Ltcheck[i*(N*2+6)+j];
                  if(abs(KD.Ltcheck[i*(N*2+6)+j])>checkmax)
                     checkmax=abs(KD.Ltcheck[i*(N*2+6)+j]);
                  if(abs(KD.Ltcheck[i*(N*2+6)+j])<checkmin)
                     checkmin=abs(KD.Ltcheck[i*(N*2+6)+j]);
               }
            }
            os << "\n";
         }
         os << "\nThe average of all the elements in the above matrix is:\n";
         os << checksum/(4.0*N*(N+4.0)+12.0) << "\n";
         os << "\nThe max and min of all the elements in the above";
         os << " matrix is:\n";
         os << checkmax << "\t" << checkmin << "\n";
      }
   }

   // return the stream object
   return os;
}

istream&
operator >>(istream& is, KiteData& KD)
{
   _fpreset();
   char temp[150];

   is.getline(temp, sizeof(temp));
   strcpy(KD.FileName_Formula,temp);
   is >> KD.IWF;
   is >> KD.AOA;
   is >> KD.Alttd;
   is >> KD.mass;
   is >> KD.e;
   is >> KD.Xcg;
   is >> KD.Zcg;
   is >> KD.Xcplg;
   is >> KD.Zcplg;
   is >> KD.b;
   is >> KD.croot;
   is >> KD.ctip;
   is >> KD.awing;
   is >> KD.Cmac;
   is >> KD.Thick;
   is >> KD.Dih;
   is >> KD.LEswp;
   is >> KD.QCswp;
   is >> KD.iwing;
   is >> KD.azll;
   is >> KD.Xac;
   is >> KD.Zac;
   is >> KD.hacwb;
is >> KD.dSiglsdP;
is >> KD.CLp_plan;
is >> KD.dSiglsdr;
is >> KD.bht;
is >> KD.cht;
is >> KD.aht;
is >> KD.Thickht;
is >> KD.Xht;
is >> KD.Zht;
is >> KD.iht;
is >> KD.azllht;
is >> KD.Netaht;
is >> KD.bvt;
is >> KD.cvt;
is >> KD.avt;
is >> KD.Thickvt;
is >> KD.Xvt;
is >> KD.Zvt;
is >> KD.Netavt;
is >> KD.dSigfdP;
is >> KD.dSigfdr;
is >> KD.dSigmadB;
is >> KD.Volfs;
is >> KD.fsw;
is >> KD.fsh;
is >> KD.fsqw;
is >> KD.fsqh;
is >> KD.fstqw;
is >> KD.fstqh;
is >> KD.cfs;
is >> KD.lfs;
is >> KD.sigfs_B;
is >> KD.afs;
is >> KD.Zfscp;
is >> KD.tdens;
is >> KD.tdia;
is >> KD.teleml;
is >> KD.tCDcyl;
is >> KD.tCDax;
is >> KD.Xcv;
is >> KD.Ixz;
is >> KD.Ixx;
is >> KD.Iyy;
is >> KD.Izz;
is >> KD.Nc;
is >> KD.tdiao;
is >> KD.tethE;
is >> KD.Clmax;
is >> KD.tGammallim;
is >> KD.aoalim;
is >> KD.aoaset;
is >> KD.aoasetomax;
is >> KD.Xcplt;
is >> KD.Zcplt;
is >> KD.bht2;
is >> KD.cht2;
is >> KD.aht2;
is >> KD.Thickht2;
is >> KD.Xht2;
is >> KD.Zht2;
is >> KD.iht2;
is >> KD.azllht2;
is >> KD.Netaht2;
is >> KD.bvt2;
is >> KD.cvt2;
is >> KD.avt2;
```

```
   is >> KD.Xvt2;
   is >> KD.Zvt2;
   is >> KD.Netavt2;
   is >> KD.CGx;
   is >> KD.CGy;
   is >> KD.CGz;
   is >> KD.CGl;
   is >> KD.CGm;
   is >> KD.CGn;

   int i;
   if(!KD.IWF)
   {
      for(i=0;i<13;i++)
         is >> KD.WFA[i];
   }

  // return the stream object
  return is;
}

Scale::Scale()
{
   YMaxSet=XMaxSet=XMinSet=false;
   YMax=XMax=XMin=0.0;
}

void
Scale::SetScale(double& ymax, double& xmax, double& xmin)
{
   if(YMaxSet&&YMax<ymax)
      ymax=YMax-YMax/1000.0;
   if(XMaxSet&&XMax<xmax)
      xmax=XMax-XMax/1000.0;
   if(XMinSet&&XMin>xmin)
      xmin=XMin-XMin/1000.0;
}
```

```
//
//Super Kite Stability Roots Analysis
//

#include <stdio.h>
#include <stdlib.h>
#include <cstring.h>
#include <math.h>
#include <classlib/arrays.h>
#include <float.h>
#include <fstream.h>
#include <iomanip.h>
#include "matrix.h"
#include "rggpool.h"
#include "suprkite.h"

// Calculate the Stability Roots of a given kite-tether configuration.
// CG contains the values of the six control gains.
// AltData contains the Altitude data for a given trim-state.
// CheckEV is set TRUE to check the mathematical validity of the Eigen value
//    solution.
// MLOut is set TRUE to output the A and B matrices for a parallel Eigen value
//    check using MatLab.
// matz is set to 1 if the eigen vectors are to be calculated or 0 not to.

int
KiteData::KiteSRoots(ControlGain& CG, AltitudeData& AltData,
                      bool CheckEV, bool MLOut ,int matz)
{
   //Initialize the output file for the MatLab cross-check
   fstream ABout;

   //Create temporary Altitude object and init. with current kite altitude
   Altitude A(Alttd);

   //Get the data assoc. with the altitude from the datafile
   AltData.GetData(A, AltData);


   const double altllim=AltData.MinAlt;
   const double altulim=AltData.MaxAlt;

   //General
   Temp=200.0;          //Approx. Temperature at altitude (deg K)
   dEpsilondq=0.001;    //Change in downwash with change in pitch rate
   Nfg=0.9;             //Fudge factor for Czq calculation
   Nls=1.0;             //ratio of velocity at ls with freestream velocity
   Nlsl=0.95;           //ratio of lateral vel. at ls with freestream lat. vel.
   //Coeff. of lift of the wing at ref. AOA
   CLo = awing*(iwing+AOA+azll)*pi/180.0;
   CNbCL=0.01;          //CNbeta/pow(Coeff of lift,2)  ->  Etkin or NACA 1098
                        //could refine with an equation in future
   CLppf=-0.58;         //(CLp)planform  ->  Etkin or NACA 1098 (pg 20)
   Cnp1=-0.08;          // ->  Etkin or NACA 1098
   Cnp2=8.5;            // ->  Etkin or NACA 1098
   CDpls=0.011;         //Change in CDo for ls wrt AOA (/degree!!!)
   CLrCLo=0.28;         // -> Etkin and Reid pg 351
   Cnr1=-0.005;         // ->  Etkin or NACA 1098
   Cnr2=-0.3;           // ->  Etkin or NACA 1098
   CWo=mass*g*2.0/A.Dens/pow(A.Wind,2.0)/S;          //Coeff. of Weight
   Mach=A.Wind/pow((1.4*287.05*Temp),0.5);           //Mach number
   //Coefficient of Vertical tether force at the cp.
      Czcpo=(Tbar*sin(Gammabar))/(0.5*A.Dens*pow(A.Wind,2.0)*S);
   //Coefficient of Horizontal tether force at the cp.
      Cxcpo=(Tbar*cos(Gammabar))/(0.5*A.Dens*pow(A.Wind,2.0)*S);
   Cmcpo=(Tbar*cos(Gammabar)*(Zcg-Zcp1g)+Tbar*sin(Gammabar)*(Xcp1g-Xcg))
            /(0.5*A.Dens*pow(A.Wind,2.0)*S*cmean);

//Calculate the Longitudinal non-dimensional stability derivatives
//U derivatives
//    X
Cxu=0.0;

//    Z
Czu=0.0;

//    M
Cmu=0.0;

//Alpha Derivatives
//    X
Cxalpha=rcl-2.0*CLo*awing/(pi*AR*e);

//    Z
Czalpha=-awing-rcd;

//    M
Cmalpha=Cxalpha*(Zcg-Znp)/cmean-Czalpha*(Xcg-Xnp)/cmean;

//q derivatives
//    X
Cxalphals=(CLo-CDpls*180.0/pi);
Cxqlsac=Cxalphals;
Cxqls=-Cxalphals*(2.0*(Xcg-Xac)/cmean+dEpsilondq)*Nfg+Cxqlsac*Nfg;

//    Z
Czalphals=-(awing+CDac);
Czqlsac=Czalphals;
Czqls=-Czalphals*(2.0*(Xcg-Xac)/cmean+dEpsilondq)*Nfg+Czqlsac*Nfg;
Czqt=-2.0*(aht*VH+aht2*VH2);
Czq=Czqls+Czqt;

//    M
Cmqlsac=-pi/4.0*cos(QCswp*pi/180.0)*(pow((AR*tan(QCswp*pi/180.0)),3.0)/3.0
   /(AR+6.0*cos(QCswp*pi/180.0))+1.0);
Cmqls=Cxqls*(Zcg-Zac)/cmean-Czqls*(Xcg-Xac)/cmean+Cmqlsac;
Cmqt=-2.0/cmean*(aht*VH*lht+aht2*VH2*lht2);
Cmq=Cmqls+Cmqt;

//alpha_dot derivatives
//    Z
Czadotbody=-2.0*0.75*Volfs/S/cmean;
Czadotls=0.0;
Czadott=-2.0*aht*VH*dEpsilonda;
Czadott2=-2.0*aht2*VH2*dEpsilonda2;
Czadot=Czadotbody+Czadotls+Czadott+Czadott2;

//    M
Cmadotbody=-(Xcg-Xcv)/cmean*Czadotbody;
Cmadotls=-(Xcg-Xac)/cmean*Czadotls;
Cmadott=-lht*Czadott-lht2*Czadott2;
Cmadot=Cmadotbody+Cmadotls+Cmadott;


//Calculate the Lateral non-dimensional stability derivatives

//Beta derivatives
//    Y
Cybfs=-(avt+CDvt)*Svt/S;
Cybf=Netavt*(1.0-dSigmadB)*Cybfs;
Cybfs2=-(avt2+CDvt2)*Svt2/S;
```

```
Cybbody=-Ki*a1s*pow(Volfs,(2.0/3.0))/S;
Cybls=-0.75*awing*pow(sin(Dih*pi/180.0),2.0);
Cyb=Cybf+Cybf2+Cybbody+Cybls;

//    L
CLbf=-(Zcg-Zvt)/b*Cybf;
CLbf2=-(Zcg-Zvt2)/b*Cybf2;
CLbls=-0.25*Dih*pi/180.0*CLo;
CLbbody=-(Zcg-Zfscp)/b*Cybbody;
CLb=CLbf+CLbf2+CLbls+CLbbody;

//    N
CNbbody=-0.96*KB*Sfs/S*lfs/b*pow(fsqh/fstqh,0.5)*pow(fstqw/fsqw,1.0/3.0);
CNbf=lvt/b*Cybf;
CNbf2=lvt2/b*Cybf2;
CNblsac=CNbCL*pow(CLo,2.0);
CNbls=(Xcg-Xac)/b*Cybls+CNblsac;
CNb=CNbbody+CNbf+CNbf2+CNbls;

//P derivatives

//    Y
Cypbody=-2.0*(Zcg-Zfscp)/b*Cybbody;
Cypf=Cybfs*(-2.2*(Zcg-Zvt)/b+dSigfdP)*Netavt;
Cypf2=Cybfs2*(-2.2*(Zcg-Zvt2)/b+dSigfdP)*Netavt2;
Cyplsac=((AR+cos(QCswp*pi/180.0))/(AR+4.0*cos(QCswp*pi/180.0))*
            tan(QCswp*pi/180.0)+1.0/AR)*CLo;
Cypls=-Cybls*(-2.0*(Zcg-Zac)/b+dSiglsdP)*Nls+Cyplsac;
Cyp=Cypbody+Cypf+Cypf2+Cypls;

//    L
CLpbody=-(Zcg-Zfscp)/b*Cypbody;
CLpf=-1.1*(Zcg-Zvt)/b*Cypf;
CLpf2=-1.1*(Zcg-Zvt2)/b*Cypf2;
CLplsac=CLppf;
CLpls=-(Zcg-Zac)/b*Cypls+CLplsac;
CLp=CLpbody+CLpf+CLpf2+CLpls;

//    N
CNpbody=2.0*(Zcg-Zfscp)/b*CNbbody;
CNpf=lvt/b*Cypf;
CNpf2=lvt2/b*Cypf2;
CNplsac=(Cnp1*CLo+Cnp2*CDpls);
CNpls=(Xcg-Xac)/b*Cypls+CNplsac;
CNp=CNpbody+CNpf+CNpf2+CNpls;

//R derivatives

//    Y
Cyrfac=-Cybf*cvt/b;
Cyrf=Cybfs*(2.0*lvt/b+dSigfdr)*Netavt+Cyrfac;
Cyrfac2=-Cybf2*cvt2/b;
Cyrf2=Cybfs2*(2.0*lvt2/b+dSigfdr)*Netavt2+Cyrfac2;
Cyrls=Cybls*(2.0*(Xcg-Xac)/b+dSiglsdr)*Nlsl;
Cyr=Cyrf+Cyrf2+Cyrls;

//    L
CLrf=-(Zcg-Zvt)/b*Cyrf;
CLrf2=-(Zcg-Zvt2)/b*Cyrf2;
cswp=cos(QCswp*pi/180.0);
B=pow((1-pow(Mach*cswp,2.0)),0.5);
CLrCL=(1.0+AR*(1.0-pow(B,2.0))/(2.0*B*(AR*B+2.0*cswp))+(AR*B+2.0*cswp)
   /(AR*B+4.0*cswp)*pow(tan(QCswp*pi/180.0),2.0)/8.0)/(1.0+(AR+2.0*cswp)
   /(AR+4.0*cswp)*pow(tan(QCswp*pi/180.0),2.0)/8.0)*CLrCLo;
CLrlsac=CLrCL*CLo;
CLrls=-(Zcg-Zac)/b*Cyrls+CLrlsac;
```

```
//    N
CNrfac=-pi/4.0*pow(cvt/b,2.0)*Svt/S;
CNrf=lvt/b*Cyrf+CNrfac;
CNrfac2=-pi/4.0*pow(cvt2/b,2.0)*Svt2/S;
CNrf2=lvt2/b*Cyrf2+CNrfac2;
CNrlsac=(Cnr1*pow(CLo,2.0)+Cnr2*CDpls);
CNrls=(Xcg-Xac)/b*Cyrls+CNrlsac;
CNr=CNrf+CNrf2+CNrls;


//Calculate the Dimensional Stability derivatives

//Longitudinal
//    X
Xu=CWo*A.Dens*A.Wind*S*sin(AOA*pi/180.0)-A.Dens*A.Wind*S*Cxcpo+
   0.5*A.Dens*A.Wind*S*Cxu;
Xw=0.5*A.Dens*A.Wind*S*Cxalpha;

//    Z
Zu=-CWo*A.Dens*A.Wind*S*cos(AOA*pi/180.0)-Czcpo*A.Dens*A.Wind*S+
   0.5*A.Dens*A.Wind*S*Czu;
Zw=0.5*A.Dens*A.Wind*S*Czalpha;
Zwdot=0.25*A.Dens*S*cmean*Czadot;
Zq=0.25*A.Dens*A.Wind*cmean*S*Czq;

//    M
Mu=-A.Dens*A.Wind*cmean*S*Cmcpo+0.5*A.Dens*A.Wind*S*cmean*Cmu;
Mw=0.5*A.Dens*A.Wind*S*cmean*Cmalpha;
Mwdot=0.25*A.Dens*S*pow(cmean,2.0)*Cmadot;
Mq=0.25*A.Dens*A.Wind*pow(cmean,2.0)*S*Cmq;


//Lateral
//    Y
Yv=0.5*A.Dens*A.Wind*S*Cyb;
Yp=0.25*A.Dens*A.Wind*b*S*Cyp;
Yr=0.25*A.Dens*A.Wind*b*S*Cyr;

//    L
Lv=0.5*A.Dens*A.Wind*b*S*CLb;
Lp=0.25*A.Dens*A.Wind*pow(b,2.0)*S*CLp;
Lr=0.25*A.Dens*A.Wind*pow(b,2.0)*S*CLr;

//    N
Nv=0.5*A.Dens*A.Wind*b*S*CNb;
Np=0.25*A.Dens*A.Wind*pow(b,2.0)*S*CNp;
Nr=0.25*A.Dens*A.Wind*pow(b,2.0)*S*CNr;

//Create three iterators to step through the tether profile data
TViter iterProfi(*Profile);
TViter iterProfj(*Profile);
TViter iterProfk(*Profile);

//Populate the A and B matrices for the longitudinal and lateral equations
int i,j,k,jj,J,Nlg,Nlt,errlg,errlt;
double Gammai,Gammaj,Gammak,sum1,sum2,sum3,curralt,Iij,Iii,tethki,ZCP,XCP;

//If there is only one tether element the problem degenerates to a massless
//dragless rigid element
if(N<=1)
   N=0;

//order of the matrices
Nlg=N*4+6;
Nlt=N*2+6;
```

```
//Delete previous persistent arrays
delete[] Lgalfr;
delete[] Lgalfi;
delete[] Ltalfr;
delete[] Ltalfi;
delete[] Lgcheck;
delete[] Ltcheck;

//Matrices for the longitudinal and lateral equations
matrix lgalfr(Nlg),lgalfi(Nlg),lgbeta(Nlg),lgz(Nlg,Nlg);
matrix ltalfr(Nlt),ltalfi(Nlt),ltbeta(Nlt),ltz(Nlt,Nlt);
matrix AAlg(Nlg,Nlg),BBlg(Nlg,Nlg);
matrix AAlt(Nlt,Nlt),BBlt(Nlt,Nlt);
matrix sorterlg(Nlg),sorterlt(Nlt);

//Matrices and variables for Eigen calculation check
complex eigenval;
matrixc eigenvec,tmpc1,tmpc2,tmpvec;

//Matrices for the calculation of total motion and final mode shapes
int Nmode,modenum;
complex dcp;
matrix cosG,sinG;

//Reset the arrays to the new dimensions
Lgalfr=new double[Nlg];
Lgalfi=new double[Nlg];
Lgz.Resize(Nlg,Nlg);
Ltalfr=new double[Nlt];
Ltalfi=new double[Nlt];
Ltz.Resize(Nlt,Nlt);
Lgcheck = new complex[1];
Ltcheck = new complex[1];

if(CheckEV)
{
   delete[] Lgcheck;
   Lgcheck=new complex[Nlg*Nlg];
   delete[] Ltcheck;
   Ltcheck=new complex[Nlt*Nlt];
}
else
{
   Lgcheck[0]=complex(100.0,0);      //flag to indicate that the
                                     //check is not done.
   Ltcheck[0]=complex(100.0,0);
}
if(!Lgalfr||!Lgalfi||!Ltalfr||!Ltalfi)
   return -5;

//Vectors for the air density and tether diameter variations as a
//function of the tether element number
matrix rho,Tdia;
if(N!=0)
{
   rho(N);Tdia(N);
}else{
   rho(2);Tdia(2);
}

//
//calculate the air density and tether diameter at every element
//(starting from the confluence point)
//
iterProfi.Restart();
sum1=0.0;
```

```
{
   curralt=Alttd-teleml/2.0*sin(iterProfi.Current())-sum1;
   sum1+=teleml*sin(iterProfi.Current());
   if(curralt<altllim)
      A.Set(altllim);
   else if(curralt>altulim)
      A.Set(altulim);
   else
      A.Set(curralt);
   AltData.GetData(A, AltData);
   rho(i)=A.Dens;
   if(N<=1)
      Tdia(i)=tdia;
   else
      Tdia(i)=tdiao+(tdia-tdiao)*(i-1)/(N-1);
   iterProfi++;
}
if(N==1) N=0;

//Set A back to kite altitude
//
A.Set(Alttd);
AltData.GetData(A, AltData);

XCP=(Xcg-Xcplg)*cos(AOA*pi/180.0)+(Zcg-Zcplg)*sin(AOA*pi/180.0);
ZCP=(Zcg-Zcplg)*cos(AOA*pi/180.0)-(Xcg-Xcplg)*sin(AOA*pi/180.0);

//*********************************************
//Populate the Longitudinal Matrices
//Eq. 55a
AAlg(1,1)=Xu;
AAlg(1,2)=Xw;
AAlg(1,4)=CG.Ccx*ZCP-mass*g*cos(AOA*pi/180.0)-Tbar*sin(Gammabar);
AAlg(1,5)=cos(Gammabar);
AAlg(1,6)=-Tbar*sin(Gammabar);
iterProfi.Restart();
for(i=2*N;i>=N+1;i--)
{
   Gammai=iterProfi.Current();
   AAlg(1,i*2+5)=-cos(Gammai)*CG.Ccx;
   AAlg(1,i*2+6)=teleml*sin(Gammai)*CG.Ccx;
   iterProfi++;
}
BBlg(1,1)=mass;

//Eq. 55b
AAlg(2,1)=Zu;
AAlg(2,2)=Zw;
AAlg(2,3)=-mass*A.Wind-Zq;
AAlg(2,4)=mass*g*sin(AOA*pi/180.0)+Tbar*cos(Gammabar)-CG.Ccz*XCP;
AAlg(2,5)=sin(Gammabar);
AAlg(2,6)=Tbar*cos(Gammabar);
iterProfi.Restart();
for(i=2*N;i>=N+1;i--)
{
   Gammai=iterProfi.Current();
   AAlg(2,i*2+5)=-sin(Gammai)*CG.Ccz;
   AAlg(2,i*2+6)=-teleml*cos(Gammai)*CG.Ccz;
   iterProfi++;
}
BBlg(2,2)=mass-Zwdot;

//Eq. 55c
double Mwdp=Mwdot/(mass-Zwdot);
AAlg(3,1)=Mu+Mwdp*Zu;
AAlg(3,3)=Mq+Mwdp*(mass*A.Wind+Zq);
AAlg(3,4)=CG.Ccm-Mwdp*(mass*g*sin(AOA*pi/180.0)-Tbar*cos(Gammabar)
            +CG.Ccz*XCP)-Tbar*(XCP*cos(Gammabar)+ZCP*sin(Gammabar));
AAlg(3,5)=ZCP*cos(Gammabar)-XCP*sin(Gammabar)+Mwdp*sin(Gammabar);
AAlg(3,6)=-Tbar*(XCP*cos(Gammabar)+ZCP*sin(Gammabar)-Mwdp*cos(Gammabar));
iterProfi.Restart();
for(i=2*N;i>=N+1;i--)
{
   Gammai=iterProfi.Current();
   AAlg(3,i*2+5)=-Mwdp*sin(Gammai)*CG.Ccz;
   AAlg(3,i*2+6)=-Mwdp*teleml*cos(Gammai)*CG.Ccz;
   iterProfi++;
}
BBlg(3,3)=Iyy;

//Eq. 38d
AAlg(4,3)=1.0;
BBlg(4,4)=1.0;

//Eq. 42a
AAlg(5,1)=1.0;
AAlg(5,3)=ZCP;
iterProfi.Restart();
for(i=N;i>=1;i--)
{
   Gammai=iterProfi.Current();
   AAlg(5,i*2+5)=cos(Gammai);
   AAlg(5,i*2+6)=-teleml*sin(Gammai);
   iterProfi++;
}

//Eq. 42c
AAlg(6,2)=1.0;
AAlg(6,3)=-XCP;
AAlg(6,4)=-A.Wind;
iterProfi.Restart();
for(i=N;i>=1;i--)
{
   Gammai=iterProfi.Current();
   AAlg(6,i*2+5)=sin(Gammai);
   AAlg(6,i*2+6)=teleml*cos(Gammai);
   iterProfi++;
}

//Pa (Eq. 26)
iterProfi.Restart();
for(i=N;i>=1;i--)
{
   Gammai=iterProfi.Current();
   AAlg(i*2+5,5)=cos(Gammabar-Gammai);
   AAlg(i*2+5,6)=-Tbar*sin(Gammabar-Gammai);
   AAlg(i*2+6,5)=teleml*sin(Gammabar-Gammai);
   AAlg(i*2+6,6)=Tbar*teleml*cos(Gammabar-Gammai);
   iterProfi++;
}

iterProfi.Restart();
for(i=N;i>=1;i--)
{
   Gammai=iterProfi.Current();
   iterProfk.Restart();
   for(k=N;k>=1;k--)
   {
      Gammak=iterProfk.Current();

      //Ma (Eq. 7)
```

```
Iij=0.0;
for(j=max(i,k);j<N;j++)
{
   Iij+=pi*pow(Tdia(j)/2.0,2.0)*teleml*tdens;
}
Iij+=pi*pow(Tdia(N)/2.0,2.0)*teleml*tdens/2.0;

BBlg(i*2+5,k*2+5)=cos(Gammai-Gammak)*Iij;
BBlg(i*2+5,k*2+6)=teleml*sin(Gammai-Gammak)*Iij;
BBlg(i*2+6,k*2+5)=-teleml*sin(Gammai-Gammak)*Iij;
BBlg(i*2+6,k*2+6)=teleml*teleml*cos(Gammai-Gammak)*Iij;



//-Fa (Eq. 25)
if(i>=k)
{
   //Note: modify in future to have the velocity vary for each segment
   //of the tether
   iterProfj.Restart();
   sum1=0.0;
   sum2=0.0;
   for(j=N;j>=i+1;j--)
   {
      Gammaj=iterProfj.Current();
      sum1+=rho(j)*Tdia(j)*sin(Gammaj)*sin(Gammai-Gammaj)*teleml
            *sin(Gammaj-Gammak);
      sum2+=rho(j)*Tdia(j)*sin(Gammaj)*sin(Gammai-Gammaj)*teleml
            *cos(Gammaj-Gammak);
      iterProfj++;
   }

   AAlg(i*2+5,k*2+5)=A.Wind*tCDcyl*sum1-Nc*delta(i,k);
   AAlg(i*2+5,k*2+6)=-A.Wind*tCDcyl*teleml*sum2;
}
if(i>k)
{
   iterProfj.Restart();
   sum1=0.0;
   sum2=0.0;
   for(j=N;j>=i;j--)
   {
      Gammaj=iterProfj.Current();
      sum1+=rho(j)*Tdia(j)*sin(Gammaj)*teleml*cos(Gammai-Gammaj)
            *sin(Gammaj-Gammak)*E(i,j);
      sum2+=rho(j)*Tdia(j)*sin(Gammaj)*teleml*cos(Gammai-Gammaj)
            *cos(Gammaj-Gammak)*E(i,j);
      iterProfj++;
   }

   AAlg(i*2+6,k*2+5)=A.Wind*tCDcyl*teleml*sum1;
   AAlg(i*2+6,k*2+6)=-A.Wind*tCDcyl*teleml*teleml*sum2;
}
if(i<=k)
{
   iterProfj.Restart();
   sum1=0.0;
   sum2=0.0;
   for(j=N;j>=k;j--)
   {
      Gammaj=iterProfj.Current();
      if(j>=k+1)
         sum1+=rho(j)*Tdia(j)*sin(Gammaj)*teleml*cos(Gammai-Gammaj)
            *sin(Gammaj-Gammak);
      sum2+=rho(j)*Tdia(j)*sin(Gammaj)*teleml*cos(Gammai-Gammaj)
            *cos(Gammaj-Gammak)*E(i,j)*E(k,j);
   }

   AAlg(i*2+6,k*2+5)=A.Wind*tCDcyl*teleml*sum1;
   AAlg(i*2+6,k*2+6)=-A.Wind*tCDcyl*teleml*teleml*sum2;
}
if(i<k)
{
   iterProfj.Restart();
   sum1=0.0;
   sum2=0.0;
   for(j=N;j>=k;j--)
   {
      Gammaj=iterProfj.Current();
      if(j>=k+1)
         sum1+=rho(j)*Tdia(j)*sin(Gammaj)*sin(Gammai-Gammaj)*teleml
            *sin(Gammaj-Gammak);
      sum2+=rho(j)*Tdia(j)*sin(Gammaj)*sin(Gammai-Gammaj)*teleml
            *cos(Gammaj-Gammak)*E(j,k);
      iterProfj++;
   }

   AAlg(i*2+5,k*2+5)=A.Wind*tCDcyl*sum1;
   AAlg(i*2+5,k*2+6)=-A.Wind*tCDcyl*teleml*sum2;
}

//Set the 2N x 2N identity matrix in AAlg and BBlg
//
if(i==k)
{
   AAlg(i*2+N*2+5,k*2+5)=1.0;
   AAlg(i*2+N*2+6,k*2+6)=1.0;
   BBlg(i*2+N*2+5,k*2+N*2+5)=1.0;
   BBlg(i*2+N*2+6,k*2+N*2+6)=1.0;
}

//Set -N - Ea in AAlg (Eq 9 & 26)
//
//Calculate the summation of mass above max(i,i)
Iii=0.0;
for(j=i;j<N;j++)
{
   Iii+=pi*pow(Tdia(j)/2.0,2.0)*teleml*tdens;
}
Iii+=pi*pow(Tdia(N)/2.0,2.0)*teleml*tdens/2.0;

//Add the -N component (Eq. 9)
//
tethki=pi*pow(Tdia(i)/2.0,2.0)*tethE/teleml;
AAlg(i*2+5,k*2+N*2+5)=-tethki*delta(i,k);
AAlg(i*2+6,k*2+N*2+6)=g*teleml*sin(Gammai)*Iii*delta(i,k);
AAlg(i*2+5,k*2+N*2+6)=AAlg(i*2+6,k*2+N*2+5)=-g*cos(Gammai)*
                      Iii*delta(i,k);

//Add the -Ea component (Eq. 26)
//
if(i==k)
{
   iterProfj.Restart();
   sum1=0.0;
   sum2=0.0;
   sum3=0.0;
   for(j=N;j>=i;j--)
   {
      Gammaj=iterProfj.Current();
      if(j>=i+1)
      {
```

```
                     cos(Gammai-Gammaj);
               sum3+=rho(j)*Tdia(j)*teleml*pow(sin(Gammaj),2.0)
                     *sin(Gammai-Gammaj);
            }
            sum2+=rho(j)*Tdia(j)*teleml*pow(sin(Gammaj),2.0)
                  *cos(Gammai-Gammaj)*E(i,j);
            iterProfj++;
         }

         AAlg(i*2+5,k*2+N*2+6)+=-0.5*tCDcyl*pow(A.Wind,2.0)*sum1+Tbar
                     *sin(Gammabar-Gammai);
         AAlg(i*2+6,k*2+N*2+5)+=-0.5*tCDcyl*pow(A.Wind,2.0)*(sum2+0.5*teleml
                     *rho(i)*Tdia(i)*pow(sin(Gammai),2.0))+Tbar
                     *sin(Gammabar-Gammai);
         AAlg(i*2+6,k*2+N*2+6)+=0.5*tCDcyl*pow(A.Wind,2.0)*teleml*(sum3-
                     teleml*rho(i)*Tdia(i)*sin(Gammai)*cos(Gammai))-Tbar
                     *teleml*cos(Gammabar-Gammai);
      }
      else if(i<k)
      {
         AAlg(i*2+5,k*2+N*2+5)+=-0.5*tCDcyl*pow(A.Wind,2.0)*rho(k)*Tdia(k)
                     *pow(sin(Gammak),2.0)*sin(Gammai-Gammak);
         AAlg(i*2+5,k*2+N*2+6)+=-0.5*tCDcyl*pow(A.Wind,2.0)*rho(k)*Tdia(k)*
                     teleml*sin(Gammak)*(2.0*cos(Gammak)*sin(Gammai-
                     Gammak)-sin(Gammak)*cos(Gammai-Gammak));
         AAlg(i*2+6,k*2+N*2+5)+=-0.5*tCDcyl*pow(A.Wind,2.0)*teleml*rho(k)*
                     Tdia(k)*pow(sin(Gammak),2.0)*cos(Gammai-Gammak);
         AAlg(i*2+6,k*2+N*2+6)+=-0.5*tCDcyl*pow(A.Wind,2.0)*teleml*rho(k)*
                     Tdia(k)*teleml*sin(Gammak)*(2.0*cos(Gammak)*cos(Gammai-
                     Gammak)+sin(Gammak)*sin(Gammai-Gammak));
      }

      iterProfk++;
   }
   iterProfi++;
}

//Copy the A and B matrices to backup matrices for post check
if(CheckEV && matz)
{
   AA.Resize(Nlg,Nlg);
   BB.Resize(Nlg,Nlg);
   eigenvec(Nlg);tmpc1(Nlg,Nlg);tmpc2(Nlg,Nlg);tmpvec(Nlg);
   AA=AAlg;
   BB=BBlg;
}

//Save the A and B matrices to MatLab script file for
// the eigen cross-check
if(MLOut)
{
   ABout.open("Lgeigchk.m",ios::out);
   ABout<< setprecision(24);
   ABout<<"AAlg = [ ";
   for(i=1;i<=Nlg;i++)
   {
      for(j=1;j<=Nlg;j++)
      {
         ABout << AAlg(i,j) << " ";
      }
      ABout << "; \n";
   }
   ABout << "]\n";

   ABout<<"BBlg = [ ";
   for(i=1;i<=Nlg;i++)
```

```
        for(j=1;j<=Nlg;j++)
        {
           ABout << BBlg(i,j) << " ";
        }
        ABout << "; \n";
     }
     ABout << "]\n\n[V,D] = eig(AAlg,BBlg)\n";
     ABout << "save lgresult.txt D V  -ascii -double -tabs\n";

     ABout.close();
  }

  //Call the eigen solver routine for the longitudinal case
  //
  errlg=rgg(AAlg,BBlg,lgalfr,lgalfi,lgbeta,matz,lgz);

  //If the function rgg returns an integer zero the function was successful
  //
  if(errlg==0)
  {
     EiglgValid=1;

     //If the eigenvalue is valid divide the real and imaginary parts
     //by the beta value
     for(i=1;i<=Nlg;i++)
     {
        sorterlg(i)=i;
        if(fabs(lgbeta(i))>1.0e-12)
        {
           lgalfr(i)=lgalfr(i)/lgbeta(i);
           lgalfi(i)=lgalfi(i)/lgbeta(i);
        }
        else
        {
           lgalfr(i)=0.0;
           lgalfi(i)=-1.0;
        }
     }

     //Check the eigen solution if required
     if(CheckEV && matz)
     {
        for(i=1;i<=Nlg;i++)
        {
           if(lgalfr(i)==0.0&&lgalfi(i)==-1.0)
           {
              for(j=1;j<=Nlg;j++)
                 Lgcheck[(j-1)*Nlg+i-1]=complex(0.0,-1.0);
           }
           else
           {
              eigenval=complex(lgalfr(i),lgalfi(i));
              if(lgalfi(i)>1.0e-10&&i<Nlg)
              {
                 for(j=1;j<=Nlg;j++)
                    eigenvec(j)=complex(lgz(j,i),lgz(j,i+1));
              }
              else if(fabs(imag(eigenval))<1.0e-10)
              {
                 for(j=1;j<=Nlg;j++)
                    eigenvec(j)=complex(lgz(j,i),0.0);
              }
              else
              {
                 for(j=1;j<=Nlg;j++)
                    eigenvec(j)=complex(lgz(j,i-1),-lgz(j,i));
```

```
            }
            matrix_mult(-eigenval,BB,tmpc1);
            matrix_sum(AA,tmpc1,tmpc2);
            matrix_mult(tmpc2,eigenvec,tmpvec);
            for(j=1;j<=Nlg;j++)
               Lgcheck[(j-1)*Nlg+i-1]=tmpvec(j);
         }
      }
   }

   //Sort the eigen values
   sort3(lgalfi,lgalfr,sorterlg);

   //copy the eigen values and vectors over to the persistent arrays
   //and count the number of eigen values/vectors (excluding complex conj)
   Nmode=0;
   for(i=1;i<=Nlg;i++)
   {
      Lgalfr[i-1]=lgalfr(i);
      Lgalfi[i-1]=lgalfi(i);
      if(lgalfi(i)>=0.0)
         Nmode++;
   }
   if(matz)
   {
      J=0;
      for(j=1;j<=Nlg;j++)
      {
         if(lgalfi(j)>=0.0||(lgalfi(j)==-1.0&&lgalfr(j)==0.0))
         {
            jj=sorterlg(j);
            J++;
            for(i=1;i<=Nlg;i++)
            {
               Lgz(i,J)=lgz(i,jj);
            }
            if(lgalfi(j)>0.0)
            {
               J++;
               jj++;
               for(i=1;i<=Nlg;i++)
               {
                  Lgz(i,J)=lgz(i,jj);
               }
            }
         }
      }
   }

   if(N!=0&&matz)
   {
      //Calculate the motion of the confluence point for each mode and
      //divide the other motions through by this value to determine the
      //predominant motion for each mode
      //This is calculated for CP motion due to stretch only, angle change
      //only, and both stretch & angle change together
      Lgmodestretch.Resize(2*N+4,Nmode);
      Lgmodeangle.Resize(2*N+4,Nmode);
      Lgmodes.Resize(2*N+4,Nmode);

      //Initialize and fill the cos and sin vectors with trimstate values
      cosG(N);sinG(N);
      iterProfi.Restart();
      //ABout.open("checker.txt",ios::out);
      for(i=N;i>=1;i--)
      {
         cosG(i)=cos(iterProfi.Current());
   //ABout<< iterProfi.current() << "\t" << cosG(i) << "\t" <<
   //     sinG(i) << "\n";
   iterProfi++;
}
//ABout.close();

//*******************************************
//Stretch only

modenum=0;
J=4;                  //start at 4 to skip the invalid eigenvectors
for(j=1;j<=Nlg;j++)   //Iterate through the eigen values
{
   if(lgalfi(j)>=0.0) //Is it a valid eigen value?
   {
      modenum++;      //increment the mode number
      J++;            //increment the column in the eigenvector matrix

      //Is the eigen value/vector complex?
      if(lgalfi(j)>0.0)
      {
         //Add the kite motion to the modes matrix
         for(i=1;i<=4;i++)
            Lgmodestretch(i,modenum)=complex(Lgz(i,J),Lgz(i,J+1));
         //Calc. the total motion (x & z) for each element due to
         //the cable stretch only.
         //Starting a i=5, every odd element is the cumulative motion
         //in the inertial X direction and every even element is the
         //cumulative motion in the inertial Z direction.
         Lgmodestretch(5,modenum)=complex(Lgz(2*N+7,J),
            Lgz(2*N+7,J+1))*cosG(1);
         Lgmodestretch(6,modenum)=complex(Lgz(2*N+7,J),
            Lgz(2*N+7,J+1))*sinG(1);
         for(i=2;i<=N;i++)
         {
            Lgmodestretch(3+2*i,modenum)=complex(Lgz(2*N+5+2*i,J),
               Lgz(2*N+5+2*i,J+1))*cosG(i)+
               Lgmodestretch(1+2*i,modenum);
            Lgmodestretch(4+2*i,modenum)=complex(Lgz(2*N+5+2*i,J),
               Lgz(2*N+5+2*i,J+1))*sinG(i)+
               Lgmodestretch(2+2*i,modenum);
         }
         //Calc. the Mag. of the CP motion
         dcp=pow(pow(Lgmodestretch(2*N+3,modenum),2.0)+
            pow(Lgmodestretch(2*N+4,modenum),2.0),0.5);
         //Normalize the entire mode wrt the total motion of the CP
         for(i=1;i<=2*N+4;i++)
            Lgmodestretch(i,modenum)/=dcp;
         J++;
      }
      else        //Or is the eigen value/vector real only?
      {
         //Add the kite motion to the modes matrix
         for(i=1;i<=4;i++)
            Lgmodestretch(i,modenum)=complex(Lgz(i,J),0.0);
         //Calc. the total motion (x & z) for each element
         Lgmodestretch(5,modenum)=complex(Lgz(2*N+7,J),0.0)*cosG(1);
         Lgmodestretch(6,modenum)=complex(Lgz(2*N+7,J),0.0)*sinG(1);
         for(i=2;i<=N;i++)
         {
            Lgmodestretch(3+2*i,modenum)=complex(Lgz(2*N+5+2*i,J),
               0.0)*cosG(i)+Lgmodestretch(1+2*i,modenum);
            Lgmodestretch(4+2*i,modenum)=complex(Lgz(2*N+5+2*i,J),
               0.0)*sinG(i)+Lgmodestretch(2+2*i,modenum);
         }
         //Calc. the Mag. of the CP motion
```

```
            pow(Lgmodestretch(2*N+4,modenum),2.0),0.5);
         //Normalize the entire mode wrt the total motion of the CP
         for(i=1;i<=2*N+4;i++)
            Lgmodestretch(i,modenum)/=dcp;
      }
   }
}

//*****************************************
//Angle change only

modenum=0;
J=4;                     //start at 4 to skip the invalid eigenvectors
for(j=1;j<=Nlg;j++)      //Iterate through the eigen values
{
   if(lgalfi(j)>=0.0)    //Is it a valid eigen value?
   {
      modenum++;      //increment the mode number
      J++;            //increment the column in the eigenvector matrix

      //Is the eigen value/vector complex?
      if(lgalfi(j)>0.0)
      {
         //Add the kite motion to the modes matrix
         for(i=1;i<=4;i++)
            Lgmodeangle(i,modenum)=complex(Lgz(i,J),Lgz(i,J+1));
         //Calc. the total motion (x & z) for each element due to both
         //the cable stretch and the element angle change.
         //Starting a i=5, every odd element is the cumulative motion
         //in the inertial X direction and every even element is the
         //cumulative motion in the inertial Z direction.
         Lgmodeangle(5,modenum)=-complex(Lgz(2*N+8,J),Lgz(2*N+8,J+1))*
               teleml*sinG(1);
         Lgmodeangle(6,modenum)=complex(Lgz(2*N+8,J),Lgz(2*N+8,J+1))*
               teleml*cosG(1);
         for(i=2;i<=N;i++)
         {
            Lgmodeangle(3+2*i,modenum)=-complex(Lgz(2*N+6+2*i,J),
               Lgz(2*N+6+2*i,J+1))*teleml*sinG(i)+
               Lgmodeangle(1+2*i,modenum);
            Lgmodeangle(4+2*i,modenum)=complex(Lgz(2*N+6+2*i,J),
               Lgz(2*N+6+2*i,J+1))*teleml*cosG(i)+
               Lgmodeangle(2+2*i,modenum);
         }
         //Calc. the Mag. of the CP motion
         dcp=pow(pow(Lgmodeangle(2*N+3,modenum),2.0)+
            pow(Lgmodeangle(2*N+4,modenum),2.0),0.5);
         //Normalize the entire mode wrt the total motion of the CP
         for(i=1;i<=2*N+4;i++)
            Lgmodeangle(i,modenum)/=dcp;
         J++;
      }
      else         //Or is the eigen value/vector real only?
      {
         //Add the kite motion to the modes matrix
         for(i=1;i<=4;i++)
            Lgmodeangle(i,modenum)=complex(Lgz(i,J),0.0);
         //Calc. the total motion (x & z) for each element
         Lgmodeangle(5,modenum)=-complex(Lgz(2*N+8,J),0.0)*
            teleml*sinG(1);
         Lgmodeangle(6,modenum)=complex(Lgz(2*N+8,J),0.0)*
            teleml*cosG(1);
         for(i=2;i<=N;i++)
         {
            Lgmodeangle(3+2*i,modenum)=-complex(Lgz(2*N+6+2*i,J),0.0)*
               teleml*sinG(i)+Lgmodeangle(1+2*i,modenum);
               teleml*cosG(1)+Lgmodeangle(2+2*i,modenum);
         }
         //Calc. the Mag. of the CP motion
         dcp=pow(pow(Lgmodeangle(2*N+3,modenum),2.0)+
            pow(Lgmodeangle(2*N+4,modenum),2.0),0.5);
         //Normalize the entire mode wrt the total motion of the CP
         for(i=1;i<=2*N+4;i++)
            Lgmodeangle(i,modenum)/=dcp;
      }
   }
}

//*******************************************
//Both stretch and angle change

modenum=0;
J=4;                  //start at 4 to skip the invalid eigenvectors
for(j=1;j<=Nlg;j++)   //Iterate through the eigen values
{
   if(lgalfi(j)>=0.0) //Is it a valid eigen value?
   {
      modenum++;      //increment the mode number
      J++;            //increment the column in the eigenvector matrix

      //Is the eigen value/vector complex?
      if(lgalfi(j)>0.0)
      {
         //Add the kite motion to the modes matrix
         for(i=1;i<=4;i++)
            Lgmodes(i,modenum)=complex(Lgz(i,J),Lgz(i,J+1));
         //Calc. the total motion (x & z) for each element due to both
         //the cable stretch and the element angle change.
         //Starting a i=5, every odd element is the cumulative motion
         //in the inertial X direction and every even element is the
         //cumulative motion in the inertial Z direction.
         Lgmodes(5,modenum)=complex(Lgz(2*N+7,J),Lgz(2*N+7,J+1))*
               cosG(1)-complex(Lgz(2*N+8,J),Lgz(2*N+8,J+1))*
               teleml*sinG(1);
         Lgmodes(6,modenum)=complex(Lgz(2*N+8,J),Lgz(2*N+8,J+1))*
               teleml*cosG(1)+complex(Lgz(2*N+7,J),Lgz(2*N+7,J+1))*
               sinG(1);
         for(i=2;i<=N;i++)
         {
            Lgmodes(3+2*i,modenum)=complex(Lgz(2*N+5+2*i,J),Lgz(2*N+5+
               2*i,J+1))*cosG(i)-complex(Lgz(2*N+6+2*i,J),Lgz(2*N+6+
               2*i,J+1))*teleml*sinG(i)+Lgmodes(1+2*i,modenum);
            Lgmodes(4+2*i,modenum)=complex(Lgz(2*N+6+2*i,J),Lgz(2*N+6+
               2*i,J+1))*teleml*cosG(i)+complex(Lgz(2*N+5+2*i,J),
               Lgz(2*N+5+2*i,J+1))*sinG(i)+Lgmodes(2+2*i,modenum);
         }
         //Calc. the Mag. of the CP motion
         dcp=pow(pow(Lgmodes(2*N+3,modenum),2.0)+
            pow(Lgmodes(2*N+4,modenum),2.0),0.5);
         //Normalize the entire mode wrt the total motion of the CP
         for(i=1;i<=2*N+4;i++)
            Lgmodes(i,modenum)/=dcp;
         J++;
      }
      else        //Or is the eigen value/vector real only?
      {
         //Add the kite motion to the modes matrix
         for(i=1;i<=4;i++)
            Lgmodes(i,modenum)=complex(Lgz(i,J),0.0);
         //Calc. the total motion (x & z) for each element
         Lgmodes(5,modenum)=complex(Lgz(2*N+7,J),0.0)*cosG(1)-
               complex(Lgz(2*N+8,J),0.0)*teleml*sinG(1);
```

```
                     *cosG(1)+complex(Lgz(2*N+7,J),0.0)*sinG(1);
               for(i=2;i<=N;i++)
               {
                  Lgmodes(3+2*i,modenum)=complex(Lgz(2*N+5+2*i,J),0.0)*
                     cosG(i)-complex(Lgz(2*N+6+2*i,J),0.0)*teleml*
                     sinG(i)+Lgmodes(1+2*i,modenum);
                  Lgmodes(4+2*i,modenum)=complex(Lgz(2*N+6+2*i,J),0.0)*
                     teleml*cosG(i)+complex(Lgz(2*N+5+2*i,J),0.0)*
                     sinG(i)+Lgmodes(2+2*i,modenum);
               }
               //Calc. the Mag. of the CP motion
               dcp=pow(pow(Lgmodes(2*N+3,modenum),2.0)+
                  pow(Lgmodes(2*N+4,modenum),2.0),0.5);
               //Normalize the entire mode wrt the total motion of the CP
               for(i=1;i<=2*N+4;i++)
                  Lgmodes(i,modenum)/=dcp;
            }
         }
      }
   }
}

XCP=(Xcg-Xcplt)*cos(AOA*pi/180.0)+(Zcg-Zcplt)*sin(AOA*pi/180.0);
ZCP=(Zcg-Zcplt)*cos(AOA*pi/180.0)-(Xcg-Xcplt)*sin(AOA*pi/180.0);

//*********************************************
//Populate the Lateral Matrices

double Ixp,Izp,Izxp;
Ixp=(Ixx*Izz-pow(Ixz,2.0))/Izz;
Izp=(Ixx*Izz-pow(Ixz,2.0))/Ixx;
Izxp=Ixz/(Ixx*Izz-pow(Ixz,2.0));

//Eq. 56a
AAlt(1,1)=Yv;
AAlt(1,2)=Yp;
AAlt(1,3)=Yr-mass*A.Wind;
AAlt(1,4)=mass*g*cos(AOA*pi/180.0)+Tbar*sin(Gammabar)-CG.Ccy*ZCP;
AAlt(1,5)=-Tbar*cos(Gammabar)+CG.Ccy*XCP;
AAlt(1,6)=-Tbar*cos(Gammabar);
iterProfi.Restart();
for(i=2*N;i>=N+1;i--)
{
   Gammai=iterProfi.Current();
   AAlt(1,i+6)=teleml*cos(Gammai)*CG.Ccy;
   iterProfi++;
}
BBlt(1,1)=mass;

//Eq. 56b
AAlt(2,1)=Lv/Ixp+Izxp*Nv;
AAlt(2,2)=Lp/Ixp+Izxp*Np;
AAlt(2,3)=Lr/Ixp+Izxp*Nr;
AAlt(2,4)=-Tbar*sin(Gammabar)*(ZCP/Ixp-Izxp*XCP)+CG.Ccl/Ixp;
AAlt(2,5)=-Tbar*cos(Gammabar)*(Izxp*XCP-ZCP/Ixp)+CG.Ccn*Izxp;;
AAlt(2,6)=Tbar*cos(Gammabar)*(ZCP/Ixp-Izxp*XCP);
BBlt(2,2)=1.0;


//Eq. 56c
AAlt(3,1)=Lv*Izxp+Nv/Izp;
AAlt(3,2)=Lp*Izxp+Np/Izp;
AAlt(3,3)=Lr*Izxp+Nr/Izp;
AAlt(3,4)=-Tbar*sin(Gammabar)*(ZCP*Izxp-XCP/Izp)+CG.Ccl*Izxp;
AAlt(3,5)=-Tbar*cos(Gammabar)*(XCP/Izp-ZCP*Izxp)+CG.Ccn/Izp;;
AAlt(3,6)=Tbar*cos(Gammabar)*(ZCP*Izxp-XCP/Izp);
```

```
//Eq. 38h
AAlt(4,2)=1.0;
AAlt(4,3)=-tan(AOA*pi/180.0);
BBlt(4,4)=1.0;

//Eq. 38i
AAlt(5,3)=1.0/cos(AOA*pi/180.0);
BBlt(5,5)=1.0;

//Eq. 42b
AAlt(6,1)=1.0;
AAlt(6,2)=-ZCP;
AAlt(6,3)=XCP;
AAlt(6,5)=A.Wind;
iterProfi.Restart();
for(i=N;i>=1;i--)
{
   Gammai=iterProfi.Current();
   AAlt(6,i+6)=-teleml*cos(Gammai);
   iterProfi++;
}

//Pb (Eq. 30)
iterProfi.Restart();
for(i=N;i>=1;i--)
{
   Gammai=iterProfi.Current();
   AAlt(i+6,6)=teleml*cos(Gammai)*Tbar*cos(Gammabar);
   iterProfi++;
}

iterProfi.Restart();
for(i=N;i>=1;i--)
{
   Gammai=iterProfi.Current();
   iterProfk.Restart();
   for(k=N;k>=1;k--)
   {
      Gammak=iterProfk.Current();

      //Mb (Eq. 8)
      //Calculate the summation of mass above max(i,k)
      Iij=0.0;
      for(j=max(i,k);j<N;j++)
      {
         Iij+=pi*pow(Tdia(j)/2.0,2.0)*teleml*tdens;
      }
      Iij+=pi*pow(Tdia(N)/2.0,2.0)*teleml*tdens/2.0;
      BBlt(i+6,k+6)=teleml*teleml*cos(Gammai)*cos(Gammak)*Iij;



      //-Fb (Eq. 29)
      if(i>k)
      {
         iterProfj.Restart();
         sum1=0.0;
         for(j=N;j>=i;j--)
         {
            Gammaj=iterProfj.Current();
            sum1+=rho(j)*Tdia(j)*teleml*sin(Gammaj)*E(i,j);
            iterProfj++;
         }

         AAlt(i+6,k+6)=-0.5*tCDcyl*A.Wind*teleml*cos(Gammai)*teleml*
```

```
        else if(i<=k)
        {
           iterProfj.Restart();
           sum1=0.0;
           for(j=N;j>=k;j--)
           {
              Gammaj=iterProfj.Current();
              sum1+=rho(j)*Tdia(j)*teleml*sin(Gammaj)*E(i,j)*E(k,j);
              iterProfj++;
           }

           AAlt(i+6,k+6)=-0.5*tCDcyl*A.Wind*teleml*cos(Gammai)*teleml*
                          cos(Gammak)*sum1;
        }

        //Set the N x N identity matrix in AAlt and BBlt
        //
        if(i==k)
        {
           AAlt(i+N+6,k+6)=1.0;
           BBlt(i+N+6,k+N+6)=1.0;
        }

        //Set -Eb in AAlt (Eq 28)
        //
        if(i==k)
        {
           iterProfj.Restart();
           sum1=0.0;
           for(j=N;j>=i;j--)
           {
              Gammaj=iterProfj.Current();
              sum1+=rho(j)*Tdia(j)*teleml*pow(sin(Gammaj),3.0)*E(i,j);
              iterProfj++;
           }

           AAlt(i+6,k+N+6)=-0.5*tCDcyl*pow(A.Wind,2.0)*teleml*cos(Gammai)*
                    (sum1-0.5*teleml*pow(cos(Gammai),2.0)*rho(i)*Tdia(i)*
                    sin(Gammai))-teleml*cos(Gammai)*Tbar*cos(Gammabar);
        }
        else if(i<k)
        {
           AAlt(i+6,k+N+6)=0.5*tCDcyl*pow(A.Wind,2.0)*teleml*cos(Gammai)*
                          rho(k)*Tdia(k)*teleml*sin(Gammak)*
                          pow(cos(Gammak),2.0);
        }

        iterProfk++;
     }
     iterProfi++;
  }

  //Copy the A and B matrices to backup matrices for post check
  if(CheckEV && matz)
  {
     AA.Resize(Nlt,Nlt);
     BB.Resize(Nlt,Nlt);
     eigenvec.Resize(Nlt);tmpc1.Resize(Nlt,Nlt);
     tmpc2.Resize(Nlt,Nlt);tmpvec.Resize(Nlt);
     AA=AAlt;
     BB=BBlt;
  }

  //Save the A and B matrices to MatLab script file for
  // the eigen cross-check
```

```
{
   ABout.open("Lteigchk.m",ios::out);
   ABout<< setprecision(24);
   ABout<<"AAlt = [ ";
   for(i=1;i<=Nlt;i++)
   {
      for(j=1;j<=Nlt;j++)
      {
         ABout << AAlt(i,j) << " ";
      }
      ABout << "; \n";
   }
   ABout << "]\n";

   ABout<<"BBlt = [ ";
   for(i=1;i<=Nlt;i++)
   {
      for(j=1;j<=Nlt;j++)
      {
         ABout << BBlt(i,j) << " ";
      }
      ABout << "; \n";
   }
   ABout << "]\n\n[V,D] = eig(AAlt,BBlt)\n";
   ABout << "save ltresult.txt D V  -ascii -double -tabs\n";

   ABout.close();
}


//Call the eigen solver routine for the lateral case
//
errlt=rgg(AAlt,BBlt,ltalfr,ltalfi,ltbeta,matz,ltz);

//If the function rgg returns an integer zero the function was successful
//
if(errlt==0)
{
   EigltValid=1;

   //If the eigenvalue is valid divide the real and imaginary parts
   //by the beta value
   for(i=1;i<=Nlt;i++)
   {
      sorterlt(i)=i;
      if(fabs(ltbeta(i))>1.0e-12)
      {
         ltalfr(i)=ltalfr(i)/ltbeta(i);
         ltalfi(i)=ltalfi(i)/ltbeta(i);
      }
      else
      {
         ltalfr(i)=0.0;
         ltalfi(i)=-1.0;
      }
   }

   //Check the eigen solution if required
   if(CheckEV && matz)
   {
      for(i=1;i<=Nlt;i++)
      {
         if(ltalfr(i)==0.0&&ltalfi(i)==-1.0)
         {
            for(j=1;j<=Nlt;j++)
               Ltcheck[(j-1)*Nlt+i-1]=complex(0.0,-1.0);
```

```
        else
        {
           eigenval=complex(ltalfr(i),ltalfi(i));
           if(ltalfi(i)>1.0e-10&&i<Nlt)
           {
              for(j=1;j<=Nlt;j++)
                 eigenvec(j)=complex(ltz(j,i),ltz(j,i+1));
           }
           else if(fabs(imag(eigenval))<1.0e-10)
           {
              for(j=1;j<=Nlt;j++)
                 eigenvec(j)=complex(ltz(j,i),0.0);
           }
           else
           {
              for(j=1;j<=Nlt;j++)
                 eigenvec(j)=complex(ltz(j,i-1),-ltz(j,i));
           }
           matrix_mult(-eigenval,BB,tmpc1);
           matrix_sum(AA,tmpc1,tmpc2);
           matrix_mult(tmpc2,eigenvec,tmpvec);
           for(j=1;j<=Nlt;j++)
              Ltcheck[(j-1)*Nlt+i-1]=tmpvec(j);
        }
     }
}

//Sort the eigen values
sort3(ltalfi,ltalfr,sorterlt);

//copy the eigen values and vectors over to the persistent arrays
//and count the number of eigen values/vectors (excluding complex conj)
Nmode=0;
for(i=1;i<=Nlt;i++)
{
   Ltalfr[i-1]=ltalfr(i);
   Ltalfi[i-1]=ltalfi(i);
   if(ltalfi(i)>=0.0)
      Nmode++;
}
if(matz)                 //if the eigen vectors were calculated
{
   J=0;
   for(j=1;j<=Nlt;j++)
   {
      if(ltalfi(j)>=0.0||(ltalfi(j)==-1.0&&ltalfr(j)==0.0))
      {
         jj=sorterlt(j);
         J++;
         for(i=1;i<=Nlt;i++)
         {
            Ltz(i,J)=ltz(i,jj);
         }
         if(ltalfi(j)>0.0)
         {
            J++;
            jj++;
            for(i=1;i<=Nlt;i++)
            {
               Ltz(i,J)=ltz(i,jj);
            }
         }
      }
   }
}
```

```
        {
           //Calculate the motion of the confluence point for each mode and
           //divide the other motions through by this value to determine the
           //predominant motion for each mode
           Ltmodes.Resize(N+5,Nmode);

           modenum=0;
           J=2;                      //start at 2 to skip the invalid eigenvectors
           for(j=1;j<=Nlt;j++)       //Iterate through the eigen values
           {
              if(ltalfi(j)>=0.0)  //Is it a valid eigen value?
              {
                 modenum++;       //increment the mode number
                 J++;             //increment the column in the eigenvector matrix
                 //Is the eigen value/vector complex?
                 if(ltalfi(j)>0.0)
                 {
                    //Add the kite motion to the modes matrix
                    for(i=1;i<=5;i++)
                       Ltmodes(i,modenum)=complex(Ltz(i,J),Ltz(i,J+1));
                    //Calc. the total motion (y) for each element
                    Ltmodes(6,modenum)=complex(Ltz(N+7,J),Ltz(N+7,J+1))
                                        *teleml*cosG(1);
                    for(i=2;i<=N;i++)
                    {
                       Ltmodes(5+i,modenum)=complex(Ltz(N+6+i,J),Lgz(N+6+i,J+1))
                                        *teleml*cosG(i)+Ltmodes(4+i,modenum);
                    }
                    //Calc. the Mag. of the CP motion
                    dcp=Ltmodes(N+5,modenum);
                    //Normalize the entire mode wrt the total motion of the CP
                    for(i=1;i<=N+5;i++)
                       Ltmodes(i,modenum)/=dcp;
                    J++;
                 }
                 else        //Or is the eigen value/vector real only?
                 {
                    //Add the kite motion to the modes matrix
                    for(i=1;i<=5;i++)
                       Ltmodes(i,modenum)=complex(Ltz(i,J),0.0);
                    //Calc. the total motion (y) for each element
                    Ltmodes(6,modenum)=complex(Ltz(N+7,J),0.0)
                                        *teleml*cosG(1);
                    for(i=2;i<=N;i++)
                    {
                       Ltmodes(5+i,modenum)=complex(Ltz(N+6+i,J),0.0)
                                        *teleml*cosG(i)+Ltmodes(4+i,modenum);
                    }
                    //Calc. the Mag. of the CP motion
                    dcp=Ltmodes(N+5,modenum);
                    //Normalize the entire mode wrt the total motion of the CP
                    for(i=1;i<=N+5;i++)
                       Ltmodes(i,modenum)/=dcp;
                 }
              }
           }
        }
     }
     //If the problem only had one tether element we need to reset N to that
     //value
     if(N<=1)
        N=1;

     return errlg+errlt;
  }
```

```
//
//Super Kite Timestep Analysis
//

#include <stdio.h>
#include <stdlib.h>
#include <cstring.h>
#include <math.h>
#include <classlib/arrays.h>
#include <float.h>
#include <fstream.h>
#include <iomanip.h>
#include "matrix.h"
#include "suprkite.h"

// Initialize the matrices neccessary for the timestep analysis.
// CG contains the values of the six control gains.
// AltData contains the Altitude data for a given trim-state.

void
KiteData::TimeStepAnalysis(ControlGain& CG, AltitudeData& AltData, bool TSOut,
                           bool KCMOut)
{
   //Initialize the output file for the MatLab cross-check
   fstream TSout;

   //Create temporary Altitude object and init. with current kite altitude
   Altitude A(Alttd);

   //Get the data assoc. with the altitude from the datafile
   AltData.GetData(A, AltData);


   const double altllim=AltData.MinAlt;
   const double altulim=AltData.MaxAlt;


   //Create three iterators to step through the tether profile data
   TViter iterProfi(*Profile);
   TViter iterProfj(*Profile);
   TViter iterProfk(*Profile);

   int i,j,k,Nlg,Nlt,Steps;
   int proceed=1;
   double Gammai,Gammaj,Gammak,sum1,sum2,sum3,curralt,Iij,Iii,tethki,ZCP,XCP;

   //If there is only one tether element the problem degenerates to a massless
   //dragless rigid element
   if(N<=1)
      N=0;

   //order of the matrices
   Nlg=N*2+5;
   Nlt=N+4;

   //Initialize temporary matrices
   matrix LgK(Nlg,Nlg),LgC(Nlg,Nlg),LgM(Nlg,Nlg),LgKBar(Nlg,Nlg);
   matrix LgFBar(Nlg),Lgtemp1(Nlg,Nlg),LgDel(Nlg),LgDelDot(Nlg),LgDelDDot(Nlg);
   matrix LgPDel(Nlg),LgPDelDot(Nlg),LgPDelDDot(Nlg);
   matrix LgKBarInv(Nlg,Nlg),Lgtemp2(Nlg),Lgtemp3(Nlg);
   matrix LtK(Nlt,Nlt),LtC(Nlt,Nlt),LtM(Nlt,Nlt),LtKBar(Nlt,Nlt);
   matrix LtFBar(Nlt),Lttemp1(Nlt,Nlt),LtDel(Nlt),LtDelDot(Nlt),LtDelDDot(Nlt);
   matrix LtPDel(Nlt),LtPDelDot(Nlt),LtPDelDDot(Nlt);
   matrix LtKBarInv(Nlt,Nlt),Lttemp2(Nlt),Lttemp3(Nlt);
```

```
Steps=TotalTime/TimeStep+1;
LgTS.Resize(Nlg,Steps);
LtTS.Resize(Nlt,Steps);

//Vectors for the air density and tether diameter variations as a
//function of the tether element number
matrix rho,Tdia;
if(N!=0)
{
   rho(N);Tdia(N);
}else{
   rho(2);Tdia(2);
}

//
//calculate the air density and tether diameter at every element
//(starting from the confluence point)
//
iterProfi.Restart();
sum1=0.0;
if(N==0) N=1;
for(i=N;i>=1;i--)
{
   curralt=Alttd-teleml/2.0*sin(iterProfi.Current())-sum1;
   sum1+=teleml*sin(iterProfi.Current());
   if(curralt<altllim)
      A.Set(altllim);
   else if(curralt>altulim)
      A.Set(altulim);
   else
      A.Set(curralt);
   AltData.GetData(A, AltData);
   rho(i)=A.Dens;
   if(N<=1)
      Tdia(i)=tdia;
   else
      Tdia(i)=tdiao+(tdia-tdiao)*(i-1)/(N-1);
   iterProfi++;
}
if(N==1) N=0;

//Set A back to kite altitude
//
A.Set(Alttd);
AltData.GetData(A, AltData);

//*****************************************************************
//Populate the Longitudinal Matrices if the analysis is required

if(XTSSet || ZTSSet || MTSSet)
{

   XCP=(Xcg-Xcplg)*cos(AOA*pi/180.0)+(Zcg-Zcplg)*sin(AOA*pi/180.0);
   ZCP=(Zcg-Zcplg)*cos(AOA*pi/180.0)-(Xcg-Xcplg)*sin(AOA*pi/180.0);

   //Eq. 55a
   LgC(1,1)=Xu;
   LgC(1,2)=Xw;
   LgK(1,3)=CG.Ccx*ZCP-mass*g*cos(AOA*pi/180.0)-Tbar*sin(Gammabar);
   LgK(1,4)=cos(Gammabar);
   LgK(1,5)=-Tbar*sin(Gammabar);
   iterProfi.Restart();
   for(i=N;i>=1;i--)
   {
      Gammai=iterProfi.Current();
      LgK(1,i*2+4)=-cos(Gammai)*CG.Ccx;
   iterProfi++;
}
LgM(1,1)=-mass;

//Eq. 55b
LgC(2,1)=Zu;
LgC(2,2)=Zw;
LgC(2,3)=-mass*A.Wind-Zq;
LgK(2,3)=mass*g*sin(AOA*pi/180.0)+Tbar*cos(Gammabar)-CG.Ccz*XCP;
LgK(2,4)=sin(Gammabar);
LgK(2,5)=Tbar*cos(Gammabar);
iterProfi.Restart();
for(i=N;i>=1;i--)
{
   Gammai=iterProfi.Current();
   LgK(2,i*2+4)=-sin(Gammai)*CG.Ccz;
   LgK(2,i*2+5)=-teleml*cos(Gammai)*CG.Ccz;
   iterProfi++;
}
LgM(2,2)=-mass+Zwdot;

//Eq. 55c
double Mwdp=Mwdot/(mass-Zwdot);
LgC(3,1)=Mu+Mwdp*Zu;
LgC(3,2)=Mw+Mwdp*Zw;
LgC(3,3)=Mq+Mwdp*(mass*A.Wind+Zq);
LgK(3,3)=CG.Ccm-Mwdp*(mass*g*sin(AOA*pi/180.0)-Tbar*cos(Gammabar)
          +CG.Ccz*XCP)-Tbar*(XCP*cos(Gammabar)+ZCP*sin(Gammabar));
LgK(3,4)=ZCP*cos(Gammabar)-XCP*sin(Gammabar)+Mwdp*sin(Gammabar);
LgK(3,5)=-Tbar*(XCP*cos(Gammabar)+ZCP*sin(Gammabar)-Mwdp*cos(Gammabar));
iterProfi.Restart();
for(i=2*N;i>=N+1;i--)
{
   Gammai=iterProfi.Current();
   LgK(3,i*2+4)=-Mwdp*sin(Gammai)*CG.Ccz;
   LgK(3,i*2+5)=-Mwdp*teleml*cos(Gammai)*CG.Ccz;
   iterProfi++;
}
LgM(3,3)=-Iyy;

//Eq. 42a
LgC(4,1)=1.0;
LgC(4,3)=ZCP;
iterProfi.Restart();
for(i=N;i>=1;i--)
{
   Gammai=iterProfi.Current();
   LgC(4,i*2+4)=cos(Gammai);
   LgC(4,i*2+5)=-teleml*sin(Gammai);
   iterProfi++;
}

//Eq. 42c
LgC(5,2)=1.0;
LgC(5,3)=-XCP;
LgK(5,3)=-A.Wind;
iterProfi.Restart();
for(i=N;i>=1;i--)
{
   Gammai=iterProfi.Current();
   LgC(5,i*2+4)=sin(Gammai);
   LgC(5,i*2+5)=teleml*cos(Gammai);
   iterProfi++;
}

//-Pa (Eq. 26)
```

```
for(i=N;i>=1;i--)
{
   Gammai=iterProfi.Current();
   LgK(i*2+4,4)=-cos(Gammabar-Gammai);
   LgK(i*2+4,5)=Tbar*sin(Gammabar-Gammai);
   LgK(i*2+5,4)=-teleml*sin(Gammabar-Gammai);
   LgK(i*2+5,5)=-Tbar*teleml*cos(Gammabar-Gammai);
   iterProfi++;
}

iterProfi.Restart();
for(i=N;i>=1;i--)
{
   Gammai=iterProfi.Current();
   iterProfk.Restart();
   for(k=N;k>=1;k--)
   {
      Gammak=iterProfk.Current();

      //Ma (Eq. 7)
      //Calculate the summation of mass above max(i,k)
      Iij=0.0;
      for(j=max(i,k);j<N;j++)
      {
         Iij+=pi*pow(Tdia(j)/2.0,2.0)*teleml*tdens;
      }
      Iij+=pi*pow(Tdia(N)/2.0,2.0)*teleml*tdens/2.0;
      LgM(i*2+4,k*2+4)=cos(Gammai-Gammak)*Iij;
      LgM(i*2+4,k*2+5)=teleml*sin(Gammai-Gammak)*Iij;
      LgM(i*2+5,k*2+4)=-teleml*sin(Gammai-Gammak)*Iij;
      LgM(i*2+5,k*2+5)=teleml*teleml*cos(Gammai-Gammak)*Iij;

      //Fa (Eq. 25)
      if(i>=k)
      {
         //Note: modify in future to have the velocity vary for each segment
         //of the tether
         iterProfj.Restart();
         sum1=0.0;
         sum2=0.0;
         for(j=N;j>=i+1;j--)
         {
            Gammaj=iterProfj.Current();
            sum1+=rho(j)*Tdia(j)*sin(Gammaj)*sin(Gammai-Gammaj)*teleml
                  *sin(Gammaj-Gammak);
            sum2+=rho(j)*Tdia(j)*sin(Gammaj)*sin(Gammai-Gammaj)*teleml
                  *cos(Gammaj-Gammak);
            iterProfj++;
         }

         LgC(i*2+4,k*2+4)=-A.Wind*tCDcyl*sum1+Nc*delta(i,k);
         LgC(i*2+4,k*2+5)=A.Wind*tCDcyl*teleml*sum2;
      }
      if(i>k)
      {
         iterProfj.Restart();
         sum1=0.0;
         sum2=0.0;
         for(j=N;j>=i;j--)
         {
            Gammaj=iterProfj.Current();
            sum1+=rho(j)*Tdia(j)*sin(Gammaj)*teleml*cos(Gammai-Gammaj)
                  *sin(Gammaj-Gammak)*E(i,j);
            sum2+=rho(j)*Tdia(j)*sin(Gammaj)*teleml*cos(Gammai-Gammaj)
                  *cos(Gammaj-Gammak)*E(i,j);
            iterProfj++;
   LgC(i*2+5,k*2+4)=-A.Wind*tCDcyl*teleml*sum1;
   LgC(i*2+5,k*2+5)=A.Wind*tCDcyl*teleml*teleml*sum2;
}
if(i<=k)
{
   iterProfj.Restart();
   sum1=0.0;
   sum2=0.0;
   for(j=N;j>=k;j--)
   {
      Gammaj=iterProfj.Current();
      if(j>=k+1)
         sum1+=rho(j)*Tdia(j)*sin(Gammaj)*teleml*cos(Gammai-Gammaj)
            *sin(Gammaj-Gammak);
      sum2+=rho(j)*Tdia(j)*sin(Gammaj)*teleml*cos(Gammai-Gammaj)
            *cos(Gammaj-Gammak)*E(i,j)*E(k,j);
      iterProfj++;
   }

   LgC(i*2+5,k*2+4)=-A.Wind*tCDcyl*teleml*sum1;
   LgC(i*2+5,k*2+5)=A.Wind*tCDcyl*teleml*teleml*sum2;
}
if(i<k)
{
   iterProfj.Restart();
   sum1=0.0;
   sum2=0.0;
   for(j=N;j>=k;j--)
   {
      Gammaj=iterProfj.Current();
      if(j>=k+1)
         sum1+=rho(j)*Tdia(j)*sin(Gammaj)*sin(Gammai-Gammaj)*teleml
            *sin(Gammaj-Gammak);
      sum2+=rho(j)*Tdia(j)*sin(Gammaj)*sin(Gammai-Gammaj)*teleml
            *cos(Gammaj-Gammak)*E(j,k);
      iterProfj++;
   }

   LgC(i*2+4,k*2+4)=-A.Wind*tCDcyl*sum1;
   LgC(i*2+4,k*2+5)=A.Wind*tCDcyl*teleml*sum2;
}

//Set N + Ea (Eq 9 & 26)
//
//Calculate the summation of mass above max(i,i)
Iii=0.0;
for(j=i;j<N;j++)
{
   Iii+=pi*pow(Tdia(j)/2.0,2.0)*teleml*tdens;
}
Iii+=pi*pow(Tdia(N)/2.0,2.0)*teleml*tdens/2.0;
//Add the N component (Eq. 9)
//
tethki=pi*pow(Tdia(i)/2.0,2.0)*tethE/teleml;
LgK(i*2+4,k*2+4)=tethki*delta(i,k);
LgK(i*2+5,k*2+5)=-g*teleml*sin(Gammai)*Iii*delta(i,k);
LgK(i*2+4,k*2+5)=LgK(i*2+5,k*2+4)=g*cos(Gammai)*Iii*delta(i,k);

//Add the Ea component (Eq. 26)
//
if(i==k)
{
   iterProfj.Restart();
   sum1=0.0;
   sum2=0.0;
```

```
         {
            Gammaj=iterProfj.Current();
            if(j>=i+1)
            {
               sum1+=rho(j)*Tdia(j)*teleml*pow(sin(Gammaj),2.0)
                     *cos(Gammai-Gammaj);
               sum3+=rho(j)*Tdia(j)*teleml*pow(sin(Gammaj),2.0)
                     *sin(Gammai-Gammaj);
            }
            sum2+=rho(j)*Tdia(j)*teleml*pow(sin(Gammaj),2.0)
                  *cos(Gammai-Gammaj)*E(i,j);
            iterProfj++;
         }

         LgK(i*2+4,k*2+5)+=0.5*tCDcyl*pow(A.Wind,2.0)*sum1-Tbar
                     *sin(Gammabar-Gammai);
         LgK(i*2+5,k*2+4)+=0.5*tCDcyl*pow(A.Wind,2.0)*(sum2+0.5*teleml
                     *rho(i)*Tdia(i)*pow(sin(Gammai),2.0))-Tbar
                     *sin(Gammabar-Gammai);
         LgK(i*2+5,k*2+5)+=-0.5*tCDcyl*pow(A.Wind,2.0)*teleml*(sum3-
                     teleml*rho(i)*Tdia(i)*sin(Gammai)*cos(Gammai))+Tbar
                     *teleml*cos(Gammabar-Gammai);
      }
      else if(i<k)
      {
         LgK(i*2+4,k*2+4)+=0.5*tCDcyl*pow(A.Wind,2.0)*rho(k)*Tdia(k)
                     *pow(sin(Gammak),2.0)*sin(Gammai-Gammak);
         LgK(i*2+4,k*2+5)+=0.5*tCDcyl*pow(A.Wind,2.0)*rho(k)*Tdia(k)*
                     teleml*sin(Gammak)*(2.0*cos(Gammak)*sin(Gammai-
                     Gammak)-sin(Gammak)*cos(Gammai-Gammak));
         LgK(i*2+5,k*2+4)+=0.5*tCDcyl*pow(A.Wind,2.0)*teleml*rho(k)*
                     Tdia(k)*pow(sin(Gammak),2.0)*cos(Gammai-Gammak);
         LgK(i*2+5,k*2+5)+=0.5*tCDcyl*pow(A.Wind,2.0)*teleml*rho(k)*
                     Tdia(k)*teleml*sin(Gammak)*(2.0*cos(Gammak)*cos(Gammai-
                     Gammak)+sin(Gammak)*sin(Gammai-Gammak));
      }

      iterProfk++;
   }
   iterProfi++;
}

//Save the K, C, and M matrices to text file for future evaluation
//
if(KCMOut)
{
   TSout.open("LgKCM.txt",ios::out);
   TSout<< setprecision(24);
   TSout << "K matrix\n";
   TSout << LgK;
   TSout << "C matrix\n";
   TSout << LgC;
   TSout << "M matrix\n";
   TSout << LgM;
   TSout.close();
}

//Construct KBar
matrix_mult(2.0/TimeStep,LgC,LgKBar);
matrix_sum(LgKBar,LgK,LgKBar);
matrix_mult(4.0/pow(TimeStep,2.0),LgM,Lgtemp1);
matrix_sum(LgKBar,Lgtemp1,LgKBar);

//Invert KBar and specify whether the inversion was successful
if(matrix_inv(LgKBar,LgKBarInv)!=0)
```

```
//Loop through the time steps
i=1;
while (i<=Steps && proceed)
{
   matrix_zero(LgFBar);

   //Form FBar
   if(XTSSet && XTSDur>=TimeStep*i)
      LgFBar(1)=-XTS;
   if(ZTSSet && ZTSDur>=TimeStep*i)
      LgFBar(2)=-ZTS;
   if(MTSSet && MTSDur>=TimeStep*i)
      LgFBar(3)=-MTS;

   matrix_mult(2.0/TimeStep,LgDel,Lgtemp2);
   matrix_sum(Lgtemp2,LgDelDot,Lgtemp2);
   matrix_mult(LgC,Lgtemp2,Lgtemp3);
   matrix_sum(LgFBar,Lgtemp3,LgFBar);

   matrix_mult(4.0/pow(TimeStep,2.0),LgDel,Lgtemp2);
   matrix_mult(4.0/TimeStep,LgDelDot,Lgtemp3);
   matrix_sum(Lgtemp2,Lgtemp3,Lgtemp3);
   matrix_sum(Lgtemp3,LgDelDDot,Lgtemp3);
   matrix_mult(LgM,Lgtemp3,Lgtemp2);
   matrix_sum(LgFBar,Lgtemp2,LgFBar);            //Gives New FBar

   //Multiply LgKBarInv by FBar to find the Delta Vectors
   matrix_mult(LgKBarInv,LgFBar,LgPDel);         //Gives New Delta
   matrix_mult(TimeStep,LgDelDot,Lgtemp2);
   matrix_sum(Lgtemp2,LgDel,Lgtemp2);
   matrix_mult(-1.0,Lgtemp2,Lgtemp3);
   matrix_sum(Lgtemp3,LgPDel,Lgtemp3);
   matrix_mult(2.0/pow(TimeStep,2.0),Lgtemp3,Lgtemp2);   //Gives DeltaDDotav
   matrix_mult(TimeStep,Lgtemp2,Lgtemp3);
   matrix_sum(Lgtemp3,LgDelDot,LgPDelDot);       //Gives New DeltaDot
   matrix_mult(2.0,Lgtemp2,Lgtemp3);
   matrix_mult(-1.0,LgDelDDot,Lgtemp2);
   matrix_sum(Lgtemp3,Lgtemp2,LgPDelDDot);    //Gives New DeltaDDot

   //Assign old Delta's to storage array and reset them with the new Delta's
   for(j=1;j<=Nlg;j++)
   {
      LgTS(j,i)=LgDel(j);
   }
   LgDel=LgPDel;
   LgDelDot=LgPDelDot;
   LgDelDDot=LgPDelDDot;

   i++;
}

//Save the Delta vectors to text file for future evaluation
//
if(TSOut)
{
   TSout.open("LgTStep.txt",ios::out);
   TSout<< setprecision(16);
   TSout << "\nLgTS\nTime Step = " << TimeStep;
   TSout << "\nTotal Time = " << TotalTime << "\n\n";
   for(j=1;j<=LgTS.n;j++)
   {
      TSout << TimeStep*(j-1) << ",";
      for(i=1;i<=LgTS.m;i++)
      {
         TSout << LgTS(i,j) << ",";
```

```
        }
        TSout.close();
      }
    }

    //*********************************************************
    //Populate the Lateral Matrices if the analysis is required

    if(YTSSet || LTSSet || NTSSet)
    {

      XCP=(Xcg-Xcplt)*cos(AOA*pi/180.0)+(Zcg-Zcplt)*sin(AOA*pi/180.0);
      ZCP=(Zcg-Zcplt)*cos(AOA*pi/180.0)-(Xcg-Xcplt)*sin(AOA*pi/180.0);

      double Ixp,Izp,Izxp;
      Ixp=(Ixx*Izz-pow(Ixz,2.0))/Izz;
      Izp=(Ixx*Izz-pow(Ixz,2.0))/Ixx;
      Izxp=Ixz/(Ixx*Izz-pow(Ixz,2.0));

      //Eq. 66a
      LtC(1,1)=Yv;
      LtC(1,2)=Yp;
      LtC(1,3)=(Yp*tan(AOA*pi/180.0)+Yr-mass*A.Wind)*cos(AOA*pi/180.0);
//    LtK(1,1)=0.05;
      LtK(1,2)=mass*g*cos(AOA*pi/180.0)+Tbar*sin(Gammabar)-CG.Ccy*ZCP;
      LtK(1,3)=-Tbar*cos(Gammabar)+CG.Ccy*XCP;
      LtK(1,4)=-Tbar*cos(Gammabar);
      iterProfi.Restart();
      for(i=N;i>=1;i--)
      {
        Gammai=iterProfi.Current();
        LtK(1,i+4)=teleml*cos(Gammai)*CG.Ccy;
        iterProfi++;
      }
      LtM(1,1)=-mass;

      //Eq. 66b
      LtC(2,1)=Lv/Ixp+Izxp*Nv;
      LtC(2,2)=Lp/Ixp+Izxp*Np;
      LtC(2,3)=sin(AOA*pi/180.0)*(Lp/Ixp+Izxp*Np)
              +cos(AOA*pi/180.0)*(Lr/Ixp+Izxp*Nr);
      LtK(2,2)=-Tbar*sin(Gammabar)*(ZCP/Ixp-Izxp*XCP)+CG.Ccl/Ixp;
      LtK(2,3)=-Tbar*cos(Gammabar)*(Izxp*XCP-ZCP/Ixp)+CG.Ccn*Izxp;;
      LtK(2,4)=Tbar*cos(Gammabar)*(ZCP/Ixp-Izxp*XCP);
      LtM(2,2)=-1.0;
      LtM(2,3)=-sin(AOA*pi/180.0);

      //Eq. 66c
      LtC(3,1)=Lv*Izxp+Nv/Izp;
      LtC(3,2)=Lp*Izxp+Np/Izp;
      LtC(3,3)=sin(AOA*pi/180.0)*(Lp*Izxp+Np/Izp)
              +cos(AOA*pi/180.0)*(Lr*Izxp+Nr/Izp);
      LtK(3,2)=-Tbar*sin(Gammabar)*(ZCP*Izxp-XCP/Izp)+CG.Ccl*Izxp;
      LtK(3,3)=-Tbar*cos(Gammabar)*(XCP/Izp-ZCP*Izxp)+CG.Ccn/Izp;;
      LtK(3,4)=Tbar*cos(Gammabar)*(ZCP*Izxp-XCP/Izp);
      LtM(3,3)=-cos(AOA*pi/180.0);

      //Eq. 67
      LtC(4,1)=-1.0;
      LtC(4,2)=ZCP;
      LtC(4,3)=ZCP*sin(AOA*pi/180.0)-XCP*cos(AOA*pi/180.0);
      LtK(4,3)=-A.Wind;
      iterProfi.Restart();
      for(i=N;i>=1;i--)
      {
```

```
   LtC(4,i+4)=teleml*cos(Gammai);
   iterProfi++;
}

//-Pb (Eq. 30)
iterProfi.Restart();
for(i=N;i>=1;i--)
{
   Gammai=iterProfi.Current();
   LtK(i+4,4)=-teleml*cos(Gammai)*Tbar*cos(Gammabar);
   iterProfi++;
}

iterProfi.Restart();
for(i=N;i>=1;i--)
{
   Gammai=iterProfi.Current();
   iterProfk.Restart();
   for(k=N;k>=1;k--)
   {
      Gammak=iterProfk.Current();

      //Mb (Eq. 8)
      //Calculate the summation of mass above max(i,k)
      Iij=0.0;
      for(j=max(i,k);j<N;j++)
      {
         Iij+=pi*pow(Tdia(j)/2.0,2.0)*teleml*tdens;
      }
      Iij+=pi*pow(Tdia(N)/2.0,2.0)*teleml*tdens/2.0;
      LtM(i+4,k+4)=teleml*teleml*cos(Gammai)*cos(Gammak)*Iij;

      //Fb (Eq. 29)
      if(i>k)
      {
         iterProfj.Restart();
         sum1=0.0;
         for(j=N;j>=i;j--)
         {
            Gammaj=iterProfj.Current();
            sum1+=rho(j)*Tdia(j)*teleml*sin(Gammaj)*E(i,j);
            iterProfj++;
         }

         LtC(i+4,k+4)=0.5*tCDcyl*A.Wind*teleml*cos(Gammai)*teleml*
                         cos(Gammak)*sum1;
      }
      else if(i<=k)
      {
         iterProfj.Restart();
         sum1=0.0;
         for(j=N;j>=k;j--)
         {
            Gammaj=iterProfj.Current();
            sum1+=rho(j)*Tdia(j)*teleml*sin(Gammaj)*E(i,j)*E(k,j);
            iterProfj++;
         }

         LtC(i+4,k+4)=0.5*tCDcyl*A.Wind*teleml*cos(Gammai)*teleml*
                         cos(Gammak)*sum1;
      }

      //Set Eb (Eq 28)
      //
      if(i==k)
      {
         sum1=0.0;
         for(j=N;j>=i;j--)
         {
            Gammaj=iterProfj.Current();
            sum1+=rho(j)*Tdia(j)*teleml*pow(sin(Gammaj),3.0)*E(i,j);
            iterProfj++;
         }

         LtK(i+4,k+4)=0.5*tCDcyl*pow(A.Wind,2.0)*teleml*cos(Gammai)*
                  (sum1-0.5*teleml*pow(cos(Gammai),2.0)*rho(i)*Tdia(i)*
                  sin(Gammai))+teleml*cos(Gammai)*Tbar*cos(Gammabar);
      }
      else if(i<k)
      {
         LtK(i+4,k+4)=-0.5*tCDcyl*pow(A.Wind,2.0)*teleml*cos(Gammai)*
                        rho(k)*Tdia(k)*teleml*sin(Gammak)*
                        pow(cos(Gammak),2.0);
      }

      iterProfk++;
   }
   iterProfi++;
}


//Save the K, C, and M matrices to text file for future evaluation
//
if(KCMOut)
{
   TSout.open("LtKCM.txt",ios::out);
   TSout<< setprecision(24);
   TSout << "K matrix\n";
   TSout << LtK;
   TSout << "C matrix\n";
   TSout << LtC;
   TSout << "M matrix\n";
   TSout << LtM;
   TSout.close();
}

//Construct KBar
matrix_mult(2.0/TimeStep,LtC,LtKBar);
matrix_sum(LtKBar,LtK,LtKBar);
matrix_mult(4.0/pow(TimeStep,2.0),LtM,Lttemp1);
matrix_sum(LtKBar,Lttemp1,LtKBar);

//Invert KBar and specify whether the inversion was successful
proceed=1;
if(matrix_inv(LtKBar,LtKBarInv)!=0)
   proceed=0;

//Loop through the time steps
i=1;
while (i<=Steps && proceed)
{
   matrix_zero(LtFBar);

   //Form FBar
   if(YTSSet && YTSDur>=TimeStep*i)
      LtFBar(1)=-YTS;
   if(LTSSet && LTSDur>=TimeStep*i)
      LtFBar(2)=-LTS;
   if(NTSSet && NTSDur>=TimeStep*i)
      LtFBar(3)=-NTS;

   matrix_mult(2.0/TimeStep,LtDel,Lttemp2);
```

```
         matrix_mult(LtC,Lttemp2,Lttemp3);
         matrix_sum(LtFBar,Lttemp3,LtFBar);

         matrix_mult(4.0/pow(TimeStep,2.0),LtDel,Lttemp2);
         matrix_mult(4.0/TimeStep,LtDelDot,Lttemp3);
         matrix_sum(Lttemp2,Lttemp3,Lttemp3);
         matrix_sum(Lttemp3,LtDelDDot,Lttemp3);
         matrix_mult(LtM,Lttemp3,Lttemp2);
         matrix_sum(LtFBar,Lttemp2,LtFBar);           //Gives New FBar

         //Multiply LtKBarInv by FBar to find the Delta Vectors
         matrix_mult(LtKBarInv,LtFBar,LtPDel);        //Gives New Delta
         matrix_mult(TimeStep,LtDelDot,Lttemp2);
         matrix_sum(Lttemp2,LtDel,Lttemp2);
         matrix_mult(-1.0,Lttemp2,Lttemp3);
         matrix_sum(Lttemp3,LtPDel,Lttemp3);
         matrix_mult(2.0/pow(TimeStep,2.0),Lttemp3,Lttemp2);   //Gives DeltaDDotav
         matrix_mult(TimeStep,Lttemp2,Lttemp3);
         matrix_sum(Lttemp3,LtDelDot,LtPDelDot);      //Gives New DeltaDot
         matrix_mult(2.0,Lttemp2,Lttemp3);
         matrix_mult(-1.0,LtDelDDot,Lttemp2);
         matrix_sum(Lttemp3,Lttemp2,LtPDelDDot);   //Gives New DeltaDDot

         //Assign old Delta's to storage array and reset them with the new Delta's
         for(j=1;j<=Nlt;j++)
         {
            LtTS(j,i)=LtDel(j);
         }
         LtDel=LtPDel;
         LtDelDot=LtPDelDot;
         LtDelDDot=LtPDelDDot;

         i++;
      }

      //Save the Delta vectors to text file for future evaluation
      //
      if(TSOut)
      {
         TSout.open("LtTStep.txt",ios::out);
         TSout<< setprecision(16);
         TSout << "\nLtTS\nTime Step = " << TimeStep;
         TSout << "\nTotal Time = " << TotalTime << "\n\n";
         for(j=1;j<=LtTS.n;j++)
         {
            TSout << TimeStep*(j-1) << ",";
            for(i=1;i<=LtTS.m;i++)
            {
               TSout << LtTS(i,j) << ",";
            }
            TSout << "\n";
         }
         TSout.close();
      }
   }

   //If the problem only had one tether element we need to reset N to that
   //value
   if(N<=1)
      N=1;
}
```

```
#if !defined(DIALSPEC_H)
#define DIALSPEC_H

#if !defined(OWL_DIALOG_H)
# include <owl/dialog.h>
#endif

#if !defined(_SUPERKITE_H_)
# include "suprkite.h"
#endif

#if !defined(_ALTITDEF_H_)
# include "altitdef.h"
#endif

class _OWLCLASS TValidator;

#include <owl\listbox.h>
#include <owl\radiobut.h>
#include <owl\groupbox.h>
#include "matrix.h"

//
//  class GenDialog
//  ----- ------------
//
class GenDialog : public TDialog {
  public:

    KiteData* TempKD;

    GenDialog(TWindow*        parent,
              KiteData*       KD,
              TModule*        module = 0,
              TValidator*     valid = 0);

   ~GenDialog();

    //
    // Override TWindow virtual member functions
    //
    void TransferData(TTransferDirection);

  protected:
    //
    // Override TWindow virtual member functions
    //
    void SetupWindow();
    void CmAoaSet();
    void CmAoaSetomax();

  private:
    //
    // hidden to prevent accidental copying or assignment
    //
    GenDialog(const GenDialog&);
    GenDialog& operator=(const GenDialog&);

    TCheckBox *aoaSet;
    TCheckBox *aoaSetomax;
    TEdit *aoaLim;
    TStatic *pretext;
    TStatic *postext;

    DECLARE_RESPONSE_TABLE(GenDialog);
```

```
};


//
//  class WingDialog
//  ----- ------------
//
class WingDialog : public TDialog {
  public:

    KiteData* TempKD;

    WingDialog(TWindow*       parent,
               KiteData*      KD,
               TModule*       module = 0,
               TValidator*    valid = 0);

   ~WingDialog();

    //
    // Override TWindow virtual member functions
    //
    void TransferData(TTransferDirection);


  protected:
    //
    // Override TWindow virtual member functions
    //
    void SetupWindow();

  private:
    //
    // hidden to prevent accidental copying or assignment
    //
    WingDialog(const WingDialog&);
    WingDialog& operator=(const WingDialog&);

};


//
//  class EmpenDialog
//  ----- ------------
//
class EmpenDialog : public TDialog {
  public:

    KiteData* TempKD;

    EmpenDialog(TWindow*         parent,
                KiteData*        KD,
                TModule*         module = 0,
                TValidator*      valid = 0);

   ~EmpenDialog();

    //
    // Override TWindow virtual member functions
    //
    void TransferData(TTransferDirection);


  protected:
    //
    // Override TWindow virtual member functions
    //
  private:
    //
    // hidden to prevent accidental copying or assignment
    //
    EmpenDialog(const EmpenDialog&);
    EmpenDialog& operator=(const EmpenDialog&);

};



//
//  class Empen2Dialog
//  ----- ------------
//
class Empen2Dialog : public TDialog {
  public:

    KiteData* TempKD;

    Empen2Dialog(TWindow*        parent,
                 KiteData*       KD,
                 TModule*        module = 0,
                 TValidator*     valid = 0);

   ~Empen2Dialog();

    //
    // Override TWindow virtual member functions
    //
    void TransferData(TTransferDirection);


  protected:
    //
    // Override TWindow virtual member functions
    //
    void SetupWindow();

  private:
    //
    // hidden to prevent accidental copying or assignment
    //
    Empen2Dialog(const Empen2Dialog&);
    Empen2Dialog& operator=(const Empen2Dialog&);

};



//
//  class FuseDialog
//  ----- ------------
//
class FuseDialog : public TDialog {
  public:

    KiteData* TempKD;

    FuseDialog(TWindow*          parent,
                 KiteData*       KD,
                 TModule*        module = 0,
                 TValidator*     valid = 0);

   ~FuseDialog();
```

```
    //
    // Override TWindow virtual member functions
    //
    void TransferData(TTransferDirection);


  protected:
    //
    // Override TWindow virtual member functions
    //
    void SetupWindow();

  private:
    //
    // hidden to prevent accidental copying or assignment
    //
    FuseDialog(const FuseDialog&);
    FuseDialog& operator=(const FuseDialog&);

};




//
//  class ControlGainDialog
//  ----- -----------------
//
class ControlGainDialog : public TDialog {
  public:

    ControlGain* TempCG;

    ControlGainDialog(TWindow*          parent,
                     ControlGain*      CG,
                     TModule*          module = 0,
                     TValidator*       valid = 0);

   ~ControlGainDialog();

    //
    // Override TWindow virtual member functions
    //
    void TransferData(TTransferDirection);


  protected:
    //
    // Override TWindow virtual member functions
    //
    void SetupWindow();

  private:
    //
    // hidden to prevent accidental copying or assignment
    //
    ControlGainDialog(const ControlGainDialog&);
    ControlGainDialog& operator=(const ControlGainDialog&);

};




//
//  class WingFormDialog
```

```
//
class WindFormDialog : public TDialog {
  public:

    KiteData* TempKD;

    WindFormDialog(TWindow*         parent,
                   KiteData*        KD,
                   TModule*         module = 0,
                   TValidator*      valid = 0);

   ~WindFormDialog();

    //
    // Override TWindow virtual member functions
    //
    void TransferData(TTransferDirection);


  protected:
    //
    // Override TWindow virtual member functions
    //
    void SetupWindow();

  private:
    //
    // hidden to prevent accidental copying or assignment
    //
    WindFormDialog(const WindFormDialog&);
    WindFormDialog& operator=(const WindFormDialog&);

};




//
//  class TrimResultsDialog
//  ----- -----------------
//
class TrimResultsDialog : public TDialog {
  public:

    KiteData* TempKD;
    int Valid;

    TrimResultsDialog(TWindow*         parent,
                 KiteData*        KD,
                 int              isvalid,
                 TModule*         module = 0,
                 TValidator*      valid = 0);

   ~TrimResultsDialog();

    //
    // Override TWindow virtual member functions
    //
    void TransferData(TTransferDirection);


  protected:
    //
    // Override TWindow virtual member functions
    //
    void SetupWindow();
  private:
    //
    // hidden to prevent accidental copying or assignment
    //
    TrimResultsDialog(const TrimResultsDialog&);
    TrimResultsDialog& operator=(const TrimResultsDialog&);

};




//
//  class StabilityResultsDialog
//  ----- ----------------------
//
class StabilityResultsDialog : public TDialog {
  public:

    KiteData* TempKD;
    int N;
    matrixi LgIter,LtIter;

    StabilityResultsDialog(TWindow*           parent,
                KiteData*        KD,
                TModule*         module = 0,
                TValidator*      valid = 0);

   ~StabilityResultsDialog();

    //
    // Override TWindow virtual member functions
    //
    void TransferData(TTransferDirection);


  protected:
    //
    // Override TWindow virtual member functions
    //
    void SetupWindow();
    void CmUpdateLgEignVect();
    void CmUpdateLtEignVect();
    void CmLgStore();
    void CmLtStore();

  private:
    //
    // hidden to prevent accidental copying or assignment
    //
    StabilityResultsDialog(const StabilityResultsDialog&);
    StabilityResultsDialog& operator=(const StabilityResultsDialog&);

    TCheckBox* LgNorm;
    TCheckBox* LtNorm;
    TListBox* LgEignVal;
    TListBox* LgEignVect;
    TListBox* LtEignVal;
    TListBox* LtEignVect;

    DECLARE_RESPONSE_TABLE(StabilityResultsDialog);
};
```

```
//   class StabilityDerivsDialog
//   ----- ---------------------
//
class StabilityDerivsDialog : public TDialog {
  public:

    KiteData* TempKD;

    StabilityDerivsDialog(TWindow*        parent,
                KiteData*        KD,
                TModule*         module = 0,
                TValidator*      valid = 0);

   ~StabilityDerivsDialog();

    //
    // Override TWindow virtual member functions
    //
    void TransferData(TTransferDirection);


  protected:
    //
    // Override TWindow virtual member functions
    //
    void SetupWindow();

  private:
    //
    // hidden to prevent accidental copying or assignment
    //
    StabilityDerivsDialog(const StabilityDerivsDialog&);
    StabilityDerivsDialog& operator=(const StabilityDerivsDialog&);

};




//
//   class QueryVarStudyDialog
//   ----- -------------------
//
class QueryVarStudyDialog : public TDialog {
  public:

    KiteData* TempKD;
    KiteQuery* KQRef;
    KiteQuery* TempKQ;
    int Init;

    QueryVarStudyDialog(TWindow*          parent,
                KiteData*        KD,
                KiteQuery*       KQ,
                TModule*         module = 0,
                TValidator*      valid = 0);

   ~QueryVarStudyDialog();

    //
    // Override TWindow virtual member functions
    //
    void TransferData(TTransferDirection);


  protected:
    // Override TWindow virtual member functions
    //
    void SetupWindow();
    void CmCheckRange();
    void CmUpdate(UINT);
  private:
    //
    // hidden to prevent accidental copying or assignment
    //
    QueryVarStudyDialog(const QueryVarStudyDialog&);
    QueryVarStudyDialog& operator=(const QueryVarStudyDialog&);

    TGroupBox*     LgGroupBox;
    TGroupBox*     LtGroupBox;
    TRadioButton* XGain;
    TRadioButton* ZGain;
    TRadioButton* MGain;
    TRadioButton* YGain;
    TRadioButton* LGain;
    TRadioButton* NGain;
    TRadioButton* VertSpan;
    TRadioButton* VertX;
    TEdit* Lgstart;
    TEdit* Lgend;
    TEdit* Ltstart;
    TEdit* Ltend;
    TEdit* steps;
    TButton* Calculate;

    DECLARE_RESPONSE_TABLE(QueryVarStudyDialog);
};

//
//  class RLScaleDialog
//  ----- -------------
//
class RLScaleDialog : public TDialog {
  public:

    Scale* LG;
    Scale* LT;

    RLScaleDialog(TWindow*         parent,
                  Scale*           LGScale,
                  Scale*           LTScale,
                  TModule*         module = 0,
                  TValidator*      valid = 0);

   ~RLScaleDialog();

    //
    // Override TWindow virtual member functions
    //
    void TransferData(TTransferDirection);

  protected:
    //
    // Override TWindow virtual member functions
    //
    void SetupWindow();
    void CmUpdateTxt();

  private:
    //
    // hidden to prevent accidental copying or assignment
    //
    RLScaleDialog& operator=(const RLScaleDialog&);

    TCheckBox *ymaxlgSet;
    TCheckBox *xmaxlgSet;
    TCheckBox *xminlgSet;
    TCheckBox *ymaxltSet;
    TCheckBox *xmaxltSet;
    TCheckBox *xminltSet;
    TEdit *ymaxlg;
    TEdit *xmaxlg;
    TEdit *xminlg;
    TEdit *ymaxlt;
    TEdit *xmaxlt;
    TEdit *xminlt;
    TStatic *ymaxlgtxt;
    TStatic *xmaxlgtxt;
    TStatic *xminlgtxt;
    TStatic *ymaxlttxt;
    TStatic *xmaxlttxt;
    TStatic *xminlttxt;

    DECLARE_RESPONSE_TABLE(RLScaleDialog);
};

//
//  class TimeStepDialog
//  ----- --------------
//
class TimeStepDialog : public TDialog {
  public:

    KiteData* TempKD;

    TimeStepDialog(TWindow*        parent,
                   KiteData*       KD,
                   TModule*        module = 0,
                   TValidator*     valid = 0);

   ~TimeStepDialog();

    //
    // Override TWindow virtual member functions
    //
    void TransferData(TTransferDirection);

  protected:
    //
    // Override TWindow virtual member functions
    //
    void SetupWindow();
    void CmUpdateTxt();

  private:
    //
    // hidden to prevent accidental copying or assignment
    //
    TimeStepDialog(const TimeStepDialog&);
    TimeStepDialog& operator=(const TimeStepDialog&);

    TCheckBox *xtsSet;
    TCheckBox *ztsSet;
    TCheckBox *mtsSet;
    TCheckBox *ytsSet;
    TCheckBox *ltsSet;
    TCheckBox *ntsSet;
    TEdit *timestep;
    TEdit *xts;
    TEdit *zts;
    TEdit *mts;
    TEdit *xtsdur;
    TEdit *ztsdur;
    TEdit *mtsdur;
    TStatic *xtstxt1;
    TStatic *ztstxt1;
    TStatic *mtstxt1;
    TStatic *xtstxt2;
    TStatic *ztstxt2;
    TStatic *mtstxt2;
    TStatic *xtstxt3;
    TStatic *ztstxt3;
    TStatic *mtstxt3;
    TStatic *xtstxt4;
    TStatic *ztstxt4;
    TStatic *mtstxt4;
    TEdit *yts;
    TEdit *lts;
    TEdit *nts;
    TEdit *ytsdur;
    TEdit *ltsdur;
    TEdit *ntsdur;
    TStatic *ytstxt1;
    TStatic *ltstxt1;
    TStatic *ntstxt1;
    TStatic *ytstxt2;
    TStatic *ltstxt2;
    TStatic *ntstxt2;
    TStatic *ytstxt3;
    TStatic *ltstxt3;
    TStatic *ntstxt3;
    TStatic *ytstxt4;
    TStatic *ltstxt4;
    TStatic *ntstxt4;
    TButton* Calculate;

    DECLARE_RESPONSE_TABLE(TimeStepDialog);
};

//
//  class RLScaleDialog
//  ----- -------------
//
class TSScaleDialog : public TDialog {
  public:

    Scale* LG;
    Scale* LT;

    TSScaleDialog(TWindow*        parent,
                  Scale*          LGScale,
                  Scale*          LTScale,
                  TModule*        module = 0,
                  TValidator*     valid = 0);

   ~TSScaleDialog();

    //
    // Override TWindow virtual member functions
    //
    void TransferData(TTransferDirection);

  protected:
    //
    //
    void SetupWindow();
    void CmUpdateTxt();

  private:
    //
    // hidden to prevent accidental copying or assignment
    //
    TSScaleDialog(const TSScaleDialog&);
    TSScaleDialog& operator=(const TSScaleDialog&);

    TCheckBox *ymaxlgSet;
    TCheckBox *yminlgSet;
    TCheckBox *ymaxltSet;
    TCheckBox *yminltSet;
    TEdit *ymaxlg;
    TEdit *yminlg;
    TEdit *ymaxlt;
    TEdit *yminlt;
    TStatic *ymaxlgtxt;
    TStatic *yminlgtxt;
    TStatic *ymaxlttxt;
    TStatic *yminlttxt;

    DECLARE_RESPONSE_TABLE(TSScaleDialog);
};


#endif
```

```
#include <string.h>
#include <fstream.h>
#include <iomanip.h>
#include <owl/edit.h>
#include <owl/checkbox.h>
#include <owl/validate.h>
#include "dialspec.h"
#include "dialspec.rh"

char gentitle[50]="Super Kite General Specifications";
char wingtitle[50]="Super Kite Wing Specifications";
char empentitle[50]="Super Kite Primary Empennage Specifications";
char empen2title[50]="Super Kite Secondary Empennage Specifications";
char fusetitle[50]="Super Kite Fuselage Specifications";
char controlgaintitle[50]="Super Kite Control Gain Specifications";
char TrimResultstitle[50]="Super Kite Trim State - Results";
char StabilityResultstitle[50]="Super Kite Stability Analysis - Results";
char windformtitle[50]="Wind Velocity Function Specification";
const int sig = 15;


//************************************************************

DEFINE_RESPONSE_TABLE1(GenDialog, TDialog)
  EV_COMMAND(ID_AOASET, CmAoaSet),
  EV_COMMAND(ID_AOASETOMAX, CmAoaSetomax),
END_RESPONSE_TABLE;

GenDialog::GenDialog(TWindow*        parent,
                     KiteData*       KD,
                     TModule*        module,
                     TValidator*     validator)
:
  TWindow(parent, gentitle, module),
  TDialog(parent, IDD_GENDIALOG, module)
{
  TempKD = KD;
  SetCaption(gentitle);
  if (validator)
  {
    new TEdit(this,ID_MASS)->SetValidator(validator);
    new TEdit(this,ID_EFF)->SetValidator(validator);
    new TEdit(this,ID_XCG)->SetValidator(validator);
    new TEdit(this,ID_ZCG)->SetValidator(validator);
    new TEdit(this,ID_XCPLG)->SetValidator(validator);
    new TEdit(this,ID_ZCPLG)->SetValidator(validator);
    new TEdit(this,ID_XCPLT)->SetValidator(validator);
    new TEdit(this,ID_ZCPLT)->SetValidator(validator);
    new TEdit(this,ID_IXX)->SetValidator(validator);
    new TEdit(this,ID_IYY)->SetValidator(validator);
    new TEdit(this,ID_IZZ)->SetValidator(validator);
    new TEdit(this,ID_IXZ)->SetValidator(validator);
    new TEdit(this,ID_TDENS)->SetValidator(validator);
    new TEdit(this,ID_TETHE)->SetValidator(validator);
    new TEdit(this,ID_TDIA)->SetValidator(validator);
    new TEdit(this,ID_TDIAO)->SetValidator(validator);
    new TEdit(this,ID_TELEML)->SetValidator(validator);
    new TEdit(this,ID_TCDCYL)->SetValidator(validator);
    new TEdit(this,ID_TCDAX)->SetValidator(validator);
    new TEdit(this,ID_TGLLIM)->SetValidator(validator);
    new TEdit(this,ID_TNC)->SetValidator(validator);
  }
  aoaSet = new TCheckBox(this, ID_AOASET);
  aoaSetomax = new TCheckBox(this, ID_AOASETOMAX);
  pretext = new TStatic(this,ID_PRETEXT);
  [illegible]
  postext = new TStatic(this,ID_POSTEXT);
}

GenDialog::~GenDialog()
{
   delete aoaSet;
   delete aoaSetomax;
   delete pretext;
   delete aoaLim;
   delete postext;
}

void
GenDialog::CmAoaSet()
{
   if (aoaSet->GetCheck() == BF_CHECKED)
   {
      aoaSetomax->EnableWindow(1);
      if(aoaSetomax->GetCheck() == BF_CHECKED)
      {
         pretext->EnableWindow(0);
         aoaLim->EnableWindow(0);
         postext->EnableWindow(0);
      }
      else
      {
         pretext->EnableWindow(1);
         aoaLim->EnableWindow(1);
         postext->EnableWindow(1);
      }
   }
   else
   {
      aoaSetomax->EnableWindow(0);
      pretext->EnableWindow(0);
      aoaLim->EnableWindow(0);
      postext->EnableWindow(0);
   }
}

void
GenDialog::CmAoaSetomax()
{
   if (aoaSet->GetCheck() == BF_CHECKED&&
               aoaSetomax->GetCheck() == BF_UNCHECKED )
   {
      pretext->EnableWindow(1);
      aoaLim->EnableWindow(1);
      postext->EnableWindow(1);
   }
   else
   {
      pretext->EnableWindow(0);
      aoaLim->EnableWindow(0);
      postext->EnableWindow(0);
   }
}

//
// sets and gets the values of the items (controls) of the input dialog
//
void
GenDialog::TransferData(TTransferDirection direction)
{
   char  Buffer[25] = "";
```

```
if (direction == tdSetData) {
   gcvt(TempKD->mass,sig,Buffer);
   SetDlgItemText(ID_MASS, Buffer);
   gcvt(TempKD->e,sig,Buffer);
   SetDlgItemText(ID_EFF, Buffer);
   gcvt(TempKD->Xcg,sig,Buffer);
   SetDlgItemText(ID_XCG, Buffer);
   gcvt(TempKD->Zcg,sig,Buffer);
   SetDlgItemText(ID_ZCG, Buffer);
   gcvt(TempKD->Xcplg,sig,Buffer);
   SetDlgItemText(ID_XCPLG, Buffer);
   gcvt(TempKD->Zcplg,sig,Buffer);
   SetDlgItemText(ID_ZCPLG, Buffer);
   gcvt(TempKD->Xcplt,sig,Buffer);
   SetDlgItemText(ID_XCPLT, Buffer);
   gcvt(TempKD->Zcplt,sig,Buffer);
   SetDlgItemText(ID_ZCPLT, Buffer);
   gcvt(TempKD->Ixx,sig,Buffer);
   SetDlgItemText(ID_IXX, Buffer);
   gcvt(TempKD->Iyy,sig,Buffer);
   SetDlgItemText(ID_IYY, Buffer);
   gcvt(TempKD->Izz,sig,Buffer);
   SetDlgItemText(ID_IZZ, Buffer);
   gcvt(TempKD->Ixz,sig,Buffer);
   SetDlgItemText(ID_IXZ, Buffer);
   gcvt(TempKD->tdens,sig,Buffer);
   SetDlgItemText(ID_TDENS, Buffer);
   gcvt(TempKD->tethE/1.0e9,sig,Buffer);
   SetDlgItemText(ID_TETHE, Buffer);
   gcvt(TempKD->tdia,sig,Buffer);
   SetDlgItemText(ID_TDIA, Buffer);
   gcvt(TempKD->tdiao,sig,Buffer);
   SetDlgItemText(ID_TDIAO, Buffer);
   gcvt(TempKD->teleml,sig,Buffer);
   SetDlgItemText(ID_TELEML, Buffer);
   gcvt(TempKD->tCDcyl,sig,Buffer);
   SetDlgItemText(ID_TCDCYL, Buffer);
   gcvt(TempKD->tCDax,sig,Buffer);
   SetDlgItemText(ID_TCDAX, Buffer);
   gcvt(TempKD->tGammallim,sig,Buffer);
   SetDlgItemText(ID_TGLLIM, Buffer);
   gcvt(TempKD->Nc,sig,Buffer);
   SetDlgItemText(ID_TNC, Buffer);
   gcvt(TempKD->aoalim,sig,Buffer);
   SetDlgItemText(ID_AOALIM, Buffer);
   if(TempKD->aoaset)
      aoaSet->SetCheck(BF_CHECKED);
   if(TempKD->aoasetomax)
      aoaSetomax->SetCheck(BF_CHECKED);
}
else if (direction == tdGetData) {
   GetDlgItemText(ID_MASS, Buffer, sig);
   tempdbl = atof(Buffer);
   if(tempdbl>0.0) TempKD->mass=tempdbl;
   GetDlgItemText(ID_EFF, Buffer, sig);
   tempdbl = atof(Buffer);
   if(tempdbl>0.0&&tempdbl<=1.0) TempKD->e=tempdbl;
   GetDlgItemText(ID_XCG, Buffer, sig);
   tempdbl = atof(Buffer);
   TempKD->Xcg=tempdbl;
   GetDlgItemText(ID_ZCG, Buffer, sig);
   tempdbl = atof(Buffer);
   TempKD->Zcg=tempdbl;
   GetDlgItemText(ID_XCPLG, Buffer, sig);
   tempdbl = atof(Buffer);
   TempKD->Xcplg=tempdbl;
```

```
        tempdbl = atof(Buffer);
        TempKD->Zcplg=tempdbl;
        GetDlgItemText(ID_XCPLT, Buffer, sig);
        tempdbl = atof(Buffer):
        TempKD->Xcplt=tempdbl;
        GetDlgItemText(ID_ZCPLT, Buffer, sig);
        tempdbl = atof(Buffer);
        TempKD->Zcplt=tempdbl;
        GetDlgItemText(ID_IXX, Buffer, sig);
        tempdbl = atof(Buffer);
        if(tempdbl>0.0) TempKD->Ixx=tempdbl;
        GetDlgItemText(ID_IYY, Buffer, sig);
        tempdbl = atof(Buffer);
        if(tempdbl>0.0) TempKD->Iyy=tempdbl;
        GetDlgItemText(ID_IZZ, Buffer, sig);
        tempdbl = atof(Buffer);
        if(tempdbl>0.0) TempKD->Izz=tempdbl;
        GetDlgItemText(ID_IXZ, Buffer, sig);
        tempdbl = atof(Buffer);
        if(tempdbl>0.0) TempKD->Ixz=tempdbl;
        GetDlgItemText(ID_TDENS, Buffer, sig);
        tempdbl = atof(Buffer);
        if(tempdbl>0.0) TempKD->tdens=tempdbl;
        GetDlgItemText(ID_TETHE, Buffer, sig);
        tempdbl = atof(Buffer)*1.0e9;
        if(tempdbl>0.0) TempKD->tethE=tempdbl;
        GetDlgItemText(ID_TDIA, Buffer, sig);
        tempdbl = atof(Buffer);
        if(tempdbl>0.0) TempKD->tdia=tempdbl;
        GetDlgItemText(ID_TDIAO, Buffer, sig);
        tempdbl = atof(Buffer);
        if(tempdbl>0.0) TempKD->tdiao=tempdbl;
        GetDlgItemText(ID_TELEML, Buffer, sig);
        tempdbl = atof(Buffer);
        if(tempdbl>0.0) TempKD->teleml=tempdbl;
        GetDlgItemText(ID_TCDCYL, Buffer, sig);
        tempdbl = atof(Buffer);
        if(tempdbl>0.0) TempKD->tCDcyl=tempdbl;
        GetDlgItemText(ID_TCDAX, Buffer, sig);
        tempdbl = atof(Buffer);
        if(tempdbl>0.0) TempKD->tCDax=tempdbl;
        GetDlgItemText(ID_TGLLIM, Buffer, sig);
        tempdbl = atof(Buffer);
        if(tempdbl>=0.0) TempKD->tGammallim=tempdbl;
        GetDlgItemText(ID_TNC, Buffer, sig);
        TempKD->Nc=atof(Buffer);
//      tempdbl = atof(Buffer);
//      if(tempdbl>0.0) TempKD->Nc=tempdbl;
        GetDlgItemText(ID_AOALIM, Buffer, sig);
        tempdbl = atof(Buffer);
        if(fabs(tempdbl)<=20.0) TempKD->aoalim=tempdbl;
        if(aoaSet->GetCheck()==BF_CHECKED)
           TempKD->aoaset=1;
        else
           TempKD->aoaset=0;
        if(aoaSetomax->GetCheck()==BF_CHECKED)
           TempKD->aoasetomax=1;
        else
           TempKD->aoasetomax=0;
    }
}

//
// sets the values of the items(controls) of the input dialog
//
void
```

```
{
  TDialog::SetupWindow();
  SendDlgItemMessage(ID_MASS, EM_LIMITTEXT, sig - 1, 0);
  SendDlgItemMessage(ID_EFF, EM_LIMITTEXT, sig - 1, 0);
  SendDlgItemMessage(ID_XCG, EM_LIMITTEXT, sig - 1, 0);
  SendDlgItemMessage(ID_ZCG, EM_LIMITTEXT, sig - 1, 0);
  SendDlgItemMessage(ID_XCPLG, EM_LIMITTEXT, sig - 1, 0);
  SendDlgItemMessage(ID_ZCPLG, EM_LIMITTEXT, sig - 1, 0);
  SendDlgItemMessage(ID_XCPLT, EM_LIMITTEXT, sig - 1, 0);
  SendDlgItemMessage(ID_ZCPLT, EM_LIMITTEXT, sig - 1, 0);
  SendDlgItemMessage(ID_IXX, EM_LIMITTEXT, sig - 1, 0);
  SendDlgItemMessage(ID_IYY, EM_LIMITTEXT, sig - 1, 0);
  SendDlgItemMessage(ID_IZZ, EM_LIMITTEXT, sig - 1, 0);
  SendDlgItemMessage(ID_IXZ, EM_LIMITTEXT, sig - 1, 0);
  SendDlgItemMessage(ID_TDENS, EM_LIMITTEXT, sig - 1, 0);
  SendDlgItemMessage(ID_TETHE, EM_LIMITTEXT, sig - 1, 0);
  SendDlgItemMessage(ID_TDIA, EM_LIMITTEXT, sig - 1, 0);
  SendDlgItemMessage(ID_TDIAO, EM_LIMITTEXT, sig - 1, 0);
  SendDlgItemMessage(ID_TELEML, EM_LIMITTEXT, sig - 1, 0);
  SendDlgItemMessage(ID_TCDCYL, EM_LIMITTEXT, sig - 1, 0);
  SendDlgItemMessage(ID_TCDAX, EM_LIMITTEXT, sig - 1, 0);
  SendDlgItemMessage(ID_TGLLIM, EM_LIMITTEXT, sig - 1, 0);
  SendDlgItemMessage(ID_TNC, EM_LIMITTEXT, sig - 1, 0);
  SendDlgItemMessage(ID_AOALIM, EM_LIMITTEXT, sig - 1, 0);
  CmAoaSet();
  CmAoaSetomax();
}


//**************************************************************

WingDialog::WingDialog(TWindow*        parent,
                       KiteData*       KD,
                       TModule*        module,
                       TValidator*     validator)
:
  TWindow(parent, wingtitle, module),
  TDialog(parent, IDD_WINGDIALOG, module)
{
  TempKD = KD;

  SetCaption(wingtitle);
  if (validator){
    new TEdit(this,ID_SPAN)->SetValidator(validator);
    new TEdit(this,ID_CROOT)->SetValidator(validator);
    new TEdit(this,ID_CTIP)->SetValidator(validator);
    new TEdit(this,ID_AWING)->SetValidator(validator);
    new TEdit(this,ID_CLMAX)->SetValidator(validator);
    new TEdit(this,ID_CMAC)->SetValidator(validator);
    new TEdit(this,ID_THICKW)->SetValidator(validator);
    new TEdit(this,ID_DIH)->SetValidator(validator);
    new TEdit(this,ID_LESWP)->SetValidator(validator);
    new TEdit(this,ID_QCSWP)->SetValidator(validator);
    new TEdit(this,ID_IWING)->SetValidator(validator);
    new TEdit(this,ID_AZLLW)->SetValidator(validator);
    new TEdit(this,ID_XAC)->SetValidator(validator);
    new TEdit(this,ID_ZAC)->SetValidator(validator);
    new TEdit(this,ID_HACWB)->SetValidator(validator);
    new TEdit(this,ID_EO)->SetValidator(validator);
    new TEdit(this,ID_SIGLSP)->SetValidator(validator);
    new TEdit(this,ID_CLPP)->SetValidator(validator);
    new TEdit(this,ID_SIGLSR)->SetValidator(validator);
  }
}

WingDialog::~WingDialog()
```

```
}

//
// sets and gets the values of the items (controls) of the input dialog
//
void
WingDialog::TransferData(TTransferDirection direction)
{
   char  Buffer[25] = "";
   double tempdbl;
   if (direction == tdSetData)
   {
      gcvt(TempKD->b,sig,Buffer);
      SetDlgItemText(ID_SPAN, Buffer);
      gcvt(TempKD->croot,sig,Buffer);
      SetDlgItemText(ID_CROOT, Buffer);
      gcvt(TempKD->ctip,sig,Buffer);
      SetDlgItemText(ID_CTIP, Buffer);
      gcvt(TempKD->awing,sig,Buffer);
      SetDlgItemText(ID_AWING, Buffer);
      gcvt(TempKD->Clmax,sig,Buffer);
      SetDlgItemText(ID_CLMAX, Buffer);
      gcvt(TempKD->Cmac,sig,Buffer);
      SetDlgItemText(ID_CMAC, Buffer);
      gcvt(TempKD->Thick,sig,Buffer);
      SetDlgItemText(ID_THICKW, Buffer);
      gcvt(TempKD->Dih,sig,Buffer);
      SetDlgItemText(ID_DIH, Buffer);
      gcvt(TempKD->LEswp,sig,Buffer);
      SetDlgItemText(ID_LESWP, Buffer);
      gcvt(TempKD->QCswp,sig,Buffer);
      SetDlgItemText(ID_QCSWP, Buffer);
      gcvt(TempKD->iwing,sig,Buffer);
      SetDlgItemText(ID_IWING, Buffer);
      gcvt(TempKD->azll,sig,Buffer);
      SetDlgItemText(ID_AZLLW, Buffer);
      gcvt(TempKD->Xac,sig,Buffer);
      SetDlgItemText(ID_XAC, Buffer);
      gcvt(TempKD->Zac,sig,Buffer);
      SetDlgItemText(ID_ZAC, Buffer);
      gcvt(TempKD->hacwb,sig,Buffer);
      SetDlgItemText(ID_HACWB, Buffer);
      gcvt(TempKD->Eo,sig,Buffer);
      SetDlgItemText(ID_EO, Buffer);
      gcvt(TempKD->dSiglsdP,sig,Buffer);
      SetDlgItemText(ID_SIGLSP, Buffer);
      gcvt(TempKD->CLp_plan,sig,Buffer);
      SetDlgItemText(ID_CLPP, Buffer);
      gcvt(TempKD->dSiglsdr,sig,Buffer);
      SetDlgItemText(ID_SIGLSR, Buffer);
   }
   else if (direction == tdGetData)
   {
      GetDlgItemText(ID_SPAN, Buffer, sig);
      tempdbl = atof(Buffer);
      if(tempdbl>0.0) TempKD->b = tempdbl;
      GetDlgItemText(ID_CROOT, Buffer, sig);
      tempdbl = atof(Buffer);
      if(tempdbl>0.0) TempKD->croot = tempdbl;
      GetDlgItemText(ID_CTIP, Buffer, sig);
      tempdbl = atof(Buffer);
      if(tempdbl>0.0) TempKD->ctip = tempdbl;
      GetDlgItemText(ID_AWING, Buffer, sig);
      tempdbl = atof(Buffer);
      if(tempdbl>0.0&&tempdbl<6.29) TempKD->awing = tempdbl;
```

```
     GetDlgItemText(ID_CLMAX, Buffer, sig);
     tempdbl = atof(Buffer);
     if(tempdbl>0.0) TempKD->Clmax = tempdbl;
     GetDlgItemText(ID_CMAC, Buffer, sig);
     tempdbl = atof(Buffer);
     TempKD->Cmac = tempdbl;
     GetDlgItemText(ID_THICKW, Buffer, sig);
     tempdbl = atof(Buffer);
     if(tempdbl>0.0&&tempdbl<40.0) TempKD->Thick = tempdbl;
     GetDlgItemText(ID_DIH, Buffer, sig);
     tempdbl = atof(Buffer);
     TempKD->Dih = tempdbl;
     GetDlgItemText(ID_LESWP, Buffer, sig);
     tempdbl = atof(Buffer);
     TempKD->LEswp = tempdbl;
     GetDlgItemText(ID_QCSWP, Buffer, sig);
     tempdbl = atof(Buffer);
     TempKD->QCswp = tempdbl;
     GetDlgItemText(ID_IWING, Buffer, sig);
     tempdbl = atof(Buffer);
     TempKD->iwing = tempdbl;
     GetDlgItemText(ID_AZLLW, Buffer, sig);
     tempdbl = atof(Buffer);
     TempKD->azll = tempdbl;
     GetDlgItemText(ID_XAC, Buffer, sig);
     tempdbl = atof(Buffer);
     TempKD->Xac = tempdbl;
     GetDlgItemText(ID_ZAC, Buffer, sig);
     tempdbl = atof(Buffer);
     TempKD->Zac = tempdbl;
     GetDlgItemText(ID_HACWB, Buffer, sig);
     tempdbl = atof(Buffer);
     if(tempdbl>0.0&&tempdbl<1.00) TempKD->hacwb = tempdbl;
     GetDlgItemText(ID_EO, Buffer, sig);
     tempdbl = atof(Buffer);
     TempKD->Eo = tempdbl;
     GetDlgItemText(ID_SIGLSP, Buffer, sig);
     tempdbl = atof(Buffer);
     TempKD->dSiglsdP = tempdbl;
     GetDlgItemText(ID_CLPP, Buffer, sig);
     tempdbl = atof(Buffer);
     TempKD->CLp_plan = tempdbl;
     GetDlgItemText(ID_SIGLSR, Buffer, sig);
     tempdbl = atof(Buffer);
     TempKD->dSiglsdr = tempdbl;
   }
}

//
// sets the values of the items(controls) of the input dialog
//
void
WingDialog::SetupWindow()
{
  TDialog::SetupWindow();
  SendDlgItemMessage(ID_SPAN, EM_LIMITTEXT, sig - 1, 0);
  SendDlgItemMessage(ID_CROOT, EM_LIMITTEXT, sig - 1, 0);
  SendDlgItemMessage(ID_CTIP, EM_LIMITTEXT, sig - 1, 0);
  SendDlgItemMessage(ID_AWING, EM_LIMITTEXT, sig - 1, 0);
  SendDlgItemMessage(ID_CLMAX, EM_LIMITTEXT, sig - 1, 0);
  SendDlgItemMessage(ID_CMAC, EM_LIMITTEXT, sig - 1, 0);
  SendDlgItemMessage(ID_THICKW, EM_LIMITTEXT, sig - 1, 0);
  SendDlgItemMessage(ID_DIH, EM_LIMITTEXT, sig - 1, 0);
  SendDlgItemMessage(ID_LESWP, EM_LIMITTEXT, sig - 1, 0);
  SendDlgItemMessage(ID_QCSWP, EM_LIMITTEXT, sig - 1, 0);
  SendDlgItemMessage(ID_IWING, EM_LIMITTEXT, sig - 1, 0);
  SendDlgItemMessage(ID_AZLLW, EM_LIMITTEXT, sig - 1, 0);
  SendDlgItemMessage(ID_ZAC, EM_LIMITTEXT, sig - 1, 0);
  SendDlgItemMessage(ID_HACWB, EM_LIMITTEXT, sig - 1, 0);
  SendDlgItemMessage(ID_EO, EM_LIMITTEXT, sig - 1, 0);
  SendDlgItemMessage(ID_SIGLSP, EM_LIMITTEXT, sig - 1, 0);
  SendDlgItemMessage(ID_CLPP, EM_LIMITTEXT, sig - 1, 0);
  SendDlgItemMessage(ID_SIGLSR, EM_LIMITTEXT, sig - 1, 0);
}


//***************************************************************

EmpenDialog::EmpenDialog(TWindow*        parent,
                        KiteData*       KD,
                        TModule*        module,
                        TValidator*     validator)
:
  TWindow(parent, empentitle, module),
  TDialog(parent, IDD_EMPENDIALOG, module)
{
  TempKD = KD;

  SetCaption(empentitle);
  if (validator){
    new TEdit(this,ID_HTSPAN)->SetValidator(validator);
    new TEdit(this,ID_CHT)->SetValidator(validator);
    new TEdit(this,ID_AHT)->SetValidator(validator);
    new TEdit(this,ID_THICKHT)->SetValidator(validator);
    new TEdit(this,ID_XHT)->SetValidator(validator);
    new TEdit(this,ID_ZHT)->SetValidator(validator);
    new TEdit(this,ID_IT)->SetValidator(validator);
    new TEdit(this,ID_AZLLHT)->SetValidator(validator);
    new TEdit(this,ID_NETAHT)->SetValidator(validator);
    new TEdit(this,ID_VTSPAN)->SetValidator(validator);
    new TEdit(this,ID_CVT)->SetValidator(validator);
    new TEdit(this,ID_AVT)->SetValidator(validator);
    new TEdit(this,ID_THICKVT)->SetValidator(validator);
    new TEdit(this,ID_XVT)->SetValidator(validator);
    new TEdit(this,ID_ZVT)->SetValidator(validator);
    new TEdit(this,ID_NETAVT)->SetValidator(validator);
    new TEdit(this,ID_SIGVTP)->SetValidator(validator);
    new TEdit(this,ID_SIGVTR)->SetValidator(validator);
    new TEdit(this,ID_SIGVTB)->SetValidator(validator);
  }
}

EmpenDialog::~EmpenDialog()
{

}

//
// sets and gets the values of the items (controls) of the input dialog
//
void
EmpenDialog::TransferData(TTransferDirection direction)
{
   char  Buffer[25] = "";
   double tempdbl;
   if (direction == tdSetData) {
      gcvt(TempKD->bht,sig,Buffer);
      SetDlgItemText(ID_HTSPAN, Buffer);
      gcvt(TempKD->cht,sig,Buffer);
      SetDlgItemText(ID_CHT, Buffer);
      gcvt(TempKD->aht,sig,Buffer);
      SetDlgItemText(ID_AHT, Buffer);
      gcvt(TempKD->Thickht,sig,Buffer);
   gcvt(TempKD->Xht,sig,Buffer);
   SetDlgItemText(ID_XHT, Buffer);
   gcvt(TempKD->Zht,sig,Buffer);
   SetDlgItemText(ID_ZHT, Buffer);
   gcvt(TempKD->iht,sig,Buffer);
   SetDlgItemText(ID_IT, Buffer);
   gcvt(TempKD->azllht,sig,Buffer);
   SetDlgItemText(ID_AZLLHT, Buffer);
   gcvt(TempKD->Netaht,sig,Buffer);
   SetDlgItemText(ID_NETAHT, Buffer);
   gcvt(TempKD->bvt,sig,Buffer);
   SetDlgItemText(ID_VTSPAN, Buffer);
   gcvt(TempKD->cvt,sig,Buffer);
   SetDlgItemText(ID_CVT, Buffer);
   gcvt(TempKD->avt,sig,Buffer);
   SetDlgItemText(ID_AVT, Buffer);
   gcvt(TempKD->Thickvt,sig,Buffer);
   SetDlgItemText(ID_THICKVT, Buffer);
   gcvt(TempKD->Xvt,sig,Buffer);
   SetDlgItemText(ID_XVT, Buffer);
   gcvt(TempKD->Zvt,sig,Buffer);
   SetDlgItemText(ID_ZVT, Buffer);
   gcvt(TempKD->Netavt,sig,Buffer);
   SetDlgItemText(ID_NETAVT, Buffer);
   gcvt(TempKD->dSigfdP,sig,Buffer);
   SetDlgItemText(ID_SIGVTP, Buffer);
   gcvt(TempKD->dSigfdr,sig,Buffer);
   SetDlgItemText(ID_SIGVTR, Buffer);
   gcvt(TempKD->dSigmadB,sig,Buffer);
   SetDlgItemText(ID_SIGVTB, Buffer);
}
else if (direction == tdGetData) {
   GetDlgItemText(ID_HTSPAN, Buffer, sig);
   tempdbl = atof(Buffer);
   if(tempdbl>=0.0) TempKD->bht = tempdbl;
   GetDlgItemText(ID_CHT, Buffer, sig);
   tempdbl = atof(Buffer);
   if(tempdbl>0.0) TempKD->cht = tempdbl;
   GetDlgItemText(ID_AHT, Buffer, sig);
   tempdbl = atof(Buffer);
   if(tempdbl>0.0&&tempdbl<6.29) TempKD->aht = tempdbl;
   GetDlgItemText(ID_THICKHT, Buffer, sig);
   tempdbl = atof(Buffer);
   if(tempdbl>0.0&&tempdbl<40.0) TempKD->Thickht = tempdbl;
   GetDlgItemText(ID_XHT, Buffer, sig);
   tempdbl = atof(Buffer);
   TempKD->Xht = tempdbl;
   GetDlgItemText(ID_ZHT, Buffer, sig);
   tempdbl = atof(Buffer);
   TempKD->Zht = tempdbl;
   GetDlgItemText(ID_IT, Buffer, sig);
   tempdbl = atof(Buffer);
   TempKD->iht = tempdbl;
   GetDlgItemText(ID_AZLLHT, Buffer, sig);
   tempdbl = atof(Buffer);
   TempKD->azllht = tempdbl;
   GetDlgItemText(ID_NETAHT, Buffer, sig);
   tempdbl = atof(Buffer);
   if(tempdbl>0.0&&tempdbl<=1.0) TempKD->Netaht = tempdbl;
   GetDlgItemText(ID_VTSPAN, Buffer, sig);
   tempdbl = atof(Buffer);
   if(tempdbl>=0.0) TempKD->bvt = tempdbl;
   GetDlgItemText(ID_CVT, Buffer, sig);
   tempdbl = atof(Buffer);
   if(tempdbl>0.0) TempKD->cvt = tempdbl;
   GetDlgItemText(ID_AVT, Buffer, sig);
```

```
      if(tempdbl>0.0&&tempdbl<0.25) TempKD->Avt = tempdbl;
      GetDlgItemText(ID_THICKVT, Buffer, sig);
      tempdbl = atof(Buffer);
      if(tempdbl>0.0&&tempdbl<40.0) TempKD->Thickvt = tempdbl;
      GetDlgItemText(ID_XVT, Buffer, sig);
      tempdbl = atof(Buffer);
      TempKD->Xvt = tempdbl;
      GetDlgItemText(ID_ZVT, Buffer, sig);
      tempdbl = atof(Buffer);
      TempKD->Zvt = tempdbl;
      GetDlgItemText(ID_NETAVT, Buffer, sig);
      tempdbl = atof(Buffer);
      if(tempdbl>0.0&&tempdbl<=1.0) TempKD->Netavt = tempdbl;
      GetDlgItemText(ID_SIGVTP, Buffer, sig);
      tempdbl = atof(Buffer);
      TempKD->dSigfdP = tempdbl;
      GetDlgItemText(ID_SIGVTR, Buffer, sig);
      tempdbl = atof(Buffer);
      TempKD->dSigfdr = tempdbl;
      GetDlgItemText(ID_SIGVTB, Buffer, sig);
      tempdbl = atof(Buffer);
      TempKD->dSigmadB = tempdbl;
   }
}

//
// sets the values of the items(controls) of the input dialog
//
void
EmpenDialog::SetupWindow()
{
  TDialog::SetupWindow();
  SendDlgItemMessage(ID_HTSPAN, EM_LIMITTEXT, sig - 1, 0);
  SendDlgItemMessage(ID_CHT, EM_LIMITTEXT, sig - 1, 0);
  SendDlgItemMessage(ID_AHT, EM_LIMITTEXT, sig - 1, 0);
  SendDlgItemMessage(ID_THICKHT, EM_LIMITTEXT, sig - 1, 0);
  SendDlgItemMessage(ID_XHT, EM_LIMITTEXT, sig - 1, 0);
  SendDlgItemMessage(ID_ZHT, EM_LIMITTEXT, sig - 1, 0);
  SendDlgItemMessage(ID_IT, EM_LIMITTEXT, sig - 1, 0);
  SendDlgItemMessage(ID_AZLLHT, EM_LIMITTEXT, sig - 1, 0);
  SendDlgItemMessage(ID_NETAHT, EM_LIMITTEXT, sig - 1, 0);
  SendDlgItemMessage(ID_VTSPAN, EM_LIMITTEXT, sig - 1, 0);
  SendDlgItemMessage(ID_CVT, EM_LIMITTEXT, sig - 1, 0);
  SendDlgItemMessage(ID_AVT, EM_LIMITTEXT, sig - 1, 0);
  SendDlgItemMessage(ID_THICKVT, EM_LIMITTEXT, sig - 1, 0);
  SendDlgItemMessage(ID_XVT, EM_LIMITTEXT, sig - 1, 0);
  SendDlgItemMessage(ID_ZVT, EM_LIMITTEXT, sig - 1, 0);
  SendDlgItemMessage(ID_NETAVT, EM_LIMITTEXT, sig - 1, 0);
  SendDlgItemMessage(ID_SIGVTP, EM_LIMITTEXT, sig - 1, 0);
  SendDlgItemMessage(ID_SIGVTR, EM_LIMITTEXT, sig - 1, 0);
  SendDlgItemMessage(ID_SIGVTB, EM_LIMITTEXT, sig - 1, 0);
}



//****************************************************************

Empen2Dialog::Empen2Dialog(TWindow*       parent,
                       KiteData*      KD,
                       TModule*       module,
                       TValidator*    validator)
:
  TWindow(parent, empentitle, module),
  TDialog(parent, IDD_EMPEN2DIALOG, module)
{
  TempKD = KD;
  SetCaption(empen2title);
  if (validator){
    new TEdit(this,ID_HTSPAN2)->SetValidator(validator);
    new TEdit(this,ID_CHT2)->SetValidator(validator);
    new TEdit(this,ID_AHT2)->SetValidator(validator);
    new TEdit(this,ID_THICKHT2)->SetValidator(validator);
    new TEdit(this,ID_XHT2)->SetValidator(validator);
    new TEdit(this,ID_ZHT2)->SetValidator(validator);
    new TEdit(this,ID_IT2)->SetValidator(validator);
    new TEdit(this,ID_AZLLHT2)->SetValidator(validator);
    new TEdit(this,ID_NETAHT2)->SetValidator(validator);
    new TEdit(this,ID_VTSPAN2)->SetValidator(validator);
    new TEdit(this,ID_CVT2)->SetValidator(validator);
    new TEdit(this,ID_AVT2)->SetValidator(validator);
    new TEdit(this,ID_THICKVT2)->SetValidator(validator);
    new TEdit(this,ID_XVT2)->SetValidator(validator);
    new TEdit(this,ID_ZVT2)->SetValidator(validator);
    new TEdit(this,ID_NETAVT2)->SetValidator(validator);
  }
}

Empen2Dialog::~Empen2Dialog()
{

}

//
// sets and gets the values of the items (controls) of the input dialog
//
void
Empen2Dialog::TransferData(TTransferDirection direction)
{
   char  Buffer[25] = "";
   double tempdbl;
   if (direction == tdSetData) {
      gcvt(TempKD->bht2,sig,Buffer);
      SetDlgItemText(ID_HTSPAN2, Buffer);
      gcvt(TempKD->cht2,sig,Buffer);
      SetDlgItemText(ID_CHT2, Buffer);
      gcvt(TempKD->aht2,sig,Buffer);
      SetDlgItemText(ID_AHT2, Buffer);
      gcvt(TempKD->Thickht2,sig,Buffer);
      SetDlgItemText(ID_THICKHT2, Buffer);
      gcvt(TempKD->Xht2,sig,Buffer);
      SetDlgItemText(ID_XHT2, Buffer);
      gcvt(TempKD->Zht2,sig,Buffer);
      SetDlgItemText(ID_ZHT2, Buffer);
      gcvt(TempKD->iht2,sig,Buffer);
      SetDlgItemText(ID_IT2, Buffer);
      gcvt(TempKD->azllht2,sig,Buffer);
      SetDlgItemText(ID_AZLLHT2, Buffer);
      gcvt(TempKD->Netaht2,sig,Buffer);
      SetDlgItemText(ID_NETAHT2, Buffer);
      gcvt(TempKD->bvt2,sig,Buffer);
      SetDlgItemText(ID_VTSPAN2, Buffer);
      gcvt(TempKD->cvt2,sig,Buffer);
      SetDlgItemText(ID_CVT2, Buffer);
      gcvt(TempKD->avt2,sig,Buffer);
      SetDlgItemText(ID_AVT2, Buffer);
      gcvt(TempKD->Thickvt2,sig,Buffer);
      SetDlgItemText(ID_THICKVT2, Buffer);
      gcvt(TempKD->Xvt2,sig,Buffer);
      SetDlgItemText(ID_XVT2, Buffer);
      gcvt(TempKD->Zvt2,sig,Buffer);
      SetDlgItemText(ID_ZVT2, Buffer);
      gcvt(TempKD->Netavt2,sig,Buffer);
    }
    else if (direction == tdGetData) {
       GetDlgItemText(ID_HTSPAN2, Buffer, sig);
       tempdbl = atof(Buffer);
       if(tempdbl>=0.0) TempKD->bht2 = tempdbl;
       GetDlgItemText(ID_CHT2, Buffer, sig);
       tempdbl = atof(Buffer);
       if(tempdbl>0.0) TempKD->cht2 = tempdbl;
       GetDlgItemText(ID_AHT2, Buffer, sig);
       tempdbl = atof(Buffer);
       if(tempdbl>0.0&&tempdbl<6.29) TempKD->aht2 = tempdbl;
       GetDlgItemText(ID_THICKHT2, Buffer, sig);
       tempdbl = atof(Buffer);
       if(tempdbl>0.0&&tempdbl<40.0) TempKD->Thickht2 = tempdbl;
       GetDlgItemText(ID_XHT2, Buffer, sig);
       tempdbl = atof(Buffer);
       TempKD->Xht2 = tempdbl;
       GetDlgItemText(ID_ZHT2, Buffer, sig);
       tempdbl = atof(Buffer);
       TempKD->Zht2 = tempdbl;
       GetDlgItemText(ID_IT2, Buffer, sig);
       tempdbl = atof(Buffer);
       TempKD->iht2 = tempdbl;
       GetDlgItemText(ID_AZLLHT2, Buffer, sig);
       tempdbl = atof(Buffer);
       TempKD->azllht2 = tempdbl;
       GetDlgItemText(ID_NETAHT2, Buffer, sig);
       tempdbl = atof(Buffer);
       if(tempdbl>0.0&&tempdbl<=1.0) TempKD->Netaht2 = tempdbl;
       GetDlgItemText(ID_VTSPAN2, Buffer, sig);
       tempdbl = atof(Buffer);
       if(tempdbl>=0.0) TempKD->bvt2 = tempdbl;
       GetDlgItemText(ID_CVT2, Buffer, sig);
       tempdbl = atof(Buffer);
       if(tempdbl>0.0) TempKD->cvt2 = tempdbl;
       GetDlgItemText(ID_AVT2, Buffer, sig);
       tempdbl = atof(Buffer);
       if(tempdbl>0.0&&tempdbl<6.29) TempKD->avt2 = tempdbl;
       GetDlgItemText(ID_THICKVT2, Buffer, sig);
       tempdbl = atof(Buffer);
       if(tempdbl>0.0&&tempdbl<40.0) TempKD->Thickvt2 = tempdbl;
       GetDlgItemText(ID_XVT2, Buffer, sig);
       tempdbl = atof(Buffer);
       TempKD->Xvt2 = tempdbl;
       GetDlgItemText(ID_ZVT2, Buffer, sig);
       tempdbl = atof(Buffer);
       TempKD->Zvt2 = tempdbl;
       GetDlgItemText(ID_NETAVT2, Buffer, sig);
       tempdbl = atof(Buffer);
       if(tempdbl>0.0&&tempdbl<=1.0) TempKD->Netavt2 = tempdbl;
    }
}

//
// sets the values of the items(controls) of the input dialog
//
void
Empen2Dialog::SetupWindow()
{
  TDialog::SetupWindow();
  SendDlgItemMessage(ID_HTSPAN2, EM_LIMITTEXT, sig - 1, 0);
  SendDlgItemMessage(ID_CHT2, EM_LIMITTEXT, sig - 1, 0);
  SendDlgItemMessage(ID_AHT2, EM_LIMITTEXT, sig - 1, 0);
  SendDlgItemMessage(ID_THICKHT2, EM_LIMITTEXT, sig - 1, 0);
  SendDlgItemMessage(ID_XHT2, EM_LIMITTEXT, sig - 1, 0);
  SendDlgItemMessage(ID_ZHT2, EM_LIMITTEXT, sig - 1, 0);
  SendDlgItemMessage(ID_NETAHT2, EM_LIMITTEXT, sig - 1, 0);
  SendDlgItemMessage(ID_VTSPAN2, EM_LIMITTEXT, sig - 1, 0);
  SendDlgItemMessage(ID_CVT2, EM_LIMITTEXT, sig - 1, 0);
  SendDlgItemMessage(ID_AVT2, EM_LIMITTEXT, sig - 1, 0);
  SendDlgItemMessage(ID_THICKVT2, EM_LIMITTEXT, sig - 1, 0);
  SendDlgItemMessage(ID_XVT2, EM_LIMITTEXT, sig - 1, 0);
  SendDlgItemMessage(ID_ZVT2, EM_LIMITTEXT, sig - 1, 0);
  SendDlgItemMessage(ID_NETAVT2, EM_LIMITTEXT, sig - 1, 0);
}


//**************************************************************

FuseDialog::FuseDialog(TWindow*       parent,
                        KiteData*       KD,
                        TModule*        module,
                        TValidator*     validator)
:
  TWindow(parent, fusetitle, module),
  TDialog(parent, IDD_FUSEDIALOG, module)
{
  TempKD = KD;

  SetCaption(fusetitle);
  if (validator){
    new TEdit(this,ID_VOLFS)->SetValidator(validator);
    new TEdit(this,ID_FSW)->SetValidator(validator);
    new TEdit(this,ID_FSH)->SetValidator(validator);
    new TEdit(this,ID_FSQW)->SetValidator(validator);
    new TEdit(this,ID_FSQH)->SetValidator(validator);
    new TEdit(this,ID_FSTQW)->SetValidator(validator);
    new TEdit(this,ID_FSTQH)->SetValidator(validator);
    new TEdit(this,ID_CFS)->SetValidator(validator);
    new TEdit(this,ID_LFS)->SetValidator(validator);
    new TEdit(this,ID_SIGFSB)->SetValidator(validator);
    new TEdit(this,ID_AFS)->SetValidator(validator);
    new TEdit(this,ID_ZFSCP)->SetValidator(validator);
    new TEdit(this,ID_FSXCV)->SetValidator(validator);
  }
}

FuseDialog::~FuseDialog()
{

}

//
// sets and gets the values of the items (controls) of the input dialog
//
void
FuseDialog::TransferData(TTransferDirection direction)
{
   char  Buffer[25] = "";
   double tempdbl;
   if (direction == tdSetData) {
      gcvt(TempKD->Volfs,sig,Buffer);
      SetDlgItemText(ID_VOLFS, Buffer);
      gcvt(TempKD->fsw,sig,Buffer);
      SetDlgItemText(ID_FSW, Buffer);
      gcvt(TempKD->fsh,sig,Buffer);
      SetDlgItemText(ID_FSH, Buffer);
      gcvt(TempKD->fsqw,sig,Buffer);
      SetDlgItemText(ID_FSQW, Buffer);
      gcvt(TempKD->fsqh,sig,Buffer);
      gcvt(TempKD->fstqw,sig,Buffer);
      SetDlgItemText(ID_FSTQW, Buffer);
      gcvt(TempKD->fstqh,sig,Buffer);
      SetDlgItemText(ID_FSTQH, Buffer);
      gcvt(TempKD->cfs,sig,Buffer);
      SetDlgItemText(ID_CFS, Buffer);
      gcvt(TempKD->lfs,sig,Buffer);
      SetDlgItemText(ID_LFS, Buffer);
      gcvt(TempKD->sigfs_B,sig,Buffer);
      SetDlgItemText(ID_SIGFSB, Buffer);
      gcvt(TempKD->afs,sig,Buffer);
      SetDlgItemText(ID_AFS, Buffer);
      gcvt(TempKD->Zfscp,sig,Buffer);
      SetDlgItemText(ID_ZFSCP, Buffer);
      gcvt(TempKD->Xcv,sig,Buffer);
      SetDlgItemText(ID_FSXCV, Buffer);

   }
   else if (direction == tdGetData) {
      GetDlgItemText(ID_VOLFS, Buffer, sig);
      tempdbl = atof(Buffer);
      if(tempdbl>0.0) TempKD->Volfs = tempdbl;
      GetDlgItemText(ID_FSW, Buffer, sig);
      tempdbl = atof(Buffer);
      if(tempdbl>0.0) TempKD->fsw = tempdbl;
      GetDlgItemText(ID_FSH, Buffer, sig);
      tempdbl = atof(Buffer);
      if(tempdbl>0.0) TempKD->fsh = tempdbl;
      GetDlgItemText(ID_FSQW, Buffer, sig);
      tempdbl = atof(Buffer);
      if(tempdbl>0.0) TempKD->fsqw = tempdbl;
      GetDlgItemText(ID_FSQH, Buffer, sig);
      tempdbl = atof(Buffer);
      if(tempdbl>0.0) TempKD->fsqh = tempdbl;
      GetDlgItemText(ID_FSTQW, Buffer, sig);
      tempdbl = atof(Buffer);
      if(tempdbl>0.0) TempKD->fstqw = tempdbl;
      GetDlgItemText(ID_FSTQH, Buffer, sig);
      tempdbl = atof(Buffer);
      if(tempdbl>0.0) TempKD->fstqh = tempdbl;
      GetDlgItemText(ID_CFS, Buffer, sig);
      tempdbl = atof(Buffer);
      if(tempdbl>0.0) TempKD->cfs = tempdbl;
      GetDlgItemText(ID_LFS, Buffer, sig);
      tempdbl = atof(Buffer);
      if(tempdbl>0.0) TempKD->lfs = tempdbl;
      GetDlgItemText(ID_SIGFSB, Buffer, sig);
      tempdbl = atof(Buffer);
      TempKD->sigfs_B = tempdbl;
      GetDlgItemText(ID_AFS, Buffer, sig);
      tempdbl = atof(Buffer);
      if(tempdbl>0.0&&tempdbl<6.29) TempKD->afs = tempdbl;
      GetDlgItemText(ID_ZFSCP, Buffer, sig);
      tempdbl = atof(Buffer);
      TempKD->Zfscp = tempdbl;
      GetDlgItemText(ID_FSXCV, Buffer, sig);
      tempdbl = atof(Buffer);
      if(tempdbl>0.0) TempKD->Xcv = tempdbl;

   }
}

//
// sets the values of the items(controls) of the input dialog
//
void
```

```
{
  TDialog::SetupWindow();
  SendDlgItemMessage(ID_VOLFS, EM_LIMITTEXT, sig - 1, 0);
  SendDlgItemMessage(ID_FSW, EM_LIMITTEXT, sig - 1, 0);
  SendDlgItemMessage(ID_FSH, EM_LIMITTEXT, sig - 1, 0);
  SendDlgItemMessage(ID_FSQW, EM_LIMITTEXT, sig - 1, 0);
  SendDlgItemMessage(ID_FSQH, EM_LIMITTEXT, sig - 1, 0);
  SendDlgItemMessage(ID_FSTQW, EM_LIMITTEXT, sig - 1, 0);
  SendDlgItemMessage(ID_FSTQH, EM_LIMITTEXT, sig - 1, 0);
  SendDlgItemMessage(ID_CFS, EM_LIMITTEXT, sig - 1, 0);
  SendDlgItemMessage(ID_LFS, EM_LIMITTEXT, sig - 1, 0);
  SendDlgItemMessage(ID_SIGFSB, EM_LIMITTEXT, sig - 1, 0);
  SendDlgItemMessage(ID_AFS, EM_LIMITTEXT, sig - 1, 0);
  SendDlgItemMessage(ID_ZFSCP, EM_LIMITTEXT, sig - 1, 0);
  SendDlgItemMessage(ID_FSXCV, EM_LIMITTEXT, sig - 1, 0);
}



//***************************************************************

ControlGainDialog::ControlGainDialog(TWindow*       parent,
                                    ControlGain*    CG,
                                    TModule*        module,
                                    TValidator*     validator)
:
  TWindow(parent, controlgaintitle, module),
  TDialog(parent, IDD_CONTROLGAINDIALOG, module)
{
  TempCG = CG;

  SetCaption(controlgaintitle);
  if (validator){
    new TEdit(this,ID_CCX)->SetValidator(validator);
    new TEdit(this,ID_CCY)->SetValidator(validator);
    new TEdit(this,ID_CCZ)->SetValidator(validator);
    new TEdit(this,ID_CCL)->SetValidator(validator);
    new TEdit(this,ID_CCM)->SetValidator(validator);
    new TEdit(this,ID_CCN)->SetValidator(validator);
  }
}

ControlGainDialog::~ControlGainDialog()
{

}

//
// sets and gets the values of the items (controls) of the input dialog
//
void
ControlGainDialog::TransferData(TTransferDirection direction)
{
   char  Buffer[25] = "";
   double tempdbl;
   if (direction == tdSetData) {
      gcvt(TempCG->Ccx,sig,Buffer);
      SetDlgItemText(ID_CCX, Buffer);
      gcvt(TempCG->Ccy,sig,Buffer);
      SetDlgItemText(ID_CCY, Buffer);
      gcvt(TempCG->Ccz,sig,Buffer);
      SetDlgItemText(ID_CCZ, Buffer);
      gcvt(TempCG->Ccl,sig,Buffer);
      SetDlgItemText(ID_CCL, Buffer);
      gcvt(TempCG->Ccm,sig,Buffer);
      SetDlgItemText(ID_CCM, Buffer);
```

```
         SetDlgItemText(ID_CCN, Buffer);

      }
      else if (direction == tdGetData) {
         GetDlgItemText(ID_CCX, Buffer, sig);
         tempdbl = atof(Buffer);
         TempCG->Ccx = tempdbl;
         GetDlgItemText(ID_CCY, Buffer, sig);
         tempdbl = atof(Buffer);
         TempCG->Ccy = tempdbl;
         GetDlgItemText(ID_CCZ, Buffer, sig);
         tempdbl = atof(Buffer);
         TempCG->Ccz = tempdbl;
         GetDlgItemText(ID_CCL, Buffer, sig);
         tempdbl = atof(Buffer);
         TempCG->Ccl = tempdbl;
         GetDlgItemText(ID_CCM, Buffer, sig);
         tempdbl = atof(Buffer);
         TempCG->Ccm = tempdbl;
         GetDlgItemText(ID_CCN, Buffer, sig);
         tempdbl = atof(Buffer);
         TempCG->Ccn = tempdbl;

      }
   }

   //
   // sets the values of the items(controls) of the input dialog
   //
   void
   ControlGainDialog::SetupWindow()
   {
     TDialog::SetupWindow();
     SendDlgItemMessage(ID_CCX, EM_LIMITTEXT, sig - 1, 0);
     SendDlgItemMessage(ID_CCY, EM_LIMITTEXT, sig - 1, 0);
     SendDlgItemMessage(ID_CCZ, EM_LIMITTEXT, sig - 1, 0);
     SendDlgItemMessage(ID_CCL, EM_LIMITTEXT, sig - 1, 0);
     SendDlgItemMessage(ID_CCM, EM_LIMITTEXT, sig - 1, 0);
     SendDlgItemMessage(ID_CCN, EM_LIMITTEXT, sig - 1, 0);
   }




   //**************************************************************

   WindFormDialog::WindFormDialog(TWindow*        parent,
                                  KiteData*       KD,
                                  TModule*        module,
                                  TValidator*     validator)
   :
     TWindow(parent, windformtitle, module),
     TDialog(parent, IDD_WINDFORMULA, module)
   {
     TempKD = KD;

     SetCaption(windformtitle);
     if (validator){
       new TEdit(this,ID_WFORDER)->SetValidator(validator);
       new TEdit(this,ID_WFMAXALT)->SetValidator(validator);
       new TEdit(this,ID_WFMINALT)->SetValidator(validator);
       new TEdit(this,ID_WFA0)->SetValidator(validator);
       new TEdit(this,ID_WFA1)->SetValidator(validator);
       new TEdit(this,ID_WFA2)->SetValidator(validator);
```

```
    new TEdit(this,ID_WFA4)->SetValidator(validator);
    new TEdit(this,ID_WFA5)->SetValidator(validator);
    new TEdit(this,ID_WFA6)->SetValidator(validator);
    new TEdit(this,ID_WFA7)->SetValidator(validator);
    new TEdit(this,ID_WFA8)->SetValidator(validator);
    new TEdit(this,ID_WFA9)->SetValidator(validator);
  }
}

WindFormDialog::~WindFormDialog()
{

}

//
// sets and gets the values of the items (controls) of the input dialog
//
void
WindFormDialog::TransferData(TTransferDirection direction)
{
   char  Buffer[25] = "";
   int i;
   double tempdbl;
   if (direction == tdSetData) {
      if (!TempKD->IWF){
         gcvt(TempKD->WFA[0],sig,Buffer);
         SetDlgItemText(ID_WFORDER, Buffer);
         gcvt(TempKD->WFA[1],sig,Buffer);
         SetDlgItemText(ID_WFMINALT, Buffer);
         gcvt(TempKD->WFA[2],sig,Buffer);
         SetDlgItemText(ID_WFMAXALT, Buffer);
         for(i=0;i<=TempKD->WFA[0];i++)
         {
            gcvt(TempKD->WFA[i+3],sig,Buffer);
            SetDlgItemText(ID_WFA0+i, Buffer);
         }
         for(i=TempKD->WFA[0]+1;i<=9;i++)
         {
            SetDlgItemText(ID_WFA0+i, "0.0");
         }
      }else{
         SetDlgItemText(ID_WFORDER, "0");
         strcpy(Buffer,"0.0");
         SetDlgItemText(ID_WFMAXALT, Buffer);
         SetDlgItemText(ID_WFMINALT, Buffer);
         SetDlgItemText(ID_WFA0, Buffer);
         SetDlgItemText(ID_WFA1, Buffer);
         SetDlgItemText(ID_WFA2, Buffer);
         SetDlgItemText(ID_WFA3, Buffer);
         SetDlgItemText(ID_WFA4, Buffer);
         SetDlgItemText(ID_WFA5, Buffer);
         SetDlgItemText(ID_WFA6, Buffer);
         SetDlgItemText(ID_WFA7, Buffer);
         SetDlgItemText(ID_WFA8, Buffer);
         SetDlgItemText(ID_WFA9, Buffer);
      }
   }
   else if (direction == tdGetData) {
      GetDlgItemText(ID_WFORDER, Buffer, sig);
      tempdbl = atof(Buffer);
      if(tempdbl>=0.0 && tempdbl<=9.0)
      {
         double tempmin,tempmax;
         GetDlgItemText(ID_WFMINALT, Buffer, sig);
         tempmin = atof(Buffer);
         GetDlgItemText(ID_WFMAXALT, Buffer, sig);
```

```
            //Max altitude > Min altitude >= 0
            if(tempmax>tempmin && tempmin>=0.0)
            {
               TempKD->IWF=0;
               TempKD->FileForm("Wind Data Equation");
               TempKD->WFA[0] = tempdbl;
               TempKD->WFA[1] = tempmin;
               TempKD->WFA[2] = tempmax;
               for(i=0;i<=TempKD->WFA[0];i++)
               {
                  GetDlgItemText(ID_WFA0+i, Buffer, sig);
                  TempKD->WFA[3+i] = atof(Buffer);
               }
            }
         }
      }
   }

   //
   // sets the values of the items(controls) of the input dialog
   //
   void
   WindFormDialog::SetupWindow()
   {
     TDialog::SetupWindow();
     SendDlgItemMessage(ID_WFORDER, EM_LIMITTEXT, sig - 12, 0);
     SendDlgItemMessage(ID_WFMAXALT, EM_LIMITTEXT, sig - 1, 0);
     SendDlgItemMessage(ID_WFMINALT, EM_LIMITTEXT, sig - 1, 0);
     SendDlgItemMessage(ID_WFA0, EM_LIMITTEXT, sig + 5, 0);
     SendDlgItemMessage(ID_WFA1, EM_LIMITTEXT, sig + 5, 0);
     SendDlgItemMessage(ID_WFA2, EM_LIMITTEXT, sig + 5, 0);
     SendDlgItemMessage(ID_WFA3, EM_LIMITTEXT, sig + 5, 0);
     SendDlgItemMessage(ID_WFA4, EM_LIMITTEXT, sig + 5, 0);
     SendDlgItemMessage(ID_WFA5, EM_LIMITTEXT, sig + 5, 0);
     SendDlgItemMessage(ID_WFA6, EM_LIMITTEXT, sig + 5, 0);
     SendDlgItemMessage(ID_WFA7, EM_LIMITTEXT, sig + 5, 0);
     SendDlgItemMessage(ID_WFA8, EM_LIMITTEXT, sig + 5, 0);
     SendDlgItemMessage(ID_WFA9, EM_LIMITTEXT, sig + 5, 0);
   }




   //***************************************************************

   TrimResultsDialog::TrimResultsDialog(TWindow*       parent,
                               KiteData*       KD,
                               int             isvalid,
                               TModule*        module,
                               TValidator*     validator)
   :
     TWindow(parent, TrimResultstitle, module),
     TDialog(parent, IDD_TRIMRESULTSDIALOG, module)
   {
     TempKD = KD;
     Valid = isvalid;

     SetCaption(TrimResultstitle);
     if (validator){
       new TEdit(this,ID_RMASS)->SetValidator(validator);
       new TEdit(this,ID_RSPAN)->SetValidator(validator);
       new TEdit(this,ID_RWIND)->SetValidator(validator);
       new TStatic(this,ID_RWINDFILE);
```

```
    new TEdit(this,ID_RSALT)->SetValidator(validator);
    new TEdit(this,ID_RALT)->SetValidator(validator);
    new TEdit(this,ID_RTTLEN)->SetValidator(validator);
    new TEdit(this,ID_RAOA)->SetValidator(validator);
    new TEdit(this,ID_RAOAMAX)->SetValidator(validator);
    new TEdit(this,ID_RCL)->SetValidator(validator);
    new TEdit(this,ID_RCD)->SetValidator(validator);
    new TEdit(this,ID_RNELEM)->SetValidator(validator);
    new TEdit(this,ID_RTBAR)->SetValidator(validator);
    new TEdit(this,ID_RGAMMAB)->SetValidator(validator);
  }
}

TrimResultsDialog::~TrimResultsDialog()
{

}

//
// sets and gets the values of the items (controls) of the input dialog
//
void
TrimResultsDialog::TransferData(TTransferDirection direction)
{
   char  Buffer[25] = "";
   if (direction == tdSetData) {
      gcvt(TempKD->mass,sig,Buffer);
      SetDlgItemText(ID_RMASS, Buffer);
      gcvt(TempKD->b,sig,Buffer);
      SetDlgItemText(ID_RSPAN, Buffer);
      SetDlgItemText(ID_RWINDFILE, TempKD->FileForm());
      if(Valid){
         if(TempKD->aoaexceed==1)
            SetDlgItemText(ID_RAOAEXCEED,
            "Max. Wing AOA has been exceeded. Please reconfigure the THARWP!");
         gcvt(TempKD->windvel,sig-9,Buffer);
         SetDlgItemText(ID_RWIND, Buffer);
         gcvt(TempKD->rsalt,sig-9,Buffer);
         SetDlgItemText(ID_RSALT, Buffer);
         gcvt(TempKD->ralt,sig-9,Buffer);
         SetDlgItemText(ID_RALT, Buffer);
         gcvt(TempKD->ttlen,sig-9,Buffer);
         SetDlgItemText(ID_RTTLEN, Buffer);
         gcvt(TempKD->AOA,sig-9,Buffer);
         SetDlgItemText(ID_RAOA, Buffer);
         gcvt(TempKD->aoamax,sig-9,Buffer);
         SetDlgItemText(ID_RAOAMAX, Buffer);
         gcvt(TempKD->rcl,sig-9,Buffer);
         SetDlgItemText(ID_RCL, Buffer);
         gcvt(TempKD->rcd,sig-9,Buffer);
         SetDlgItemText(ID_RCD, Buffer);
         gcvt(TempKD->N,sig-9,Buffer);
         SetDlgItemText(ID_RNELEM, Buffer);
         gcvt(TempKD->Tbar,sig-9,Buffer);
         SetDlgItemText(ID_RTBAR, Buffer);
         gcvt(TempKD->Gammabar*180.0/TempKD->pi,sig-9,Buffer);
         SetDlgItemText(ID_RGAMMAB, Buffer);
      }else{
         strcpy(Buffer,"");
         if(TempKD->AOA==-99.0)
            SetDlgItemText(ID_RAOAEXCEED,
            "No solution for trim state AOA. Please reconfigure the THARWP!");
         SetDlgItemText(ID_RWIND, Buffer);
         SetDlgItemText(ID_RSALT, Buffer);
         SetDlgItemText(ID_RALT, Buffer);
         SetDlgItemText(ID_RTTLEN, Buffer);
         SetDlgItemText(ID_RAOAMAX, Buffer);
         SetDlgItemText(ID_RCL, Buffer);
         SetDlgItemText(ID_RCD, Buffer);
         SetDlgItemText(ID_RNELEM, Buffer);
         SetDlgItemText(ID_RTBAR, Buffer);
         SetDlgItemText(ID_RGAMMAB, Buffer);
      }
   }
}

//
// sets the values of the items(controls) of the input dialog
//
void
TrimResultsDialog::SetupWindow()
{
  TDialog::SetupWindow();
  SendDlgItemMessage(ID_RMASS, EM_LIMITTEXT, sig - 1, 0);
  SendDlgItemMessage(ID_RSPAN, EM_LIMITTEXT, sig - 1, 0);
  SendDlgItemMessage(ID_RWIND, EM_LIMITTEXT, sig - 1, 0);
  SendDlgItemMessage(ID_RWINDFILE, EM_LIMITTEXT, 150, 0);
  SendDlgItemMessage(ID_RAOAEXCEED, EM_LIMITTEXT, 150, 0);
  SendDlgItemMessage(ID_RSALT, EM_LIMITTEXT, sig - 1, 0);
  SendDlgItemMessage(ID_RALT, EM_LIMITTEXT, sig - 1, 0);
  SendDlgItemMessage(ID_RTTLEN, EM_LIMITTEXT, sig - 1, 0);
  SendDlgItemMessage(ID_RAOA, EM_LIMITTEXT, sig - 9, 0);
  SendDlgItemMessage(ID_RAOAMAX, EM_LIMITTEXT, sig - 9, 0);
  SendDlgItemMessage(ID_RCL, EM_LIMITTEXT, sig - 9, 0);
  SendDlgItemMessage(ID_RCD, EM_LIMITTEXT, sig - 9, 0);
  SendDlgItemMessage(ID_RNELEM, EM_LIMITTEXT, sig - 9, 0);
  SendDlgItemMessage(ID_RTBAR, EM_LIMITTEXT, sig - 9, 0);
  SendDlgItemMessage(ID_RGAMMAB, EM_LIMITTEXT, sig - 9, 0);
}


//**************************************************************

DEFINE_RESPONSE_TABLE1(StabilityResultsDialog, TDialog)
  EV_COMMAND(ID_LGNORMALIZE, CmUpdateLgEignVect),
  EV_COMMAND(ID_LTNORMALIZE, CmUpdateLtEignVect),
  EV_COMMAND(ID_LGSTORE, CmLgStore),
  EV_COMMAND(ID_LTSTORE, CmLtStore),
  EV_LBN_SELCHANGE(IDL_LGEIGNVALS,  CmUpdateLgEignVect),
  EV_LBN_SELCHANGE(IDL_LTEIGNVALS,  CmUpdateLtEignVect),
END_RESPONSE_TABLE;

StabilityResultsDialog::StabilityResultsDialog(
                        TWindow*       parent,
                        KiteData*         KD,
                        TModule*        module,
                        TValidator*     validator)
:
  TWindow(parent, StabilityResultstitle, module),
  TDialog(parent, IDD_STABILITYRESULTSDIALOG, module)
{
   TempKD   = KD;

   SetCaption(StabilityResultstitle);
   if (validator){
      new TEdit(this,ID_RMASS)->SetValidator(validator);
      new TEdit(this,ID_RSPAN)->SetValidator(validator);
      new TStatic(this,ID_RWINDFILE);
   }
   LgNorm = new TCheckBox(this, ID_LGNORMALIZE);
   LtNorm = new TCheckBox(this, ID_LTNORMALIZE);
   LgEignVal=0;
   LtEignVal=0;
   LtEignVect=0;

   if(TempKD->N<=1)
      N=0;
   else
      N=TempKD->N;

   LgIter(N*4+6);
   LtIter(N*2+6);
}

StabilityResultsDialog::~StabilityResultsDialog()
{
   delete LgEignVal;
   delete LgEignVect;
   delete LtEignVal;
   delete LtEignVect;
   delete LgNorm;
   delete LtNorm;
}

void
StabilityResultsDialog::CmUpdateLgEignVect()
{
   char Buffer[40] = "";
   char Buf[25] = "";
   int i;
   int index = LgEignVal->GetSelIndex();        // Current selection.
   complex div,eigtmp;
   //Clear the previous modeshape
   LgEignVect->ClearList();
   if (index != LB_ERR)                         // is something selected?
   {
      //Check to see if the modeshape needs to be normalized and set the
      //divisor to the appropriate value
      if (LgNorm->GetCheck() == BF_CHECKED)
      {
         //Normalize wrt Theta
         if(LgIter(index+1)<0)
            div = complex(TempKD->Lgz(4,abs(LgIter(index+1))),
                          TempKD->Lgz(4,abs(LgIter(index+1))+1));
         else
            div = complex(TempKD->Lgz(4,abs(LgIter(index+1))),0.0);

      }else
         div = complex(1.0,0.0);

      for(i=1;i<=4;i++)
      {
         switch(i) {
            case 1:
               strcpy(Buffer,"u: ");
               break;
            case 2:
               strcpy(Buffer,"w: ");
               break;
            case 3:
               strcpy(Buffer,"q: ");
               break;
            case 4:
               strcpy(Buffer,"theta: ");
               break;
         }

         if(LgIter(index+1)<0&&div!=complex(0.0,0.0)) //eigenvalue is complex
```

```
          TempKD->Lgz(i,abs(LgIter(index+1))+1))/div;
      gcvt(real(eigtmp),8,Buf);
      strcat(Buffer,Buf);
      gcvt(fabs(imag(eigtmp)),8,Buf);
      if(imag(eigtmp)>0.0)
        strcat(Buffer," + i");
      else
        strcat(Buffer," - i");
      strcat(Buffer,Buf);
    }else if(div!=complex(0.0,0.0)){          //eigenvalue is real
      eigtmp=complex(TempKD->Lgz(i,abs(LgIter(index+1))),0.0)/div;
      gcvt(real(eigtmp),8,Buf);
      strcat(Buffer,Buf);
    }
    LgEignVect->AddString(Buffer);
  }
  LgEignVect->AddString("                    ");
  LgEignVect->AddString("Element Stretch");
  for(i=N*2+7;i<=N*4+6;i+=2)
  {
    strcpy(Buffer,"");
    if(LgIter(index+1)<0&&div!=complex(0.0,0.0)) //eigenvalue is complex
    {
      eigtmp=complex(TempKD->Lgz(i,abs(LgIter(index+1))),
          TempKD->Lgz(i,abs(LgIter(index+1))+1))/div;
      gcvt(real(eigtmp),8,Buf);
      strcat(Buffer,Buf);
      gcvt(fabs(imag(eigtmp)),8,Buf);
      if(imag(eigtmp)>0.0)
        strcat(Buffer," + i");
      else
        strcat(Buffer," - i");
      strcat(Buffer,Buf);
    }else if(div!=complex(0.0,0.0)){    //eigenvalue is real
      eigtmp=complex(TempKD->Lgz(i,abs(LgIter(index+1))),0.0)/div;
      gcvt(real(eigtmp),8,Buf);
      strcat(Buffer,Buf);
    }
    LgEignVect->AddString(Buffer);
  }
  LgEignVect->AddString("             ");
  LgEignVect->AddString("Delta Gamma");
  for(i=N*2+8;i<=N*4+6;i+=2)
  {
    strcpy(Buffer,"");
    if(LgIter(index+1)<0&&div!=complex(0.0,0.0)) //eigenvalue is complex
    {
      eigtmp=complex(TempKD->Lgz(i,abs(LgIter(index+1))),
          TempKD->Lgz(i,abs(LgIter(index+1))+1))/div;
      gcvt(real(eigtmp),8,Buf);
      strcat(Buffer,Buf);
      gcvt(fabs(imag(eigtmp)),8,Buf);
      if(imag(eigtmp)>0.0)
        strcat(Buffer," + i");
      else
        strcat(Buffer," - i");
      strcat(Buffer,Buf);
    }else if(div!=complex(0.0,0.0)){                    //eigenvalue is real
      eigtmp=complex(TempKD->Lgz(i,abs(LgIter(index+1))),0.0)/div;
      gcvt(real(eigtmp),8,Buf);
      strcat(Buffer,Buf);
    }
    LgEignVect->AddString(Buffer);
  }
}
```

```
void
StabilityResultsDialog::CmUpdateLtEignVect()
{
   char Buffer[40] = "";
   char Buf[25]  = "";
   int i;
   int index = LtEignVal->GetSelIndex();          // Current selection.
   complex div,eigtmp;
   // Clear the previous modeshape
   LtEignVect->ClearList();
   if (index != LB_ERR)                           // is something selected?
   {
      //Check to see if the modeshape needs to be normalized and set the
      //divisor to the appropriate value
      if (LtNorm->GetCheck() == BF_CHECKED)
      {
         //Normalize wrt Theta
         if(LtIter(index+1)<0)
            div = complex(TempKD->Ltz(5,abs(LtIter(index+1))),
                          TempKD->Ltz(5,abs(LtIter(index+1))+1));
         else
            div = complex(TempKD->Ltz(5,abs(LtIter(index+1))),0.0);

      }else
         div = complex(1.0,0.0);

      for(i=1;i<=5;i++)
      {
         switch(i) {
            case 1:
               strcpy(Buffer,"v: ");
               break;
            case 2:
               strcpy(Buffer,"p: ");
               break;
            case 3:
               strcpy(Buffer,"r: ");
               break;
            case 4:
               strcpy(Buffer,"phi: ");
               break;
            case 5:
               strcpy(Buffer,"psi: ");
               break;
         }

         if(LtIter(index+1)<0&&div!=complex(0.0,0.0)) //eigenvalue is complex
         {
            eigtmp=complex(TempKD->Ltz(i,abs(LtIter(index+1))),
                  TempKD->Ltz(i,abs(LtIter(index+1))+1))/div;
            gcvt(real(eigtmp),8,Buf);
            strcat(Buffer,Buf);
            gcvt(fabs(imag(eigtmp)),8,Buf);
            if(imag(eigtmp)>0.0)
               strcat(Buffer," + i");
            else
               strcat(Buffer," - i");
            strcat(Buffer,Buf);
         }else if(div!=complex(0.0,0.0)){     //eigenvalue is real
            eigtmp=complex(TempKD->Ltz(i,abs(LtIter(index+1))),0.0)/div;
            gcvt(real(eigtmp),8,Buf);
            strcat(Buffer,Buf);
         }
         LtEignVect->AddString(Buffer);
      }
```

```
        for(i=N+7;i<=N*2+6;i++)
        {
           strcpy(Buffer,"");
           if(LtIter(index+1)<0&&div!=complex(0.0,0.0)) //eigenvalue is complex
           {
              eigtmp=complex(TempKD->Ltz(i,abs(LtIter(index+1))),
                    TempKD->Ltz(i,abs(LtIter(index+1))+1))/div;
              gcvt(real(eigtmp),8,Buf);
              strcat(Buffer,Buf);
              gcvt(fabs(imag(eigtmp)),8,Buf);
              if(imag(eigtmp)>0.0)
                 strcat(Buffer," + i");
              else
                 strcat(Buffer," - i");
              strcat(Buffer,Buf);
           }else if(div!=complex(0.0,0.0)){ //eigenvalue is real
              eigtmp=complex(TempKD->Ltz(i,abs(LtIter(index+1))),0.0)/div;
              gcvt(real(eigtmp),8,Buf);
              strcat(Buffer,Buf);
           }
           LtEignVect->AddString(Buffer);
        }
     }
  }

  void
  StabilityResultsDialog::CmLgStore()
  {
     int i;

     //Initialize the output file for the individual stability root data
     fstream Stabout;
     Stabout.open("LgSRoots.txt",ios::out|ios::app);

     //Output the eigen data to the listboxes
     if(TempKD->EiglgValid)
     {
        for(i=0;i<N*4+6;i++)
        {
           if(TempKD->Lgalfi[i]>=0.0&&!(TempKD->Lgalfr[i]==0.0 && TempKD->Lgalfi[i]==0.0))
           {
              Stabout << TempKD->Lgalfr[i] << "\t" << TempKD->Lgalfi[i] << "\n";
           }
        }
        Stabout << "\n";
     }
     Stabout.close();
  }

  void
  StabilityResultsDialog::CmLtStore()
  {
     int i;

     //Initialize the output file for the individual stability root data
     fstream Stabout;
     Stabout.open("LtSRoots.txt",ios::out|ios::app);

     //Output the eigen data to the listboxes
     if(TempKD->EigltValid)
     {
        for(i=0;i<N*2+6;i++)
        {
           if(TempKD->Ltalfi[i]>=0.0&&!(TempKD->Ltalfr[i]==0.0 && TempKD->Ltalfi[i]==0.0))
           {
        }
        Stabout << "\n";
     }
     Stabout.close();
  }

  //
  // sets and gets the values of the items (controls) of the input dialog
  //
  void
  StabilityResultsDialog::TransferData(TTransferDirection direction)
  {
     char  Buffer[40] = "";
     if (direction == tdSetData)
     {
        gcvt(TempKD->mass,sig,Buffer);
        SetDlgItemText(ID_RMASS, Buffer);
        gcvt(TempKD->b,sig,Buffer);
        SetDlgItemText(ID_RSPAN, Buffer);
        SetDlgItemText(ID_RWINDFILE, TempKD->FileForm());

     }
  }

  //
  // sets the values of the items(controls) of the input dialog
  //
  void
  StabilityResultsDialog::SetupWindow()
  {
     TDialog::SetupWindow();
     SendDlgItemMessage(ID_RMASS, EM_LIMITTEXT, sig - 1, 0);
     SendDlgItemMessage(ID_RSPAN, EM_LIMITTEXT, sig - 1, 0);
     SendDlgItemMessage(ID_RWINDFILE, EM_LIMITTEXT, 150, 0);

     char  Buffer[40] = "";
     char  Imag[25] = "";
     int i,j,k;
     LgEignVal = new TListBox(this, IDL_LGEIGNVALS);
     LgEignVal->Attr.Style;
     LgEignVal->Create();
     LgEignVal->SetFocus();
     LgEignVect = new TListBox(this, IDL_LGMODESHAPE);
     LgEignVect->Attr.Style;
     LgEignVect->Create();
     LtEignVal = new TListBox(this, IDL_LTEIGNVALS);
     LtEignVal->Attr.Style;
     LtEignVal->Create();
     LtEignVect = new TListBox(this, IDL_LTMODESHAPE);
     LtEignVect->Attr.Style;
     LtEignVect->Create();


     //Output the eigen data to the listboxes
     if(TempKD->EiglgValid)
     {
        i=1;
        j=5;
        for(k=0;k<N*4+6;k++)
        {
           if(TempKD->Lgalfi[k]>=0.0)
           {
              LgIter(i)=j;
              gcvt(TempKD->Lgalfr[k],10,Buffer);
              if(TempKD->Lgalfi[k]>0.0)
```

```
                gcvt(TempKD->Lgalfi[k],10,Imag);
                strcat(Buffer," + i");
                strcat(Buffer,Imag);
                LgIter(i)=-j;               //Set index -ve to indicate complex
                j++;
             }
             LgEignVal->AddString(Buffer);
             i++;
             j++;
          }
       }
       LgEignVal->SetSelIndex(0);
    }
    if(TempKD->EigltValid)
    {
       i=1;
       j=3;
       for(k=0;k<N*2+6;k++)
       {
          if(TempKD->Ltalfi[k]>=0.0)
          {
             LtIter(i)=j;
             gcvt(TempKD->Ltalfr[k],10,Buffer);
             if(TempKD->Ltalfi[k]>0.0)
             {
                gcvt(TempKD->Ltalfi[k],10,Imag);
                strcat(Buffer," + i");
                strcat(Buffer,Imag);
                LtIter(i)=-j;               //Set index -ve to indicate complex
                j++;
             }
             LtEignVal->AddString(Buffer);
             i++;
             j++;
          }
       }
       LtEignVal->SetSelIndex(0);
    }

    //Set both mode shapes to be normalized initially
    LgNorm->SetCheck(BF_CHECKED);
    LtNorm->SetCheck(BF_CHECKED);

    //Update the mode shape listboxes
    CmUpdateLgEignVect();
    CmUpdateLtEignVect();

 }
```

```
#include <string.h>
#include <owl/edit.h>
#include <owl/checkbox.h>
#include <owl/validate.h>
#include "dialspec.h"
#include "dialspec.rh"

char queryvarstudytitle[50]="Super Kite Variation Study Parameters";
char stabilityderivstitle[50]="Super Kite Non-Dimensional Stability Derivatives";
char RLScaletitle[50]="Super Kite Root-Locus Plot Limits";
char TimeSteptitle[50]="Super Kite Root-Time Step Initialization";
char TSScaletitle[50]="Super Kite Time-Step Plot Limits";
const int sig = 15;


//***************************************************************

DEFINE_RESPONSE_TABLE1(QueryVarStudyDialog, TDialog)
   EV_EN_CHANGE(ID_LGSTART,CmCheckRange),
   EV_EN_CHANGE(ID_LGEND,CmCheckRange),
   EV_EN_CHANGE(ID_NSTEPS,CmCheckRange),
   EV_EN_CHANGE(ID_LTSTART,CmCheckRange),
   EV_EN_CHANGE(ID_LTEND,CmCheckRange),
   EV_CHILD_NOTIFY_ALL_CODES(ID_LGGROUP, CmUpdate),
   EV_CHILD_NOTIFY_ALL_CODES(ID_LTGROUP, CmUpdate),
END_RESPONSE_TABLE;

QueryVarStudyDialog::QueryVarStudyDialog(TWindow*       parent,
                                          KiteData*      KD,
                                          KiteQuery*     KQ,
                                          TModule*       module,
                                          TValidator*    validator)
:
  TWindow(parent, queryvarstudytitle, module),
  TDialog(parent, IDD_QUERYVARSTUDYDIALOG, module)
{
   TempKD   = KD;
   KQRef    = KQ;
   TempKQ   = new KiteQuery();
   *TempKQ  = *KQ;
   Init     = 0;

   SetCaption(queryvarstudytitle);
   if (validator){
      new TEdit(this,ID_LGSS)->SetValidator(validator);
      new TEdit(this,ID_LTSS)->SetValidator(validator);
   }

   LgGroupBox = new TGroupBox(this, ID_LGGROUP);
   LtGroupBox = new TGroupBox(this, ID_LTGROUP);

   XGain = new TRadioButton(this, ID_XGAIN, LgGroupBox);
   ZGain = new TRadioButton(this, ID_ZGAIN, LgGroupBox);
   MGain = new TRadioButton(this, ID_MGAIN, LgGroupBox);
   YGain = new TRadioButton(this, ID_YGAIN, LtGroupBox);
   LGain = new TRadioButton(this, ID_LGAIN, LtGroupBox);
   NGain = new TRadioButton(this, ID_NGAIN, LtGroupBox);
   VertSpan = new TRadioButton(this, ID_VERTSPAN, LtGroupBox);
   VertX = new TRadioButton(this, ID_VERTX, LtGroupBox);

   Lgstart = new TEdit(this,ID_LGSTART);
   Lgend = new TEdit(this,ID_LGEND);
   Ltstart = new TEdit(this,ID_LTSTART);
   Ltend = new TEdit(this,ID_LTEND);
   steps = new TEdit(this,ID_NSTEPS);
   Calculate = new TButton(this,ID_OK);
}

QueryVarStudyDialog::~QueryVarStudyDialog()
{
   delete LgGroupBox;
   delete LtGroupBox;
   delete XGain;
   delete ZGain;
   delete MGain;
   delete YGain;
   delete LGain;
   delete NGain;
   delete VertSpan;
   delete VertX;
   delete Lgstart;
   delete Lgend;
   delete Ltstart;
   delete Ltend;
   delete steps;
   delete Calculate;
   delete TempKQ;
}

void
QueryVarStudyDialog::CmCheckRange()
{
   char  Buffer[25] = "";
   int tmpsteps,error=0;
   double tmpstart,tmpend;

   if(Init)
   {
      GetDlgItemText(ID_LGSTART, Buffer, sig);
      tmpstart = atof(Buffer);
      GetDlgItemText(ID_LGEND, Buffer, sig);
      tmpend = atof(Buffer);
      GetDlgItemText(ID_NSTEPS, Buffer, sig);
      tmpsteps = atoi(Buffer);

      if(XGain->GetCheck() == BF_CHECKED)
      {
         TempKQ->Start(1)=tmpstart;
         TempKQ->End(1)=tmpend;
         TempKQ->Steps(1)=tmpsteps;
         if(tmpstart-tmpend!=0.0&&tmpsteps!=0)
         {
            gcvt((tmpend-tmpstart)/tmpsteps,4,Buffer);
            SetDlgItemText(ID_LGSS, Buffer);
            Calculate->EnableWindow(1);
         }
         else
         {
            SetDlgItemText(ID_LGSS, "ERROR");
            Calculate->EnableWindow(0);
            if(tmpsteps==0) error=1;
         }
         strcpy(TempKQ->LgParam,"X Gain Variation");
         TempKQ->LgActive=1;
      }
      else if (ZGain->GetCheck() == BF_CHECKED)
      {
         TempKQ->Start(2)=tmpstart;
         TempKQ->End(2)=tmpend;
         TempKQ->Steps(2)=tmpsteps;
         if(tmpstart-tmpend!=0.0&&tmpsteps!=0)
      gcvt((tmpend-tmpstart)/tmpsteps,4,Buffer);
      SetDlgItemText(ID_LGSS, Buffer);
      Calculate->EnableWindow(1);
   }
   else
   {
      SetDlgItemText(ID_LGSS, "ERROR");
      Calculate->EnableWindow(0);
      if(tmpsteps==0) error=1;
   }
   strcpy(TempKQ->LgParam,"Z Gain Variation");
   TempKQ->LgActive=2;
}
else if (MGain->GetCheck() == BF_CHECKED)
{
   TempKQ->Start(3)=tmpstart;
   TempKQ->End(3)=tmpend;
   TempKQ->Steps(3)=tmpsteps;
   if(tmpstart-tmpend!=0.0&&tmpsteps!=0)
   {
      gcvt((tmpend-tmpstart)/tmpsteps,4,Buffer);
      SetDlgItemText(ID_LGSS, Buffer);
      Calculate->EnableWindow(1);
   }
   else
   {
      SetDlgItemText(ID_LGSS, "ERROR");
      Calculate->EnableWindow(0);
      if(tmpsteps==0) error=1;
   }
   strcpy(TempKQ->LgParam,"Pitch Gain Variation");
   TempKQ->LgActive=3;
}


GetDlgItemText(ID_LTSTART, Buffer, sig);
tmpstart = atof(Buffer);
GetDlgItemText(ID_LTEND, Buffer, sig);
tmpend = atof(Buffer);

if(YGain->GetCheck() == BF_CHECKED)
{
   TempKQ->Start(4)=tmpstart;
   TempKQ->End(4)=tmpend;
   if(tmpstart-tmpend!=0.0&&error!=1)
   {
      gcvt((tmpend-tmpstart)/tmpsteps,4,Buffer);
      SetDlgItemText(ID_LTSS, Buffer);
   }
   else
   {
      SetDlgItemText(ID_LTSS, "ERROR");
      Calculate->EnableWindow(0);
   }
   strcpy(TempKQ->LtParam,"Y Gain Variation");
   TempKQ->LtActive=4;
}
else if (LGain->GetCheck() == BF_CHECKED)
{
   TempKQ->Start(5)=tmpstart;
   TempKQ->End(5)=tmpend;
   if(tmpstart-tmpend!=0.0&&error!=1)
   {
      gcvt((tmpend-tmpstart)/tmpsteps,4,Buffer);
      SetDlgItemText(ID_LTSS, Buffer);
   }
```

```
          {
            SetDlgItemText(ID_LTSS, "ERROR");
            Calculate->EnableWindow(0);
          }
          strcpy(TempKQ->LtParam,"Roll Gain Variation");
          TempKQ->LtActive=5;
        }
        else if (NGain->GetCheck() == BF_CHECKED)
        {
          TempKQ->Start(6)=tmpstart;
          TempKQ->End(6)=tmpend;
          if(tmpstart-tmpend!=0.0&&error!=1)
          {
            gcvt((tmpend-tmpstart)/tmpsteps,4,Buffer);
            SetDlgItemText(ID_LTSS, Buffer);
          }
          else
          {
            SetDlgItemText(ID_LTSS, "ERROR");
            Calculate->EnableWindow(0);
          }
          strcpy(TempKQ->LtParam,"Yaw Gain Variation");
          TempKQ->LtActive=6;
        }
        else if (VertSpan->GetCheck() == BF_CHECKED)
        {
          if(tmpstart>=0.0)
            TempKQ->Start(7)=tmpstart;
          if(tmpend>=0.0)
            TempKQ->End(7)=tmpend;
          if(tmpstart-tmpend!=0.0&&error!=1)
          {
            gcvt((tmpend-tmpstart)/tmpsteps,4,Buffer);
            SetDlgItemText(ID_LTSS, Buffer);
          }
          else
          {
            SetDlgItemText(ID_LTSS, "ERROR");
            Calculate->EnableWindow(0);
          }
          strcpy(TempKQ->LtParam,"Vert. Stab. Span Variation");
          TempKQ->LtActive=7;
        }
        else if (VertX->GetCheck() == BF_CHECKED)
        {
          TempKQ->Start(8)=tmpstart;
          TempKQ->End(8)=tmpend;
          if(tmpstart-tmpend!=0.0&&error!=1)
          {
            gcvt((tmpend-tmpstart)/tmpsteps,4,Buffer);
            SetDlgItemText(ID_LTSS, Buffer);
          }
          else
          {
            SetDlgItemText(ID_LTSS, "ERROR");
            Calculate->EnableWindow(0);
          }
          strcpy(TempKQ->LtParam,"Vert. Stab. Position Variation");
          TempKQ->LtActive=8;
        }
    }
}

void
QueryVarStudyDialog::CmUpdate(UINT)
{
```

```
int tmpsteps,error=0;

Init = 0;

//*****************************
//Longitudinal Updates
//*****************************
if (XGain->GetCheck() == BF_CHECKED)
{
   gcvt(TempKQ->Start(1),sig,Buffer);
   SetDlgItemText(ID_LGSTART, Buffer);
   gcvt(TempKQ->End(1),sig,Buffer);
   SetDlgItemText(ID_LGEND, Buffer);
   gcvt(TempKQ->Steps(1),sig,Buffer);
   SetDlgItemText(ID_NSTEPS, Buffer);
   tmpsteps=TempKQ->Steps(1);
   if(TempKQ->Start(1)-TempKQ->End(1)!=0.0&&TempKQ->Steps(1)!=0)
   {
      gcvt((TempKQ->End(1)-TempKQ->Start(1))/TempKQ->Steps(1),4,Buffer);
      SetDlgItemText(ID_LGSS, Buffer);
      Calculate->EnableWindow(1);
   }
   else
   {
      SetDlgItemText(ID_LGSS, "ERROR");
      Calculate->EnableWindow(0);
      if(tmpsteps==0) error=1;
   }
}
else if (ZGain->GetCheck() == BF_CHECKED)
{
   gcvt(TempKQ->Start(2),sig,Buffer);
   SetDlgItemText(ID_LGSTART, Buffer);
   gcvt(TempKQ->End(2),sig,Buffer);
   SetDlgItemText(ID_LGEND, Buffer);
   gcvt(TempKQ->Steps(2),sig,Buffer);
   SetDlgItemText(ID_NSTEPS, Buffer);
   tmpsteps=TempKQ->Steps(2);
   if(TempKQ->Start(2)-TempKQ->End(2)!=0.0&&TempKQ->Steps(2)!=0)
   {
      gcvt((TempKQ->End(2)-TempKQ->Start(2))/TempKQ->Steps(2),4,Buffer);
      SetDlgItemText(ID_LGSS, Buffer);
      Calculate->EnableWindow(1);
   }
   else
   {
      SetDlgItemText(ID_LGSS, "ERROR");
      Calculate->EnableWindow(0);
      if(tmpsteps==0) error=1;
   }
}
else if (MGain->GetCheck() == BF_CHECKED)
{
   gcvt(TempKQ->Start(3),sig,Buffer);
   SetDlgItemText(ID_LGSTART, Buffer);
   gcvt(TempKQ->End(3),sig,Buffer);
   SetDlgItemText(ID_LGEND, Buffer);
   gcvt(TempKQ->Steps(3),sig,Buffer);
   SetDlgItemText(ID_NSTEPS, Buffer);
   tmpsteps=TempKQ->Steps(3);
   if(TempKQ->Start(3)-TempKQ->End(3)!=0.0&&TempKQ->Steps(3)!=0)
   {
      gcvt((TempKQ->End(3)-TempKQ->Start(3))/TempKQ->Steps(3),4,Buffer);
      SetDlgItemText(ID_LGSS, Buffer);
      Calculate->EnableWindow(1);
   }
```

```
      {
         SetDlgItemText(ID_LGSS, "ERROR");
         Calculate->EnableWindow(0);
         if(tmpsteps==0) error=1;
      }
   }

   //******************************
   //Lateral Updates
   //******************************
   if (YGain->GetCheck() == BF_CHECKED)
   {
      gcvt(TempKQ->Start(4),sig,Buffer);
      SetDlgItemText(ID_LTSTART, Buffer);
      gcvt(TempKQ->End(4),sig,Buffer);
      SetDlgItemText(ID_LTEND, Buffer);
      if(TempKQ->Start(4)-TempKQ->End(4)!=0.0&&error!=1)
      {
         gcvt((TempKQ->End(4)-TempKQ->Start(4))/tmpsteps,4,Buffer);
         SetDlgItemText(ID_LTSS, Buffer);
      }
      else
      {
         SetDlgItemText(ID_LTSS, "ERROR");
         Calculate->EnableWindow(0);
      }
   }
   else if (LGain->GetCheck() == BF_CHECKED)
   {
      gcvt(TempKQ->Start(5),sig,Buffer);
      SetDlgItemText(ID_LTSTART, Buffer);
      gcvt(TempKQ->End(5),sig,Buffer);
      SetDlgItemText(ID_LTEND, Buffer);
      if(TempKQ->Start(5)-TempKQ->End(5)!=0.0&&error!=1)
      {
         gcvt((TempKQ->End(5)-TempKQ->Start(5))/tmpsteps,4,Buffer);
         SetDlgItemText(ID_LTSS, Buffer);
      }
      else
      {
         SetDlgItemText(ID_LTSS, "ERROR");
         Calculate->EnableWindow(0);
      }
   }else if (NGain->GetCheck() == BF_CHECKED)
   {
      gcvt(TempKQ->Start(6),sig,Buffer);
      SetDlgItemText(ID_LTSTART, Buffer);
      gcvt(TempKQ->End(6),sig,Buffer);
      SetDlgItemText(ID_LTEND, Buffer);
      if(TempKQ->Start(6)-TempKQ->End(6)!=0.0&&error!=1)
      {
         gcvt((TempKQ->End(6)-TempKQ->Start(6))/tmpsteps,4,Buffer);
         SetDlgItemText(ID_LTSS, Buffer);
      }
      else
      {
         SetDlgItemText(ID_LTSS, "ERROR");
         Calculate->EnableWindow(0);
      }
   }else if (VertSpan->GetCheck() == BF_CHECKED)
   {
      gcvt(TempKQ->Start(7),sig,Buffer);
      SetDlgItemText(ID_LTSTART, Buffer);
      gcvt(TempKQ->End(7),sig,Buffer);
      SetDlgItemText(ID_LTEND, Buffer);
      if(TempKQ->Start(7)-TempKQ->End(7)!=0.0&&error!=1)
```

```
         gcvt((TempKQ->End(7)-TempKQ->Start(7))/tmpsteps,4,Buffer);
         SetDlgItemText(ID_LTSS, Buffer);
      }
      else
      {
         SetDlgItemText(ID_LTSS, "ERROR");
         Calculate->EnableWindow(0);
      }
   }else if (VertX->GetCheck() == BF_CHECKED)
   {
      gcvt(TempKQ->Start(8),sig,Buffer);
      SetDlgItemText(ID_LTSTART, Buffer);
      gcvt(TempKQ->End(8),sig,Buffer);
      SetDlgItemText(ID_LTEND, Buffer);
      if(TempKQ->Start(8)-TempKQ->End(8)!=0.0&&error!=1)
      {
         gcvt((TempKQ->End(8)-TempKQ->Start(8))/tmpsteps,4,Buffer);
         SetDlgItemText(ID_LTSS, Buffer);
      }
      else
      {
         SetDlgItemText(ID_LTSS, "ERROR");
         Calculate->EnableWindow(0);
      }
   }
   Init = 1;
   CmCheckRange();
}

//
// sets and gets the values of the items (controls) of the input dialog
//
void
QueryVarStudyDialog::TransferData(TTransferDirection direction)
{
   if (direction == tdSetData)
   {

   }
   else if (direction == tdGetData)
   {
      *KQRef = *TempKQ;
      KQRef->SetArraySize(TempKD->N);
   }
}

//
// sets the values of the items(controls) of the input dialog
//
void
QueryVarStudyDialog::SetupWindow()
{
   TDialog::SetupWindow();
   SendDlgItemMessage(ID_LGSTART, EM_LIMITTEXT, sig - 1, 0);
   SendDlgItemMessage(ID_LGEND, EM_LIMITTEXT, sig - 1, 0);
   SendDlgItemMessage(ID_LTSTART, EM_LIMITTEXT, sig - 1, 0);
   SendDlgItemMessage(ID_LTEND, EM_LIMITTEXT, sig - 1, 0);
   SendDlgItemMessage(ID_NSTEPS, EM_LIMITTEXT, sig - 1, 0);
   SendDlgItemMessage(ID_LGSS, EM_LIMITTEXT, sig - 1, 0);
   SendDlgItemMessage(ID_LTSS, EM_LIMITTEXT, sig - 1, 0);

   switch(KQRef->LgActive)
   {
      case 1:
         XGain->SetCheck(BF_CHECKED);
         break;
         ZGain->SetCheck(BF_CHECKED);
         break;
      case 3:
         MGain->SetCheck(BF_CHECKED);
         break;
   }

   switch(KQRef->LtActive)
   {
      case 4:
         YGain->SetCheck(BF_CHECKED);
         break;
      case 5:
         LGain->SetCheck(BF_CHECKED);
         break;
      case 6:
         NGain->SetCheck(BF_CHECKED);
         break;
      case 7:
         VertSpan->SetCheck(BF_CHECKED);
         break;
      case 8:
         VertX->SetCheck(BF_CHECKED);
         break;
   }
   Init=1;
   CmUpdate(ID_XGAIN);
}


//****************************************************************

StabilityDerivsDialog::StabilityDerivsDialog(TWindow*       parent,
                       KiteData*       KD,
                       TModule*        module,
                       TValidator*     validator)
:
  TWindow(parent, stabilityderivstitle, module),
  TDialog(parent, IDD_STABILITYDERIVSDIALOG, module)
{
  TempKD = KD;

  SetCaption(stabilityderivstitle);
  if (validator){
    new TEdit(this,ID_CXU)->SetValidator(validator);
    new TEdit(this,ID_CXALPHA)->SetValidator(validator);
    new TEdit(this,ID_CXQ)->SetValidator(validator);
    new TEdit(this,ID_CZU)->SetValidator(validator);
    new TEdit(this,ID_CZALPHA)->SetValidator(validator);
    new TEdit(this,ID_CZQ)->SetValidator(validator);
    new TEdit(this,ID_CZALPHADOT)->SetValidator(validator);
    new TEdit(this,ID_CMU)->SetValidator(validator);
    new TEdit(this,ID_CMALPHA)->SetValidator(validator);
    new TEdit(this,ID_CMQ)->SetValidator(validator);
    new TEdit(this,ID_CMALPHADOT)->SetValidator(validator);
    new TEdit(this,ID_CYBETA)->SetValidator(validator);
    new TEdit(this,ID_CYR)->SetValidator(validator);
    new TEdit(this,ID_CYP)->SetValidator(validator);
    new TEdit(this,ID_CNBETA)->SetValidator(validator);
    new TEdit(this,ID_CNR)->SetValidator(validator);
    new TEdit(this,ID_CNP)->SetValidator(validator);
    new TEdit(this,ID_CLBETA)->SetValidator(validator);
    new TEdit(this,ID_CLR)->SetValidator(validator);
    new TEdit(this,ID_CLP)->SetValidator(validator);
  }
```

```
StabilityDerivsDialog::~StabilityDerivsDialog()
{

}

//
// sets and gets the values of the items (controls) of the input dialog
//
void
StabilityDerivsDialog::TransferData(TTransferDirection direction)
{
   char  Buffer[25] = "";
   if (direction == tdSetData) {
      gcvt(TempKD->Cxu,sig-7,Buffer);
      SetDlgItemText(ID_CXU, Buffer);
      gcvt(TempKD->Cxalpha,sig-7,Buffer);
      SetDlgItemText(ID_CXALPHA, Buffer);
      gcvt(TempKD->Cxqls,sig-7,Buffer);
      SetDlgItemText(ID_CXQ, Buffer);
      gcvt(TempKD->Czu,sig-7,Buffer);
      SetDlgItemText(ID_CZU, Buffer);
      gcvt(TempKD->Czalpha,sig-7,Buffer);
      SetDlgItemText(ID_CZALPHA, Buffer);
      gcvt(TempKD->Czq,sig-7,Buffer);
      SetDlgItemText(ID_CZQ, Buffer);
      gcvt(TempKD->Czadot,sig-7,Buffer);
      SetDlgItemText(ID_CZALPHADOT, Buffer);
      gcvt(TempKD->Cmu,sig-7,Buffer);
      SetDlgItemText(ID_CMU, Buffer);
      gcvt(TempKD->Cmalpha,sig-7,Buffer);
      SetDlgItemText(ID_CMALPHA, Buffer);
      gcvt(TempKD->Cmq,sig-7,Buffer);
      SetDlgItemText(ID_CMQ, Buffer);
      gcvt(TempKD->Cmadot,sig-7,Buffer);
      SetDlgItemText(ID_CMALPHADOT, Buffer);
      gcvt(TempKD->Cyb,sig-7,Buffer);
      SetDlgItemText(ID_CYBETA, Buffer);
      gcvt(TempKD->Cyr,sig-7,Buffer);
      SetDlgItemText(ID_CYR, Buffer);
      gcvt(TempKD->Cyp,sig-7,Buffer);
      SetDlgItemText(ID_CYP, Buffer);
      gcvt(TempKD->CNb,sig-7,Buffer);
      SetDlgItemText(ID_CNBETA, Buffer);
      gcvt(TempKD->CNr,sig-7,Buffer);
      SetDlgItemText(ID_CNR, Buffer);
      gcvt(TempKD->CNp,sig-7,Buffer);
      SetDlgItemText(ID_CNP, Buffer);
      gcvt(TempKD->CLb,sig-7,Buffer);
      SetDlgItemText(ID_CLBETA, Buffer);
      gcvt(TempKD->CLr,sig-7,Buffer);
      SetDlgItemText(ID_CLR, Buffer);
      gcvt(TempKD->CLp,sig-7,Buffer);
      SetDlgItemText(ID_CLP, Buffer);
   }
}

//
// sets the values of the items(controls) of the input dialog
//
void
StabilityDerivsDialog::SetupWindow()
{
  TDialog::SetupWindow();
  SendDlgItemMessage(ID_CXU, EM_LIMITTEXT, sig - 7, 0);
  SendDlgItemMessage(ID_CXALPHA, EM_LIMITTEXT, sig - 7, 0);
  SendDlgItemMessage(ID_CZO, EM_LIMITTEXT, sig - 7, 0);
  SendDlgItemMessage(ID_CZALPHA, EM_LIMITTEXT, sig - 7, 0);
  SendDlgItemMessage(ID_CZQ, EM_LIMITTEXT, sig - 7, 0);
  SendDlgItemMessage(ID_CZALPHADOT, EM_LIMITTEXT, sig - 7, 0);
  SendDlgItemMessage(ID_CMU, EM_LIMITTEXT, sig - 7, 0);
  SendDlgItemMessage(ID_CMALPHA, EM_LIMITTEXT, sig - 7, 0);
  SendDlgItemMessage(ID_CMQ, EM_LIMITTEXT, sig - 7, 0);
  SendDlgItemMessage(ID_CMALPHADOT, EM_LIMITTEXT, sig - 7, 0);
  SendDlgItemMessage(ID_CYBETA, EM_LIMITTEXT, sig - 7, 0);
  SendDlgItemMessage(ID_CYR, EM_LIMITTEXT, sig - 7, 0);
  SendDlgItemMessage(ID_CYP, EM_LIMITTEXT, sig - 7, 0);
  SendDlgItemMessage(ID_CNBETA, EM_LIMITTEXT, sig - 7, 0);
  SendDlgItemMessage(ID_CNR, EM_LIMITTEXT, sig - 7, 0);
  SendDlgItemMessage(ID_CNP, EM_LIMITTEXT, sig - 7, 0);
  SendDlgItemMessage(ID_CLBETA, EM_LIMITTEXT, sig - 7, 0);
  SendDlgItemMessage(ID_CLR, EM_LIMITTEXT, sig - 7, 0);
  SendDlgItemMessage(ID_CLP, EM_LIMITTEXT, sig - 7, 0);
}

//**************************************************************
DEFINE_RESPONSE_TABLE1(RLScaleDialog, TDialog)
  EV_COMMAND(ID_YMAXLGSET, CmUpdateTxt),
  EV_COMMAND(ID_XMAXLGSET, CmUpdateTxt),
  EV_COMMAND(ID_XMINLGSET, CmUpdateTxt),
  EV_COMMAND(ID_YMAXLTSET, CmUpdateTxt),
  EV_COMMAND(ID_XMAXLTSET, CmUpdateTxt),
  EV_COMMAND(ID_XMINLTSET, CmUpdateTxt),
END_RESPONSE_TABLE;

RLScaleDialog::RLScaleDialog(TWindow*        parent,
                              Scale*            LGScale,
                              Scale*            LTScale,
                              TModule*          module,
                              TValidator*       validator)
:
  TWindow(parent, RLScaletitle, module),
  TDialog(parent, IDD_RLSDIALOG, module)
{
  LG = LGScale;
  LT = LTScale;
  SetCaption(RLScaletitle);

  ymaxlgSet = new TCheckBox(this, ID_YMAXLGSET);
  xmaxlgSet = new TCheckBox(this, ID_XMAXLGSET);
  xminlgSet = new TCheckBox(this, ID_XMINLGSET);
  ymaxltSet = new TCheckBox(this, ID_YMAXLTSET);
  xmaxltSet = new TCheckBox(this, ID_XMAXLTSET);
  xminltSet = new TCheckBox(this, ID_XMINLTSET);
  ymaxlg = new TEdit(this,ID_YMAXLG);
  ymaxlg->SetValidator(validator);
  xmaxlg = new TEdit(this,ID_XMAXLG);
  xmaxlg->SetValidator(validator);
  xminlg = new TEdit(this,ID_XMINLG);
  xminlg->SetValidator(validator);
  ymaxlt = new TEdit(this,ID_YMAXLT);
  ymaxlt->SetValidator(validator);
  xmaxlt = new TEdit(this,ID_XMAXLT);
  xmaxlt->SetValidator(validator);
  xminlt = new TEdit(this,ID_XMINLT);
  xminlt->SetValidator(validator);
  ymaxlgtxt = new TStatic(this,ID_YMAXLGTXT);
  xmaxlgtxt = new TStatic(this,ID_XMAXLGTXT);
  xminlgtxt = new TStatic(this,ID_XMINLGTXT);
  ymaxlttxt = new TStatic(this,ID_YMAXLTTXT);
  xmaxlttxt = new TStatic(this,ID_XMAXLTTXT);
  xminlttxt = new TStatic(this,ID_XMINLTTXT);
```

```
}

RLScaleDialog::~RLScaleDialog()
{
   delete ymaxlgSet;
   delete xmaxlgSet;
   delete xminlgSet;
   delete ymaxltSet;
   delete xmaxltSet;
   delete xminltSet;
   delete ymaxlg;
   delete xmaxlg;
   delete xminlg;
   delete ymaxlt;
   delete xmaxlt;
   delete xminlt;
   delete ymaxlgtxt;
   delete xmaxlgtxt;
   delete xminlgtxt;
   delete ymaxlttxt;
   delete xmaxlttxt;
   delete xminlttxt;
}

void
RLScaleDialog::CmUpdateTxt()
{
   if (ymaxlgSet->GetCheck() == BF_CHECKED)
   {
      ymaxlg->EnableWindow(1);
      ymaxlgtxt->EnableWindow(1);
   }
   else
   {
      ymaxlg->EnableWindow(0);
      ymaxlgtxt->EnableWindow(0);
   }
   if (xmaxlgSet->GetCheck() == BF_CHECKED)
   {
      xmaxlg->EnableWindow(1);
      xmaxlgtxt->EnableWindow(1);
   }
   else
   {
      xmaxlg->EnableWindow(0);
      xmaxlgtxt->EnableWindow(0);
   }
   if (xminlgSet->GetCheck() == BF_CHECKED)
   {
      xminlg->EnableWindow(1);
      xminlgtxt->EnableWindow(1);
   }
   else
   {
      xminlg->EnableWindow(0);
      xminlgtxt->EnableWindow(0);
   }
   if (ymaxltSet->GetCheck() == BF_CHECKED)
   {
      ymaxlt->EnableWindow(1);
      ymaxlttxt->EnableWindow(1);
   }
   else
   {
      ymaxlt->EnableWindow(0);
      ymaxlttxt->EnableWindow(0);
   }
```

```
    {
       xmaxlt->EnableWindow(1);
       xmaxlttxt->EnableWindow(1);
    }
    else
    {
       xmaxlt->EnableWindow(0);
       xmaxlttxt->EnableWindow(0);
    }
    if (xminltSet->GetCheck() == BF_CHECKED)
    {
       xminlt->EnableWindow(1);
       xminlttxt->EnableWindow(1);
    }
    else
    {
       xminlt->EnableWindow(0);
       xminlttxt->EnableWindow(0);
    }
}


//
// sets and gets the values of the items (controls) of the input dialog
//
void
RLScaleDialog::TransferData(TTransferDirection direction)
{
    char  Buffer[25] = "";
    double tempdbl;
    if (direction == tdSetData) {
       if(LG->YMaxSet)
          ymaxlgSet->SetCheck(BF_CHECKED);
       if(LG->XMaxSet)
          xmaxlgSet->SetCheck(BF_CHECKED);
       if(LG->XMinSet)
          xminlgSet->SetCheck(BF_CHECKED);
       if(LT->YMaxSet)
          ymaxltSet->SetCheck(BF_CHECKED);
       if(LT->XMaxSet)
          xmaxltSet->SetCheck(BF_CHECKED);
       if(LT->XMinSet)
          xminltSet->SetCheck(BF_CHECKED);
       gcvt(LG->YMax,sig,Buffer);
       SetDlgItemText(ID_YMAXLG, Buffer);
       gcvt(LG->XMax,sig,Buffer);
       SetDlgItemText(ID_XMAXLG, Buffer);
       gcvt(LG->XMin,sig,Buffer);
       SetDlgItemText(ID_XMINLG, Buffer);
       gcvt(LT->YMax,sig,Buffer);
       SetDlgItemText(ID_YMAXLT, Buffer);
       gcvt(LT->XMax,sig,Buffer);
       SetDlgItemText(ID_XMAXLT, Buffer);
       gcvt(LT->XMin,sig,Buffer);
       SetDlgItemText(ID_XMINLT, Buffer);
    }
    else if (direction == tdGetData) {
       GetDlgItemText(ID_YMAXLG, Buffer, sig);
       tempdbl = atof(Buffer);
       if(tempdbl>0.0) LG->YMax=tempdbl;
       GetDlgItemText(ID_XMAXLG, Buffer, sig);
       tempdbl = atof(Buffer);
       if(tempdbl>0.0) LG->XMax=tempdbl;
       GetDlgItemText(ID_XMINLG, Buffer, sig);
       tempdbl = atof(Buffer);
       if(tempdbl<0.0) LG->XMin=tempdbl;
```

```
      if(tempdbl>0.0) LT->YMax=tempdbl;
      GetDlgItemText(ID_XMAXLT, Buffer, sig);
      tempdbl = atof(Buffer);
      if(tempdbl>0.0) LT->XMax=tempdbl;
      GetDlgItemText(ID_XMINLT, Buffer, sig);
      tempdbl = atof(Buffer);
      if(tempdbl<0.0) LT->XMin=tempdbl;
      if(ymaxlgSet->GetCheck()==BF_CHECKED&&LG->YMax>0.0)
         LG->YMaxSet=true;
      else
         LG->YMaxSet=false;
      if(xmaxlgSet->GetCheck()==BF_CHECKED&&LG->XMax>0.0)
         LG->XMaxSet=true;
      else
         LG->XMaxSet=false;
      if(xminlgSet->GetCheck()==BF_CHECKED&&LG->XMin<0.0)
         LG->XMinSet=true;
      else
         LG->XMinSet=false;
      if(ymaxltSet->GetCheck()==BF_CHECKED&&LT->YMax>0.0)
         LT->YMaxSet=true;
      else
         LT->YMaxSet=false;
      if(xmaxltSet->GetCheck()==BF_CHECKED&&LT->XMax>0.0)
         LT->XMaxSet=true;
      else
         LT->XMaxSet=false;
      if(xminltSet->GetCheck()==BF_CHECKED&&LT->XMin<0.0)
         LT->XMinSet=true;
      else
         LT->XMinSet=false;
   }
}

//
// sets the values of the items(controls) of the input dialog
//
void
RLScaleDialog::SetupWindow()
{
  TDialog::SetupWindow();
  SendDlgItemMessage(ID_YMAXLG, EM_LIMITTEXT, sig - 1, 0);
  SendDlgItemMessage(ID_XMAXLG, EM_LIMITTEXT, sig - 1, 0);
  SendDlgItemMessage(ID_XMINLG, EM_LIMITTEXT, sig - 1, 0);
  SendDlgItemMessage(ID_YMAXLT, EM_LIMITTEXT, sig - 1, 0);
  SendDlgItemMessage(ID_XMAXLT, EM_LIMITTEXT, sig - 1, 0);
  SendDlgItemMessage(ID_XMINLT, EM_LIMITTEXT, sig - 1, 0);
  CmUpdateTxt();
}

//*****************************************************************
DEFINE_RESPONSE_TABLE1(TimeStepDialog, TDialog)
   EV_EN_CHANGE(ID_TIMESTEP,CmUpdateTxt),
   EV_EN_CHANGE(ID_TOTALTIME,CmUpdateTxt),
   EV_COMMAND(ID_XTSSET, CmUpdateTxt),
   EV_COMMAND(ID_ZTSSET, CmUpdateTxt),
   EV_COMMAND(ID_MTSSET, CmUpdateTxt),
   EV_COMMAND(ID_YTSSET, CmUpdateTxt),
   EV_COMMAND(ID_LTSSET, CmUpdateTxt),
   EV_COMMAND(ID_NTSSET, CmUpdateTxt),
END_RESPONSE_TABLE;

TimeStepDialog::TimeStepDialog(TWindow*         parent,
                               KiteData*        KD,
                               TModule*         module,
  TWindow(parent, TimeSteptitle, module),
  TDialog(parent, IDD_TIMESTEPDIALOG, module)
{
   TempKD = KD;
   SetCaption(TimeSteptitle);

   xtsSet = new TCheckBox(this, ID_XTSSET);
   ztsSet = new TCheckBox(this, ID_ZTSSET);
   mtsSet = new TCheckBox(this, ID_MTSSET);
   ytsSet = new TCheckBox(this, ID_YTSSET);
   ltsSet = new TCheckBox(this, ID_LTSSET);
   ntsSet = new TCheckBox(this, ID_NTSSET);
   timestep = new TEdit(this,ID_TIMESTEP);
   timestep->SetValidator(validator);
   totaltime = new TEdit(this,ID_TOTALTIME);
   totaltime->SetValidator(validator);
   xts = new TEdit(this,ID_XTS);
   xts->SetValidator(validator);
   zts = new TEdit(this,ID_ZTS);
   zts->SetValidator(validator);
   mts = new TEdit(this,ID_MTS);
   mts->SetValidator(validator);
   xtsdur = new TEdit(this,ID_XTSDUR);
   xtsdur->SetValidator(validator);
   ztsdur = new TEdit(this,ID_ZTSDUR);
   ztsdur->SetValidator(validator);
   mtsdur = new TEdit(this,ID_MTSDUR);
   mtsdur->SetValidator(validator);
   xtstxt1 = new TStatic(this,ID_XTSTXT1);
   ztstxt1 = new TStatic(this,ID_ZTSTXT1);
   mtstxt1 = new TStatic(this,ID_MTSTXT1);
   xtstxt2 = new TStatic(this,ID_XTSTXT2);
   ztstxt2 = new TStatic(this,ID_ZTSTXT2);
   mtstxt2 = new TStatic(this,ID_MTSTXT2);
   xtstxt3 = new TStatic(this,ID_XTSTXT3);
   ztstxt3 = new TStatic(this,ID_ZTSTXT3);
   mtstxt3 = new TStatic(this,ID_MTSTXT3);
   xtstxt4 = new TStatic(this,ID_XTSTXT4);
   ztstxt4 = new TStatic(this,ID_ZTSTXT4);
   mtstxt4 = new TStatic(this,ID_MTSTXT4);
   yts = new TEdit(this,ID_YTS);
   yts->SetValidator(validator);
   lts = new TEdit(this,ID_LTS);
   lts->SetValidator(validator);
   nts = new TEdit(this,ID_NTS);
   nts->SetValidator(validator);
   ytsdur = new TEdit(this,ID_YTSDUR);
   ytsdur->SetValidator(validator);
   ltsdur = new TEdit(this,ID_LTSDUR);
   ltsdur->SetValidator(validator);
   ntsdur = new TEdit(this,ID_NTSDUR);
   ntsdur->SetValidator(validator);
   ytstxt1 = new TStatic(this,ID_YTSTXT1);
   ltstxt1 = new TStatic(this,ID_LTSTXT1);
   ntstxt1 = new TStatic(this,ID_NTSTXT1);
   ytstxt2 = new TStatic(this,ID_YTSTXT2);
   ltstxt2 = new TStatic(this,ID_LTSTXT2);
   ntstxt2 = new TStatic(this,ID_NTSTXT2);
   ytstxt3 = new TStatic(this,ID_YTSTXT3);
   ltstxt3 = new TStatic(this,ID_LTSTXT3);
   ntstxt3 = new TStatic(this,ID_NTSTXT3);
   ytstxt4 = new TStatic(this,ID_YTSTXT4);
   ltstxt4 = new TStatic(this,ID_LTSTXT4);
   ntstxt4 = new TStatic(this,ID_NTSTXT4);
```

```
}

TimeStepDialog::~TimeStepDialog()
{
   delete xtsSet;
   delete ztsSet;
   delete mtsSet;
   delete ytsSet;
   delete ltsSet;
   delete ntsSet;
   delete timestep;
   delete totaltime;
   delete xts;
   delete zts;
   delete mts;
   delete xtsdur;
   delete ztsdur;
   delete mtsdur;
   delete xtstxt1;
   delete ztstxt1;
   delete mtstxt1;
   delete xtstxt2;
   delete ztstxt2;
   delete mtstxt2;
   delete xtstxt3;
   delete ztstxt3;
   delete mtstxt3;
   delete xtstxt4;
   delete ztstxt4;
   delete mtstxt4;
   delete yts;
   delete lts;
   delete nts;
   delete ytsdur;
   delete ltsdur;
   delete ntsdur;
   delete ytstxt1;
   delete ltstxt1;
   delete ntstxt1;
   delete ytstxt2;
   delete ltstxt2;
   delete ntstxt2;
   delete ytstxt3;
   delete ltstxt3;
   delete ntstxt3;
   delete ytstxt4;
   delete ltstxt4;
   delete ntstxt4;
   delete Calculate;
}

void
TimeStepDialog::CmUpdateTxt()
{
   double tmpts,tmptime;
   char  Buffer[25] = "";

   GetDlgItemText(ID_TIMESTEP, Buffer, sig);
   tmpts = atof(Buffer);
   GetDlgItemText(ID_TOTALTIME, Buffer, sig);
   tmptime = atof(Buffer);

   if(tmpts>0.0 && tmptime>0.0 && (xtsSet->GetCheck()==BF_CHECKED||
      ztsSet->GetCheck()==BF_CHECKED||mtsSet->GetCheck()==BF_CHECKED||
      ytsSet->GetCheck()==BF_CHECKED||ltsSet->GetCheck()==BF_CHECKED||
      ntsSet->GetCheck()==BF_CHECKED))
```

```
else
   Calculate->EnableWindow(0);

if (xtsSet->GetCheck() == BF_CHECKED && tmpts>0.0 && tmptime>0.0)
{
   xts->EnableWindow(1);
   xtsdur->EnableWindow(1);
   xtstxt1->EnableWindow(1);
   xtstxt2->EnableWindow(1);
   xtstxt3->EnableWindow(1);
   xtstxt4->EnableWindow(1);
}
else
{
   xts->EnableWindow(0);
   xtsdur->EnableWindow(0);
   xtstxt1->EnableWindow(0);
   xtstxt2->EnableWindow(0);
   xtstxt3->EnableWindow(0);
   xtstxt4->EnableWindow(0);
}
if (ztsSet->GetCheck() == BF_CHECKED && tmpts>0.0 && tmptime>0.0)
{
   zts->EnableWindow(1);
   ztsdur->EnableWindow(1);
   ztstxt1->EnableWindow(1);
   ztstxt2->EnableWindow(1);
   ztstxt3->EnableWindow(1);
   ztstxt4->EnableWindow(1);
}
else
{
   zts->EnableWindow(0);
   ztsdur->EnableWindow(0);
   ztstxt1->EnableWindow(0);
   ztstxt2->EnableWindow(0);
   ztstxt3->EnableWindow(0);
   ztstxt4->EnableWindow(0);
}
if (mtsSet->GetCheck() == BF_CHECKED && tmpts>0.0 && tmptime>0.0)
{
   mts->EnableWindow(1);
   mtsdur->EnableWindow(1);
   mtstxt1->EnableWindow(1);
   mtstxt2->EnableWindow(1);
   mtstxt3->EnableWindow(1);
   mtstxt4->EnableWindow(1);
}
else
{
   mts->EnableWindow(0);
   mtsdur->EnableWindow(0);
   mtstxt1->EnableWindow(0);
   mtstxt2->EnableWindow(0);
   mtstxt3->EnableWindow(0);
   mtstxt4->EnableWindow(0);
}
if (ytsSet->GetCheck() == BF_CHECKED && tmpts>0.0 && tmptime>0.0)
{
   yts->EnableWindow(1);
   ytsdur->EnableWindow(1);
   ytstxt1->EnableWindow(1);
   ytstxt2->EnableWindow(1);
   ytstxt3->EnableWindow(1);
   ytstxt4->EnableWindow(1);
}
```

```
      yts->EnableWindow(0);
      ytsdur->EnableWindow(0);
      ytstxt1->EnableWindow(0);
      ytstxt2->EnableWindow(0);
      ytstxt3->EnableWindow(0);
      ytstxt4->EnableWindow(0);
   }
   if (ltsSet->GetCheck() == BF_CHECKED && tmpts>0.0 && tmptime>0.0)
   {
      lts->EnableWindow(1);
      ltsdur->EnableWindow(1);
      ltstxt1->EnableWindow(1);
      ltstxt2->EnableWindow(1);
      ltstxt3->EnableWindow(1);
      ltstxt4->EnableWindow(1);
   }
   else
   {
      lts->EnableWindow(0);
      ltsdur->EnableWindow(0);
      ltstxt1->EnableWindow(0);
      ltstxt2->EnableWindow(0);
      ltstxt3->EnableWindow(0);
      ltstxt4->EnableWindow(0);
   }
   if (ntsSet->GetCheck() == BF_CHECKED && tmpts>0.0 && tmptime>0.0)
   {
      nts->EnableWindow(1);
      ntsdur->EnableWindow(1);
      ntstxt1->EnableWindow(1);
      ntstxt2->EnableWindow(1);
      ntstxt3->EnableWindow(1);
      ntstxt4->EnableWindow(1);
   }
   else
   {
      nts->EnableWindow(0);
      ntsdur->EnableWindow(0);
      ntstxt1->EnableWindow(0);
      ntstxt2->EnableWindow(0);
      ntstxt3->EnableWindow(0);
      ntstxt4->EnableWindow(0);
   }
}


//
// sets and gets the values of the items (controls) of the input dialog
//
void
TimeStepDialog::TransferData(TTransferDirection direction)
{
   char  Buffer[25] = "";
   double tempdbl;
   if (direction == tdSetData) {
      if(TempKD->XTSSet)
         xtsSet->SetCheck(BF_CHECKED);
      if(TempKD->ZTSSet)
         ztsSet->SetCheck(BF_CHECKED);
      if(TempKD->MTSSet)
         mtsSet->SetCheck(BF_CHECKED);
      if(TempKD->YTSSet)
         ytsSet->SetCheck(BF_CHECKED);
      if(TempKD->LTSSet)
         ltsSet->SetCheck(BF_CHECKED);
      hcbSet->SetCheck(BF_CHECKED);
   gcvt(TempKD->TimeStep,sig,Buffer);
   SetDlgItemText(ID_TIMESTEP, Buffer);
   gcvt(TempKD->TotalTime,sig,Buffer);
   SetDlgItemText(ID_TOTALTIME, Buffer);
   gcvt(TempKD->XTS,sig,Buffer);
   SetDlgItemText(ID_XTS, Buffer);
   gcvt(TempKD->ZTS,sig,Buffer);
   SetDlgItemText(ID_ZTS, Buffer);
   gcvt(TempKD->MTS,sig,Buffer);
   SetDlgItemText(ID_MTS, Buffer);
   gcvt(TempKD->XTSDur,sig,Buffer);
   SetDlgItemText(ID_XTSDUR, Buffer);
   gcvt(TempKD->ZTSDur,sig,Buffer);
   SetDlgItemText(ID_ZTSDUR, Buffer);
   gcvt(TempKD->MTSDur,sig,Buffer);
   SetDlgItemText(ID_MTSDUR, Buffer);
   gcvt(TempKD->YTS,sig,Buffer);
   SetDlgItemText(ID_YTS, Buffer);
   gcvt(TempKD->LTS,sig,Buffer);
   SetDlgItemText(ID_LTS, Buffer);
   gcvt(TempKD->NTS,sig,Buffer);
   SetDlgItemText(ID_NTS, Buffer);
   gcvt(TempKD->YTSDur,sig,Buffer);
   SetDlgItemText(ID_YTSDUR, Buffer);
   gcvt(TempKD->LTSDur,sig,Buffer);
   SetDlgItemText(ID_LTSDUR, Buffer);
   gcvt(TempKD->NTSDur,sig,Buffer);
   SetDlgItemText(ID_NTSDUR, Buffer);
}
else if (direction == tdGetData) {
   GetDlgItemText(ID_TIMESTEP, Buffer, sig);
   tempdbl = atof(Buffer);
   if(tempdbl>0.0) TempKD->TimeStep=tempdbl;
   GetDlgItemText(ID_TOTALTIME, Buffer, sig);
   tempdbl = atof(Buffer);
   if(tempdbl>0.0) TempKD->TotalTime=tempdbl;
   GetDlgItemText(ID_XTS, Buffer, sig);
   tempdbl = atof(Buffer);
   TempKD->XTS=tempdbl;
   GetDlgItemText(ID_ZTS, Buffer, sig);
   tempdbl = atof(Buffer);
   TempKD->ZTS=tempdbl;
   GetDlgItemText(ID_MTS, Buffer, sig);
   tempdbl = atof(Buffer);
   TempKD->MTS=tempdbl;
   GetDlgItemText(ID_XTSDUR, Buffer, sig);
   tempdbl = atof(Buffer);
   if(tempdbl>0.0) TempKD->XTSDur=tempdbl;
   GetDlgItemText(ID_ZTSDUR, Buffer, sig);
   tempdbl = atof(Buffer);
   if(tempdbl>0.0) TempKD->ZTSDur=tempdbl;
   GetDlgItemText(ID_MTSDUR, Buffer, sig);
   tempdbl = atof(Buffer);
   if(tempdbl>0.0) TempKD->MTSDur=tempdbl;
   GetDlgItemText(ID_YTS, Buffer, sig);
   tempdbl = atof(Buffer);
   TempKD->YTS=tempdbl;
   GetDlgItemText(ID_LTS, Buffer, sig);
   tempdbl = atof(Buffer);
   TempKD->LTS=tempdbl;
   GetDlgItemText(ID_NTS, Buffer, sig);
   tempdbl = atof(Buffer);
   TempKD->NTS=tempdbl;
   GetDlgItemText(ID_YTSDUR, Buffer, sig);
   tempdbl = atof(Buffer);
```

```
        GetDlgItemText(ID_LTSDUR, Buffer, sig);
        tempdbl = atof(Buffer);
        if(tempdbl>0.0) TempKD->LTSDur=tempdbl;
        GetDlgItemText(ID_NTSDUR, Buffer, sig);
        tempdbl = atof(Buffer);
        if(tempdbl>0.0) TempKD->NTSDur=tempdbl;

        if(xtsSet->GetCheck()==BF_CHECKED&&TempKD->XTSDur>0.0)
           TempKD->XTSSet=true;
        else
           TempKD->XTSSet=false;
        if(ztsSet->GetCheck()==BF_CHECKED&&TempKD->ZTSDur>0.0)
           TempKD->ZTSSet=true;
        else
           TempKD->ZTSSet=false;
        if(mtsSet->GetCheck()==BF_CHECKED&&TempKD->MTSDur>0.0)
           TempKD->MTSSet=true;
        else
           TempKD->MTSSet=false;
        if(ytsSet->GetCheck()==BF_CHECKED&&TempKD->YTSDur>0.0)
           TempKD->YTSSet=true;
        else
           TempKD->YTSSet=false;
        if(ltsSet->GetCheck()==BF_CHECKED&&TempKD->LTSDur>0.0)
           TempKD->LTSSet=true;
        else
           TempKD->LTSSet=false;
        if(ntsSet->GetCheck()==BF_CHECKED&&TempKD->NTSDur>0.0)
           TempKD->NTSSet=true;
        else
           TempKD->NTSSet=false;
    }
}

//
// sets the values of the items(controls) of the input dialog
//
void
TimeStepDialog::SetupWindow()
{
   TDialog::SetupWindow();
   SendDlgItemMessage(ID_TIMESTEP, EM_LIMITTEXT, sig - 1, 0);
   SendDlgItemMessage(ID_TOTALTIME, EM_LIMITTEXT, sig - 1, 0);
   SendDlgItemMessage(ID_XTS, EM_LIMITTEXT, sig - 1, 0);
   SendDlgItemMessage(ID_ZTS, EM_LIMITTEXT, sig - 1, 0);
   SendDlgItemMessage(ID_MTS, EM_LIMITTEXT, sig - 1, 0);
   SendDlgItemMessage(ID_XTSDUR, EM_LIMITTEXT, sig - 1, 0);
   SendDlgItemMessage(ID_ZTSDUR, EM_LIMITTEXT, sig - 1, 0);
   SendDlgItemMessage(ID_MTSDUR, EM_LIMITTEXT, sig - 1, 0);
   SendDlgItemMessage(ID_YTS, EM_LIMITTEXT, sig - 1, 0);
   SendDlgItemMessage(ID_LTS, EM_LIMITTEXT, sig - 1, 0);
   SendDlgItemMessage(ID_NTS, EM_LIMITTEXT, sig - 1, 0);
   SendDlgItemMessage(ID_YTSDUR, EM_LIMITTEXT, sig - 1, 0);
   SendDlgItemMessage(ID_LTSDUR, EM_LIMITTEXT, sig - 1, 0);
   SendDlgItemMessage(ID_NTSDUR, EM_LIMITTEXT, sig - 1, 0);
   CmUpdateTxt();
}

//*****************************************************************
DEFINE_RESPONSE_TABLE1(TSScaleDialog, TDialog)
   EV_COMMAND(ID_YMAXLGSET, CmUpdateTxt),
   EV_COMMAND(ID_YMINLGSET, CmUpdateTxt),
   EV_COMMAND(ID_YMAXLTSET, CmUpdateTxt),
   EV_COMMAND(ID_YMINLTSET, CmUpdateTxt),
END_RESPONSE_TABLE;
```

```
                              Scale*            LGScale,
                              Scale*            LTScale,
                              TModule*          module,
                              TValidator*       validator)
:
  TWindow(parent, TSScaletitle, module),
  TDialog(parent, IDD_TSSDIALOG, module)
{
  LG = LGScale;
  LT = LTScale;
  SetCaption(TSScaletitle);

  ymaxlgSet = new TCheckBox(this, ID_YMAXLGSET);
  yminlgSet = new TCheckBox(this, ID_YMINLGSET);
  ymaxltSet = new TCheckBox(this, ID_YMAXLTSET);
  yminltSet = new TCheckBox(this, ID_YMINLTSET);
  ymaxlg = new TEdit(this,ID_YMAXLG);
  ymaxlg->SetValidator(validator);
  yminlg = new TEdit(this,ID_YMINLG);
  yminlg->SetValidator(validator);
  ymaxlt = new TEdit(this,ID_YMAXLT);
  ymaxlt->SetValidator(validator);
  yminlt = new TEdit(this,ID_YMINLT);
  yminlt->SetValidator(validator);
  ymaxlgtxt = new TStatic(this,ID_YMAXLGTXT);
  yminlgtxt = new TStatic(this,ID_YMINLGTXT);
  ymaxlttxt = new TStatic(this,ID_YMAXLTTXT);
  yminlttxt = new TStatic(this,ID_YMINLTTXT);
}

TSScaleDialog::~TSScaleDialog()
{
   delete ymaxlgSet;
   delete yminlgSet;
   delete ymaxltSet;
   delete yminltSet;
   delete ymaxlg;
   delete yminlg;
   delete ymaxlt;
   delete yminlt;
   delete ymaxlgtxt;
   delete yminlgtxt;
   delete ymaxlttxt;
   delete yminlttxt;
}

void
TSScaleDialog::CmUpdateTxt()
{
   if (ymaxlgSet->GetCheck() == BF_CHECKED)
   {
      ymaxlg->EnableWindow(1);
      ymaxlgtxt->EnableWindow(1);
   }
   else
   {
      ymaxlg->EnableWindow(0);
      ymaxlgtxt->EnableWindow(0);
   }
   if (yminlgSet->GetCheck() == BF_CHECKED)
   {
      yminlg->EnableWindow(1);
      yminlgtxt->EnableWindow(1);
   }
   else
   {
         yminlgtxt->EnableWindow(0);
      }
      if (ymaxltSet->GetCheck() == BF_CHECKED)
      {
         ymaxlt->EnableWindow(1);
         ymaxlttxt->EnableWindow(1);
      }
      else
      {
         ymaxlt->EnableWindow(0);
         ymaxlttxt->EnableWindow(0);
      }
      if (yminltSet->GetCheck() == BF_CHECKED)
      {
         yminlt->EnableWindow(1);
         yminlttxt->EnableWindow(1);
      }
      else
      {
         yminlt->EnableWindow(0);
         yminlttxt->EnableWindow(0);
      }
   }


   //
   // sets and gets the values of the items (controls) of the input dialog
   //
   void
   TSScaleDialog::TransferData(TTransferDirection direction)
   {
      char  Buffer[25] = "";
      double tempdbl;
      if (direction == tdSetData) {
         if(LG->XMaxSet)
            ymaxlgSet->SetCheck(BF_CHECKED);
         if(LG->XMinSet)
            yminlgSet->SetCheck(BF_CHECKED);
         if(LT->XMaxSet)
            ymaxltSet->SetCheck(BF_CHECKED);
         if(LT->XMinSet)
            yminltSet->SetCheck(BF_CHECKED);
         gcvt(LG->XMax,sig,Buffer);
         SetDlgItemText(ID_YMAXLG, Buffer);
         gcvt(LG->XMin,sig,Buffer);
         SetDlgItemText(ID_YMINLG, Buffer);
         gcvt(LT->XMax,sig,Buffer);
         SetDlgItemText(ID_YMAXLT, Buffer);
         gcvt(LT->XMin,sig,Buffer);
         SetDlgItemText(ID_YMINLT, Buffer);
      }
      else if (direction == tdGetData) {
         GetDlgItemText(ID_YMAXLG, Buffer, sig);
         tempdbl = atof(Buffer);
         if(tempdbl>0.0) LG->XMax=tempdbl;
         GetDlgItemText(ID_YMINLG, Buffer, sig);
         tempdbl = atof(Buffer);
         if(tempdbl<0.0) LG->XMin=tempdbl;
         GetDlgItemText(ID_YMAXLT, Buffer, sig);
         tempdbl = atof(Buffer);
         if(tempdbl>0.0) LT->XMax=tempdbl;
         GetDlgItemText(ID_YMINLT, Buffer, sig);
         tempdbl = atof(Buffer);
         if(tempdbl<0.0) LT->XMin=tempdbl;
         if(ymaxlgSet->GetCheck()==BF_CHECKED&&LG->XMax>0.0)
            LG->XMaxSet=true;
            LG->XMaxSet=false;
         if(yminlgSet->GetCheck()==BF_CHECKED&&LG->XMin<0.0)
            LG->XMinSet=true;
         else
            LG->XMinSet=false;
         if(ymaxltSet->GetCheck()==BF_CHECKED&&LT->XMax>0.0)
            LT->XMaxSet=true;
         else
            LT->XMaxSet=false;
         if(yminltSet->GetCheck()==BF_CHECKED&&LT->XMin<0.0)
            LT->XMinSet=true;
         else
            LT->XMinSet=false;
      }
   }

   //
   // sets the values of the items(controls) of the input dialog
   //
   void
   TSScaleDialog::SetupWindow()
   {
     TDialog::SetupWindow();
     SendDlgItemMessage(ID_YMAXLG, EM_LIMITTEXT, sig - 1, 0);
     SendDlgItemMessage(ID_YMINLG, EM_LIMITTEXT, sig - 1, 0);
     SendDlgItemMessage(ID_YMAXLT, EM_LIMITTEXT, sig - 1, 0);
     SendDlgItemMessage(ID_YMINLT, EM_LIMITTEXT, sig - 1, 0);
     CmUpdateTxt();
   }
```

```
#ifndef _TRIMSTAT_H_
#define _TRIMSTAT_H_

#include <owl/scroller.h>
#include <owl/opensave.h>
#include <owl/printer.h>
#include <classlib/arrays.h>
#include <owl/checkbox.h>
#include <math.h>
#include <string.h>
#include <fstream.h>
#include "altitdef.h"
#include "suprkite.h"

//---------------------------------------------------------------------------
// TWindowPrintout

class TWindowPrintout : public TPrintout {
   public:
      TWindowPrintout(const char* title, TWindow* window);

      void GetDialogInfo(int& minPage, int& maxPage,
                         int& selFromPage, int& selToPage);
      void PrintPage(int page, TRect& rect, unsigned flags);
      void SetBanding(BOOL b) {Banding = b;}
      BOOL HasPage(int pageNumber) {return pageNumber == 1;}

   protected:
      TWindow* Window;
      BOOL     Scale;
};

//---------------------------------------------------------------------------
//TStateWindow
class TStateWindow : public TWindow {
  public:
    TStateWindow(TWindow* parent = 0);
   ~TStateWindow();
    void Paint(TDC&, BOOL, TRect&);
    int PixelWidth;
    int PixelHeight;

  protected:
    KiteData* KData;
    AltitudeData* AData;
    ControlGain* Control;
    KiteQuery* KQuery;
    Scale* LgScale;
    Scale* LtScale;
    Scale* TSLgScale;
    Scale* TSLtScale;
    TOpenSaveDialog::TData* KiteFile;
    TOpenSaveDialog::TData* AltitudeFile;
    bool IsDirty,IsNewFile,IsWindFile,IsTStateValid,IsSRootValid,ShowProf;
    bool HighSpeedTS,CheckEigVect,ShowRoots,IsRLocusValid,RootLocusOut;
    bool MatlabOut,Proceed,TimeStepOut,IsTimeStepValid,ShowTimeStep;
    double AltAcc,origMag;
    void AdjustScroller();

    // Override member function of TWindow
    bool CanClose();

    // Message response functions
    void EvSize(UINT sizeType, TSize&);
    void RLDataOut();
    void CmFileOpen();
    void CmFileSave();
    void CmFileSaveAs();
    void CmKiteGeneral();
    void CmKiteWing();
    void CmKiteEmpen();
    void CmKiteEmpen2();
    void CmKiteFuselage();
    void CmKiteControlGain();
    void CmRlplotScale();
    void CmTSplotScale();
    void CmAltitudeFile();
    void CmAltitudeSaveFile();
    void CmWindFunction();
    void CmCalcTState();
    void CmCalcStability();
    void CmCalcTimeStep();
    void CmVarStudy();
    void CmTStateResults();
    void CmStabilityResults();
    void CmStabilityDerivs();
    void CmToggleProfView();
    void CmToggleRootsView();
    void CmToggleTimeStepView();
    void CmToggleHighSpeedTS();
    void CmToggleCheckEigVect();
    void CmToggleMatlabOut();
    void CmToggleRootLocusOut();
    void CmToggleTimeStepOut();
    void CmPrintProfile();
    void CmFilePrinterSetup();
    void CmAbout();
    void SaveFile(TOpenSaveDialog::TData* file);
    void OpenFile(TOpenSaveDialog::TData* file);

    // Command-enabling functions
    void CeCalcStability(TCommandEnabler&);
    void CeCalcTimeStep(TCommandEnabler&);
    void CeVarStudy(TCommandEnabler&);
    void CeStabilityResults(TCommandEnabler&);
    void CeStabilityDerivs(TCommandEnabler&);
    void CeToggleProfView(TCommandEnabler&);
    void CeToggleRootsView(TCommandEnabler&);
    void CeToggleTimeStepView(TCommandEnabler&);
    void CeToggleHighSpeedTS(TCommandEnabler&);
    void CeToggleCheckEigVect(TCommandEnabler&);
    void CeToggleMatlabOut(TCommandEnabler&);
    void CeToggleRootLocusOut(TCommandEnabler&);
    void CeToggleTimeStepOut(TCommandEnabler&);
    void CePrintProfile(TCommandEnabler&);
    void CeRlplotScale(TCommandEnabler&);
    void CeTSplotScale(TCommandEnabler&);
  private:
    TPrinter* Printer;
    void SetupWindow();

  DECLARE_RESPONSE_TABLE(TStateWindow);
};


#endif
```

```
//
//Super Kite Trim State analysis
//

#include <stdio.h>
#include <stdlib.h>
#include <fstream.h>
#include <iomanip.h>
#include <owl/owlpch.h>
#include <owl/applicat.h>
#include <owl/scroller.h>
#include <owl/opensave.h>
#include <owl/printer.h>
#include <classlib/arrays.h>
#include <math.h>
#include <string.h>
#include "altitdef.h"
#include "suprkite.h"
#include "dialspec.h"
#include "trimstat.h"
#include "trimstat.rh"

//----------------------------------------------------------------------------
// TWindowPrintout

TWindowPrintout::TWindowPrintout(const char* title, TWindow* window)
  : TPrintout(title)
{
  Window = window;
  Scale = TRUE;
}

void
TWindowPrintout::PrintPage(int, TRect& rect, unsigned)
{
  // Conditionally scale the DC to the window so the printout will
  // resemble the window
  //
  int    prevMode;
  TSize  oldVExt, oldWExt;
  if (Scale) {
    prevMode = DC->SetMapMode(MM_ISOTROPIC);
    TRect windowSize = Window->GetClientRect();
    DC->SetViewportExt(PageSize, &oldVExt);
    DC->SetWindowExt(windowSize.Size(), &oldWExt);
    DC->IntersectClipRect(windowSize);
    DC->DPtoLP(rect, 2);
  }

  // Call the window to paint itself
  Window->Paint(*DC, FALSE, rect);

  // Restore changes made to the DC
  if (Scale) {
    DC->SetWindowExt(oldWExt);
    DC->SetViewportExt(oldVExt);
    DC->SetMapMode(prevMode);
  }
}

// Do not enable page range in the print dialog since only one page is
// available to be printed
//
void
TWindowPrintout::GetDialogInfo(int& minPage, int& maxPage,
{
  minPage = 0;
  maxPage = 0;
  selFromPage = selToPage = 0;
}

//---------------------------------------------------------------------------
//TStateWindow
DEFINE_RESPONSE_TABLE1(TStateWindow, TWindow)
  EV_WM_SIZE,
  EV_COMMAND(CM_FILENEW, CmFileNew),
  EV_COMMAND(CM_FILEOPEN, CmFileOpen),
  EV_COMMAND(CM_FILESAVE, CmFileSave),
  EV_COMMAND(CM_FILESAVEAS, CmFileSaveAs),
  EV_COMMAND(CM_KITE_GENERAL, CmKiteGeneral),
  EV_COMMAND(CM_KITE_WING, CmKiteWing),
  EV_COMMAND(CM_KITE_EMPEN, CmKiteEmpen),
  EV_COMMAND(CM_KITE_EMPEN2, CmKiteEmpen2),
  EV_COMMAND(CM_KITE_FUSELAGE, CmKiteFuselage),
  EV_COMMAND(CM_KITE_CONTROLGAIN, CmKiteControlGain),
  EV_COMMAND(CM_RLPLOT_SCALE, CmRlplotScale),
  EV_COMMAND(CM_TSPLOT_SCALE, CmTSplotScale),
  EV_COMMAND(CM_ALTITUDE_FILE, CmAltitudeFile),
  EV_COMMAND(CM_ALTITUDE_SAVE_FILE, CmAltitudeSaveFile),
  EV_COMMAND(CM_WIND_FUNCTION, CmWindFunction),
  EV_COMMAND(CM_CALCULATE_TRIMSTATE, CmCalcTState),
  EV_COMMAND(CM_CALCULATE_STABILITY, CmCalcStability),
  EV_COMMAND(CM_CALCULATE_TIMESTEP, CmCalcTimeStep),
  EV_COMMAND(CM_VARIATION_STUDY, CmVarStudy),
  EV_COMMAND(CM_TRIMSTATE_RESULTS, CmTStateResults),
  EV_COMMAND(CM_STABILITY_RESULTS, CmStabilityResults),
  EV_COMMAND(CM_STABILITY_DERIVS, CmStabilityDerivs),
  EV_COMMAND(CM_TOGGLEPROFVIEW, CmToggleProfView),
  EV_COMMAND(CM_TOGGLEROOTSVIEW, CmToggleRootsView),
  EV_COMMAND(CM_TOGGLETIMESTEPVIEW, CmToggleTimeStepView),
  EV_COMMAND(CM_HIGHSPEEDTS, CmToggleHighSpeedTS),
  EV_COMMAND(CM_TOGGLECHECKEIGVECT, CmToggleCheckEigVect),
  EV_COMMAND(CM_TOGGLEMATLABOUT, CmToggleMatlabOut),
  EV_COMMAND(CM_TOGGLEROOTLOCUSOUT, CmToggleRootLocusOut),
  EV_COMMAND(CM_TOGGLETIMESTEPOUT, CmToggleTimeStepOut),
  EV_COMMAND(CM_PRINTPROFILE, CmPrintProfile),
  EV_COMMAND(CM_FILEPRINTERSETUP, CmFilePrinterSetup),
  EV_COMMAND(CM_ABOUT, CmAbout),
  EV_COMMAND_ENABLE(CM_STABILITY_RESULTS, CeStabilityResults),
  EV_COMMAND_ENABLE(CM_STABILITY_DERIVS, CeStabilityDerivs),
  EV_COMMAND_ENABLE(CM_CALCULATE_STABILITY, CeCalcStability),
  EV_COMMAND_ENABLE(CM_CALCULATE_TIMESTEP, CeCalcTimeStep),
  EV_COMMAND_ENABLE(CM_VARIATION_STUDY, CeVarStudy),
  EV_COMMAND_ENABLE(CM_PRINTPROFILE, CePrintProfile),
  EV_COMMAND_ENABLE(CM_TOGGLEPROFVIEW, CeToggleProfView),
  EV_COMMAND_ENABLE(CM_TOGGLEROOTSVIEW, CeToggleRootsView),
  EV_COMMAND_ENABLE(CM_TOGGLETIMESTEPVIEW, CeToggleTimeStepView),
  EV_COMMAND_ENABLE(CM_HIGHSPEEDTS, CeToggleHighSpeedTS),
  EV_COMMAND_ENABLE(CM_TOGGLECHECKEIGVECT, CeToggleCheckEigVect),
  EV_COMMAND_ENABLE(CM_TOGGLEMATLABOUT, CeToggleMatlabOut),
  EV_COMMAND_ENABLE(CM_TOGGLEROOTLOCUSOUT, CeToggleRootLocusOut),
  EV_COMMAND_ENABLE(CM_TOGGLETIMESTEPOUT, CeToggleTimeStepOut),
  EV_COMMAND_ENABLE(CM_RLPLOT_SCALE, CeRlplotScale),
  EV_COMMAND_ENABLE(CM_TSPLOT_SCALE, CeTSplotScale),
END_RESPONSE_TABLE;

TStateWindow::TStateWindow(TWindow* parent)
  : TWindow(parent)
{
  Init(parent, 0, 0);
  Printer          = new TPrinter;
  KData            = new KiteData();
  AData            = new AltitudeData();
  Control          = new ControlGain();
  KQuery           = new KiteQuery();
  LgScale          = new Scale();
  LtScale          = new Scale();
  TSLgScale        = new Scale();
  TSLtScale        = new Scale();
  IsNewFile        = true;
  IsWindFile       = true;
  IsDirty          = false;
  IsTStateValid    = false;
  IsSRootValid     = false;
  IsRLocusValid    = false;
  IsTimeStepValid  = false;
  HighSpeedTS      = true;
  CheckEigVect     = false;
  MatlabOut        = false;
  RootLocusOut     = false;
  TimeStepOut      = false;
  ShowProf         = true;
  ShowRoots        = false;
  ShowTimeStep     = false;
  Proceed          = true;

  AltAcc           = 10.0;
  origMag          = 10.0;
  PixelWidth       = 0;
  PixelHeight      = 0;
  KiteFile     = new TOpenSaveDialog::TData(OFN_HIDEREADONLY|OFN_FILEMUSTEXIST,
                                   "Kite Data Files (*.kdf)|*.KDF|", 0, "",
                                   "KDF");
  AltitudeFile = new TOpenSaveDialog::TData(OFN_HIDEREADONLY|OFN_FILEMUSTEXIST,
                                   "Wind Data Files (*.wdf)|*.WDF|", 0, "",
                                   "WDF");

  //Initialize KiteQuery object with the current ControlGain object
  KQuery->SetControl(*Control, *KData);
}

TStateWindow::~TStateWindow()
{
   delete Printer;
   delete KData;
   delete AData;
   delete KiteFile;
   delete Control;
   delete KQuery;
   delete LgScale;
   delete LtScale;
   delete TSLgScale;
   delete TSLtScale;
   delete AltitudeFile;
}

void
TStateWindow::SetupWindow()
{
   TWindow::SetupWindow();
   KData->FileForm("wind10_0.wdf");
   strcpy(AltitudeFile->FileName, KData->FileForm());
   OpenFile(AltitudeFile);
}
```

```
// Adjust the scroller range so that the the origin is the
// upper-most scrollable point and the corner is the
// bottom-most.
void
TStateWindow::AdjustScroller()
{
  TRect  clientRect = GetClientRect();

  // only show scrollbars when image is larger than
  // the client area and we are not stretching to fit.
  //
  TPoint Range(Max(PixelWidth-clientRect.Width(), 0),
               Max(PixelHeight-clientRect.Height(), 0));
  Scroller->SetRange(Range.x, Range.y);

//  Scroller->ScrollTo(0, 0);
  if (!GetUpdateRect(clientRect, FALSE))
    Invalidate(FALSE);
}

// Reset scroller range.
//
void
TStateWindow::EvSize(UINT SizeType, TSize& Size)
{
  TWindow::EvSize(SizeType, Size);
  if (SizeType != SIZEICONIC) {
    AdjustScroller();
    Invalidate(FALSE);
  }
}

void
TStateWindow::CmPrintProfile()          // Execute File:Print command
{
  if (Printer) {
    TWindowPrintout printout("Print the Equilibrium Tether Profile", this);
    printout.SetBanding(TRUE);
    Printer->Print(this, printout, TRUE);
  }
}

void
TStateWindow::CmFilePrinterSetup()    // Execute File:Printer-setup command
{
  if (Printer)
    Printer->Setup(this);
}

bool
TStateWindow::CanClose()
{
  if (IsDirty)
    switch(MessageBox("Do you want to save?",
      "Super Kite configuration has changed",MB_YESNOCANCEL | MB_ICONQUESTION))
    {
      case IDCANCEL:
        // Choosing Cancel means to abort the close -- return false.
        return false;

      case IDYES:
        // Choosing Yes means to save the drawing.
        CmFileSave();
    }
  return true;
}
```

```
TStateWindow::CmKiteGeneral()
{
   if((GenDialog(this, KData)).Execute()==1)
   {
      IsDirty=true;
      IsTStateValid=IsSRootValid=IsRLocusValid=IsTimeStepValid=false;
   }
   Invalidate();
}

void
TStateWindow::CmKiteWing()
{
   if((WingDialog(this, KData)).Execute()==1)
   {
      IsDirty=true;
      IsTStateValid=IsSRootValid=IsRLocusValid=IsTimeStepValid=false;
   }
   Invalidate();
}

void
TStateWindow::CmKiteEmpen()
{
   if((EmpenDialog(this, KData)).Execute()==1)
   {
      IsDirty=true;
      IsTStateValid=IsSRootValid=IsRLocusValid=IsTimeStepValid=false;
   }
   Invalidate();
}

void
TStateWindow::CmKiteEmpen2()
{
   if((Empen2Dialog(this, KData)).Execute()==1)
   {
      IsDirty=true;
      IsTStateValid=IsSRootValid=IsRLocusValid=IsTimeStepValid=false;
   }
   Invalidate();
}

void
TStateWindow::CmKiteFuselage()
{
   if((FuseDialog(this, KData)).Execute()==1)
   {
      IsDirty=true;
      IsTStateValid=IsSRootValid=IsRLocusValid=IsTimeStepValid=false;
   }
   Invalidate();
}

void
TStateWindow::CmKiteControlGain()
{
   if((ControlGainDialog(this, Control)).Execute()==1)
   {
      KData->SetControl(*Control);
      IsSRootValid=IsTimeStepValid=false;
   }
   Invalidate();
}
```

```
TStateWindow::CmRlplotScale()
{
   (RLScaleDialog(this, LgScale, LtScale)).Execute();
   AdjustScroller();
   Invalidate();
}

void
TStateWindow::CmTSplotScale()
{
   (TSScaleDialog(this, TSLgScale, TSLtScale)).Execute();
   AdjustScroller();
   Invalidate();
}

void
TStateWindow::CmAltitudeFile()
{
   char buffer[150]="";
   if((TFileOpenDialog(this, *AltitudeFile)).Execute() == IDOK) {
      OpenFile(AltitudeFile);
      strcpy(buffer,AltitudeFile->FileName);
      KData->FileForm(buffer);
      KData->IWF=1;
      IsDirty=true;
      IsTStateValid=IsSRootValid=IsRLocusValid=IsTimeStepValid=false;
   }
   Invalidate();
}

void
TStateWindow::CmAltitudeSaveFile()
{
   char buffer[150]="";
   if ((TFileSaveDialog(this, *AltitudeFile)).Execute() == IDOK)
   {
      SaveFile(AltitudeFile);
      strcpy(buffer,AltitudeFile->FileName);
      KData->FileForm(buffer);
      KData->IWF=1;
      IsDirty=true;
   }
}

void
TStateWindow::CmWindFunction()
{
   if((WindFormDialog(this, KData)).Execute()==1)
   {
      if(KData->IWF==0)
      {
         KData->SpecWind(*AData);
         IsDirty=true;
         IsTStateValid=IsSRootValid=IsRLocusValid=IsTimeStepValid=false;
      }
   }
   Invalidate();
}

void
TStateWindow::CmCalcTState()
{
   int flag=1,highct=0,lowct=0,Iternum=0;
   double scale,A1,A2,rate,maxalt,minalt,percent,origAlt;
   IsTStateValid=false;
   scale=1.0;
```

```
percent=1.0;
origAlt=KData->teleml;
minalt=AData->MinAlt;
maxalt=AData->MaxAlt;

// set the cursor to an hourglass - this might take awhile
//
HCURSOR oldCur = ::SetCursor(::LoadCursor(0, IDC_WAIT));

//Calculate the various characteristics of SuperKite that are indep. of RN's
KData->KiteCValues();

//Get the currently saved Super Kite altitude
A1=KData->KiteAlt();

//Check to see if the altitude is in the range of the datafile
if(A1<minalt||A1>maxalt||KData->AOA==-99.0) A1=(maxalt+minalt)/2.0;

//Create a temporary Altitude object
Altitude *Alt;

//Start the iteration loop for Calculating the Trimstate
while(flag!=0&&Iternum<1000)
{
   Iternum++;

   //Instantiate the temporary Altitude object/initialize to current alt
   Alt = new Altitude(A1);

   //Get the data assoc. with the altitude from the datafile
   // and store them in Alt
   AData->GetData(*Alt, *AData);

   //Calculate the RN's and q for the given altitude
   KData->KiteqRN(*Alt);

   //Set the element length ten times larger on the initial
   //altitude.  Then set it back to the actual altitude.
   if(flag==1)
      KData->teleml=origAlt*origMag;
   else if(flag==2)
      KData->teleml=origAlt;

   //Calculate the "Actual" altitude given the RN's and q
   A2=KData->KiteTrimAngle();

   //Compare the "Actual" alt. to prev. guess and change
   // new guess appropriately
   if(fabs(A1-A2)<max(percent*KData->teleml,AltAcc))
   {
      if(flag==3)
      {
         if(KData->AOA!=-99.0)
            IsTStateValid=true;
         flag=0;
      }
      else if(flag==2)
      {
         flag=3;
         percent=0.1;
      }
      else if(flag==1)
      {
         flag=2;
      }
   }
```

```
      //available set the new altitude to the lower limit
      else if(A2<minalt){
         ++lowct;
         if(lowct>100){
            IsTStateValid=false;
            flag=0;
         }else{
            A2=minalt;
            if(fabs((A2-A1)/(scale*5.0))<5.0)
               A1+=(((A2-A1)/fabs(A2-A1))*5.0);
            else
               A1+=((A2-A1)/(scale*5.0));
         }
      }
      //If the resulting altitude is above the range for which data is
      //available set the new altitude to the upper limit
      else if(A2>maxalt){
         ++highct;
         if(highct>100){
            IsTStateValid=false;
            flag=0;
         }else{
            A2=maxalt;
            if(fabs((A2-A1)/(scale*5.0))<5.0)
               A1+=(((A2-A1)/fabs(A2-A1))*5.0);
            else
               A1+=((A2-A1)/(scale*5.0));
         }
      }
      //If the new altitude is not within the desired accuracy increment
      //the altitude another step
      else{
         if(fabs((A2-A1)/(scale*5.0))<5.0)
            A1+=(((A2-A1)/fabs(A2-A1))*5.0);
         else
            A1+=((A2-A1)/(scale*5.0));
      }
      scale+=rate;
      delete Alt;
   }

   if(KData->AOA!=-99.0)
      KData->KiteTetherProfile();
   IsDirty = true;

   //Set the cursor back to arrow
   ::SetCursor(oldCur);

   if(!ShowProf) CmToggleProfView();

   Invalidate();

   (TrimResultsDialog(this, KData, IsTStateValid)).Execute();
   AdjustScroller();
   Invalidate();
}

void
TStateWindow::CmCalcStability()
{
   if(IsTStateValid)
   {
      Proceed=true;
      if(KData->N>15)
      {
         if(MessageBox("Number of tether elements exceeds 15.\n"
            MB_YESNO | MB_ICONQUESTION)==IDNO)
               Proceed=false;
      }

      if(Proceed)
      {
         // set the cursor to an hourglass - this might take awhile
         //
         HCURSOR oldCur = ::SetCursor(::LoadCursor(0, IDC_WAIT));

         int err;
         IsSRootValid=false;

         err=KData->KiteSRoots(*Control, *AData, CheckEigVect, MatlabOut);

         //Set the cursor back to arrow
         ::SetCursor(oldCur);
         Invalidate();

         if(err==0)
         {
            (StabilityResultsDialog(this, KData)).Execute();
            IsSRootValid=true;

         }else if(err==-5)
            MessageBox("Could not allocate arrays.  Out of Memory!"
            , "Super Kite Stability Roots  --  ERROR", MB_OK | MB_ICONQUESTION);
         else
            MessageBox("Error calculating the Stability Roots!"
            , "Super Kite Stability Roots  --  ERROR", MB_OK | MB_ICONQUESTION);

         IsDirty = true;
      }
   }
   else
   {
      MessageBox("You must calculate the equilibrium trimstate first."
      , "Super Kite configuration has changed", MB_OK | MB_ICONQUESTION);
   }
   AdjustScroller();
   Invalidate();
}

void
TStateWindow::CmCalcTimeStep()
{
   if(IsSRootValid&&(TimeStepDialog(this, KData)).Execute() == IDOK)
   {
      KData->TimeStepAnalysis(*Control, *AData, TimeStepOut);
      IsTimeStepValid=true;
      if(!ShowTimeStep) CmToggleTimeStepView();
   }
   AdjustScroller();
   Invalidate();
}

void
TStateWindow::CmVarStudy()
{

   if((QueryVarStudyDialog(this, KData, KQuery)).Execute() == IDOK)
   {
      Proceed=true;
      if(KData->N>15)
      {
         if(MessageBox("Number of tether elements exceeds 15.\n"
        MB_YESNO | MB_ICONQUESTION)==IDNO)
          Proceed=false;
    }

    if(Proceed)
    {
      int err,i,j,f,itr;
      ControlGain VCG;
      VCG=*Control;
      IsSRootValid=false;
      f=4;
      if(KData->N<=1) f=0;
      // set the cursor to an hourglass - this might take awhile
      //
      HCURSOR oldCur = ::SetCursor(::LoadCursor(0, IDC_WAIT));

      //Get variation parameters
      VarParam VP;
      KQuery->GetVarParam(VP,VCG,*KData);

      //Loop through the NSteps specified in the KQuery object
      for(i=1;i<=VP.NSteps+1;i++)
      {
        //Calculate the various characteristics of SuperKite that are indep. of RN's
        KData->KiteCValues();
        err=KData->KiteSRoots(VCG, *AData, false, false, 0);
        if(err==0)
        {
          itr=1;
          for(j=0;j<KQuery->LgRoots.m+f;j++)
          {
            if(KData->Lgalfi[j]>=0.0)
            {
              KQuery->LgRoots(itr,i) =
                complex(KData->Lgalfr[j],KData->Lgalfi[j]);
              itr++;
            }
          }
          itr=1;
          for(j=0;j<KQuery->LtRoots.m+f/2;j++)
          {
            if(KData->Ltalfi[j]>=0.0)
            {
              KQuery->LtRoots(itr,i) =
                complex(KData->Ltalfr[j],KData->Ltalfi[j]);
              itr++;
            }
          }
        }

        //Increment the ControlGain object
        switch(VP.LgActive)
        {
          case 1:
            VCG.Ccx+=VP.LgInc;
            break;
          case 2:
            VCG.Ccz+=VP.LgInc;
            break;
          case 3:
            VCG.Ccm+=VP.LgInc;
            break;
        }
        switch(VP.LtActive)
        {
          case 4:
```

```
                    break;
                 case 5:
                    VCG.Ccl+=VP.LtInc;
                    break;
                 case 6:
                    VCG.Ccn+=VP.LtInc;
                    break;
                 case 7:
                    KData->bvt2+=VP.LtInc;
                    break;
                 case 8:
                    KData->Xvt2+=VP.LtInc;
                    break;
              }
           }
           IsRLocusValid = true;

           KQuery->ResetKiteData(*KData);

           //Set the cursor back to arrow
           ::SetCursor(oldCur);

           if(!ShowRoots) CmToggleRootsView();
        }
     }
     if(RootLocusOut && IsRLocusValid)  RLDataOut();
     AdjustScroller();
     Invalidate();
  }

  void
  TStateWindow::RLDataOut()
  {
     int i,j,k;

     //Initialize the output file for the Root-Locus data
     fstream RLout;

     //Save the data from the long. and lat. Root-Locus plots
     //
     RLout.open("LgRLocus.txt",ios::out|ios::app);
     RLout<< setprecision(24);
     RLout<<"Data for Longitudinal Root-Locus Plot\n";
     RLout<<KQuery->LgParam << " from " << KQuery->Start(KQuery->LgActive);
     RLout<< " to " << KQuery->End(KQuery->LgActive) <<"\n\n";
     for(i=1;i<=KQuery->LgRoots.n;i++)
     {
        for(j=1;j<=KQuery->LgRoots.m;j++)
        {
           RLout << real(KQuery->LgRoots(j,i));
           for(k=1;k<=j;k++)
              RLout << "\t";
           RLout << imag(KQuery->LgRoots(j,i)) << "\n";
        }
     }
     RLout << "\n\n";
     RLout.close();

     RLout.open("LtRLocus.txt",ios::out|ios::app);
     RLout<< setprecision(24);
     RLout<<"Data for Lateral Root-Locus Plot\n";
     RLout<<KQuery->LtParam << " from " << KQuery->Start(KQuery->LtActive);
     RLout<< " to " << KQuery->End(KQuery->LtActive) <<"\n\n";
     for(i=1;i<=KQuery->LtRoots.n;i++)
     {
        for(j=1;j<=KQuery->LtRoots.m;j++)
```

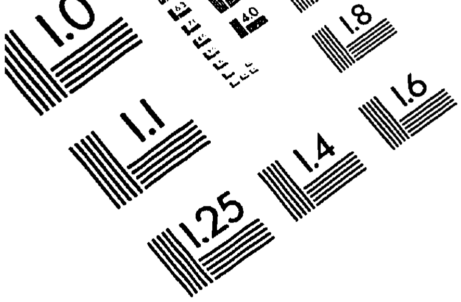
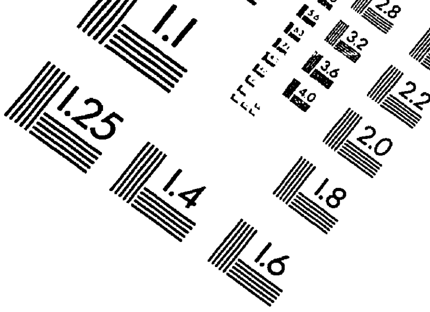
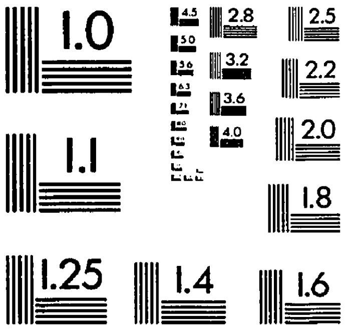
```
         RLout << real(KQuery->LtRoots(j,i));
         for(k=1;k<=j;k++)
            RLout << "\t";
         RLout << imag(KQuery->LtRoots(j,i)) << "\n";
      }
   }
   RLout << "\n\n";
   RLout.close();
}

void
TStateWindow::CmTStateResults()
{
   (TrimResultsDialog(this, KData, IsTStateValid)).Execute();
   Invalidate();
}

void
TStateWindow::CmStabilityResults()
{
   (StabilityResultsDialog(this, KData)).Execute();
   Invalidate();
}

void
TStateWindow::CmStabilityDerivs()
{
   KData->CalculateNonDims(*AData);
   (StabilityDerivsDialog(this, KData)).Execute();
   Invalidate();
}

void
TStateWindow::CmToggleProfView()
{
   ShowProf=true;
   ShowRoots=false;
   ShowTimeStep=false;
   AdjustScroller();
   Invalidate();
}

void
TStateWindow::CmToggleRootsView()
{
   ShowProf=false;
   ShowRoots=true;
   ShowTimeStep=false;
   AdjustScroller();
   Invalidate();
}

void
TStateWindow::CmToggleTimeStepView()
{
   ShowProf=false;
   ShowRoots=false;
   ShowTimeStep=true;
   PixelWidth = 5600.0/7.5;
   PixelHeight = 8300.0/7.7;
   AdjustScroller();
   Invalidate();
}

void
TStateWindow::CmToggleHighSpeedTS()
```

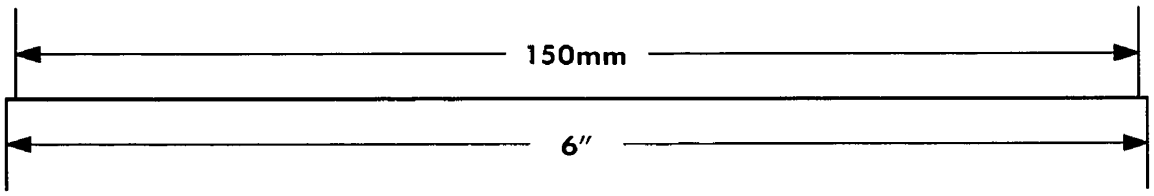
```
   if(HighSpeedTS)
      origMag=10.0;
   else
      origMag=1.0;

   Invalidate();
}

void
TStateWindow::CmToggleCheckEigVect()
{
   CheckEigVect=!CheckEigVect;
   Invalidate();
}

void
TStateWindow::CmToggleMatlabOut()
{
   MatlabOut=!MatlabOut;
   Invalidate();
}

void
TStateWindow::CmToggleRootLocusOut()
{
   RootLocusOut=!RootLocusOut;
   Invalidate();
}

void
TStateWindow::CmToggleTimeStepOut()
{
   TimeStepOut=!TimeStepOut;
   Invalidate();
}

void
TStateWindow::CmFileNew()
{
   char temp[100] = "Super Kite Analysis Program";

   if (CanClose()) {
      //Clear filename in main window caption
      Parent->SetCaption(temp);

      delete KData;
      KData = new KiteData();
      KData->FileForm("wind10_0.wdf");
      strcpy(AltitudeFile->FileName, KData->FileForm());
      OpenFile(AltitudeFile);

      //Initialize KiteQuery object with the current ControlGain object
      KQuery->SetControl(*Control, *KData);

      Control->Zero();

      IsDirty = false;
      IsNewFile = true;
      IsTStateValid=IsSRootValid=IsRLocusValid=IsTimeStepValid=false;
   }
   Invalidate();
}

void
TStateWindow::CmFileOpen()
```

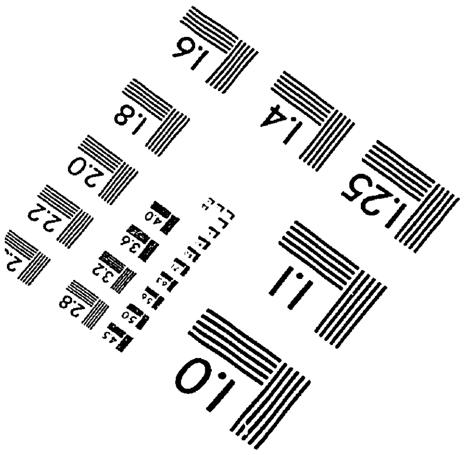
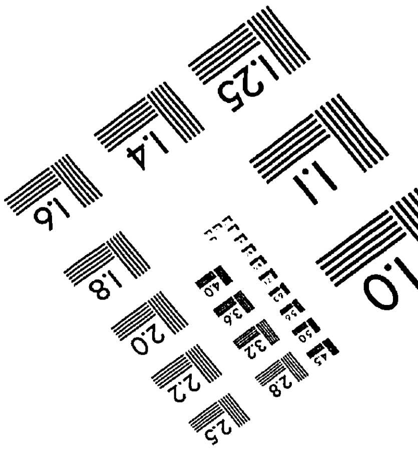
```
   if (CanClose())
   {
      if ((TFileOpenDialog(this, *KiteFile)).Execute() == IDOK)
      {
         OpenFile(KiteFile);
         KData->GetControl(*Control);
         //Initialize KiteQuery object with the current ControlGain object
         KQuery->SetControl(*Control, *KData);
         if(KData->IWF)
         {
            strcpy(AltitudeFile->FileName, KData->FileForm());
            OpenFile(AltitudeFile);
         }else{
            strcpy(AltitudeFile->FileName, "wind10_0.wdf");
            OpenFile(AltitudeFile);
            KData->SpecWind(*AData);
         }
      }

   }
   Invalidate();
}

void
TStateWindow::CmFileSave()
{
   if (IsNewFile)
      CmFileSaveAs();
   else
      SaveFile(KiteFile);
   Invalidate();
}

void
TStateWindow::CmFileSaveAs()
{
   if (IsNewFile)
      strcpy(KiteFile->FileName, "");

   if ((TFileSaveDialog(this, *KiteFile)).Execute() == IDOK)
      SaveFile(KiteFile);
   Invalidate();
}

void
TStateWindow::CmAbout()
{
   TDialog(this, IDD_ABOUT).Execute();
   Invalidate();
}

void
TStateWindow::SaveFile(TOpenSaveDialog::TData* file)
{
  ofstream os(file->FileName);
  char temp[100] = "Super Kite Analysis Program - ";

  if (!os)
    MessageBox("Unable to open file", "File Error", MB_OK|MB_ICONEXCLAMATION);
  else {
    if(file==AltitudeFile){
      os << *AData;
    }else{

      os << *KData;
      strcat(temp,file->FileName);
      Parent->SetCaption(temp);

      // Set new file and dirty display indicator to false.
      IsNewFile = IsDirty = false;
    }
  }
}

void
TStateWindow::OpenFile(TOpenSaveDialog::TData* file)
{
  ifstream is(file->FileName);
  char temp[100] = "Super Kite Analysis Program - ";

  if (!is){
    MessageBox("Unable to open file", "File Error", MB_OK|MB_ICONEXCLAMATION);
  }else {
    if(file==AltitudeFile){
      AData->Flush();
      is >> *AData;
    }else{
      is >> *KData;

      //Set filename in main window caption
      strcat(temp,file->FileName);
      Parent->SetCaption(temp);

      IsNewFile=IsDirty=IsTStateValid=IsSRootValid=IsRLocusValid=false;
      IsTimeStepValid=false;
    }
  }
}

void
TStateWindow::CePrintProfile(TCommandEnabler& ce)
{
  // Enable CmPrintProfile if ShowProf is true.
  ce.Enable(ShowProf||ShowRoots||ShowTimeStep);
}

void
TStateWindow::CeCalcStability(TCommandEnabler& ce)
{
  // Enable CmCalcStability if Trim state is already valid
  ce.Enable(IsTStateValid);
}

void
TStateWindow::CeCalcTimeStep(TCommandEnabler& ce)
{
  // Enable CeCalcTimeStep if Stability Derivatives are already valid
  ce.Enable(IsSRootValid);
}

void
TStateWindow::CeVarStudy(TCommandEnabler& ce)
{
  // Enable CmVarStudy if Trim state is already valid
  ce.Enable(IsTStateValid);
}

void
TStateWindow::CeStabilityResults(TCommandEnabler& ce)
{
  // Enable CmStabilityResults if the stability roots are already valid
```

```
}

void
TStateWindow::CeStabilityDerivs(TCommandEnabler& ce)
{
  // Enable CmStabilityDerivs if Trim state is already valid
  ce.Enable(IsTStateValid);
}

void
TStateWindow::CeToggleProfView(TCommandEnabler& ce)
{
  ce.SetCheck(ShowProf ? TCommandEnabler::Checked :
               TCommandEnabler::Unchecked);
}

void
TStateWindow::CeToggleRootsView(TCommandEnabler& ce)
{
  ce.SetCheck(ShowRoots ? TCommandEnabler::Checked :
               TCommandEnabler::Unchecked);
}

void
TStateWindow::CeToggleTimeStepView(TCommandEnabler& ce)
{
  ce.SetCheck(ShowTimeStep ? TCommandEnabler::Checked :
               TCommandEnabler::Unchecked);
}

void
TStateWindow::CeToggleHighSpeedTS(TCommandEnabler& ce)
{
   ce.SetCheck(HighSpeedTS ? TCommandEnabler::Checked :
               TCommandEnabler::Unchecked);
}

void
TStateWindow::CeToggleCheckEigVect(TCommandEnabler& ce)
{
  ce.SetCheck(CheckEigVect ? TCommandEnabler::Checked :
               TCommandEnabler::Unchecked);
}

void
TStateWindow::CeToggleMatlabOut(TCommandEnabler& ce)
{
  ce.SetCheck(MatlabOut ? TCommandEnabler::Checked :
               TCommandEnabler::Unchecked);
}

void
TStateWindow::CeToggleRootLocusOut(TCommandEnabler& ce)
{
  ce.SetCheck(RootLocusOut ? TCommandEnabler::Checked :
               TCommandEnabler::Unchecked);
}

void
TStateWindow::CeToggleTimeStepOut(TCommandEnabler& ce)
{
  ce.SetCheck(TimeStepOut ? TCommandEnabler::Checked :
               TCommandEnabler::Unchecked);
}

void
```

```
{
  // Enable CmRlplotScale if Root-Locus plot is displayed
  ce.Enable(ShowRoots);
}

void
TStateWindow::CeTSplotScale(TCommandEnabler& ce)
{
  // Enable CmTSplotScale if Time-Step plot is displayed
  ce.Enable(ShowTimeStep);
}

class TStateApp : public TApplication {
  public:
    TStateApp() : TApplication() {}

    void InitMainWindow();

  protected:
    TGadgetWindow::THintMode HintMode;

};


void
TStateApp::InitMainWindow()
{
   // Construct the decorated frame window
   TDecoratedFrame* frame = new TDecoratedFrame(0,"Super Kite Analysis Program"
                            , new TStateWindow, true);

   // Construct a status bar
   TStatusBar* sb = new TStatusBar(frame, TGadget::Recessed);

   // Construct a control bar
   HintMode = TGadgetWindow::EnterHints;
   TControlBar* cb = new TControlBar(frame);
   cb->Insert(*new TButtonGadget(CM_FILENEW, CM_FILENEW,
         TButtonGadget::Command));
   cb->Insert(*new TButtonGadget(CM_FILEOPEN, CM_FILEOPEN,
         TButtonGadget::Command));
   cb->Insert(*new TButtonGadget(CM_FILESAVE, CM_FILESAVE,
         TButtonGadget::Command));
   cb->Insert(*new TButtonGadget(CM_FILESAVEAS, CM_FILESAVEAS,
         TButtonGadget::Command));
   cb->Insert(*new TSeparatorGadget);
   cb->Insert(*new TButtonGadget(CM_KITE_GENERAL, CM_KITE_GENERAL,
         TButtonGadget::Command));
   cb->Insert(*new TButtonGadget(CM_KITE_WING, CM_KITE_WING,
         TButtonGadget::Command));
   cb->Insert(*new TButtonGadget(CM_KITE_EMPEN, CM_KITE_EMPEN,
         TButtonGadget::Command));
   cb->Insert(*new TButtonGadget(CM_KITE_EMPEN2, CM_KITE_EMPEN2,
         TButtonGadget::Command));
   cb->Insert(*new TButtonGadget(CM_KITE_FUSELAGE, CM_KITE_FUSELAGE,
         TButtonGadget::Command));
   cb->Insert(*new TSeparatorGadget);
   cb->Insert(*new TButtonGadget(CM_CALCULATE_TRIMSTATE,
         CM_CALCULATE_TRIMSTATE, TButtonGadget::Command));
   cb->Insert(*new TButtonGadget(CM_CALCULATE_STABILITY,
         CM_CALCULATE_STABILITY, TButtonGadget::Command));
   cb->Insert(*new TButtonGadget(CM_VARIATION_STUDY,
         CM_VARIATION_STUDY, TButtonGadget::Command));
   cb->Insert(*new TButtonGadget(CM_CALCULATE_TIMESTEP,
         CM_CALCULATE_TIMESTEP, TButtonGadget::Command));
   cb->Insert(*new TSeparatorGadget);
   cb->SetDirection(TControlBar::Vertical);

   // Insert the status bar and control bar into the frame
   frame->Insert(*sb, TDecoratedFrame::Bottom);
   frame->Insert(*cb, TDecoratedFrame::Left);
   cb->SetHintMode(HintMode);

   // Set the main window and its menu
   SetMainWindow(frame);
   GetMainWindow()->AssignMenu("COMMANDS");
   GetMainWindow()->SetIcon(this,"THARWP");

   GetMainWindow()->Attr.X = GetSystemMetrics(SM_CXSCREEN) / 8;
   GetMainWindow()->Attr.Y = GetSystemMetrics(SM_CYSCREEN) / 8;
   GetMainWindow()->Attr.H = GetMainWindow()->Attr.Y * 6;
   GetMainWindow()->Attr.W = GetMainWindow()->Attr.X * 6;

   EnableBWCC(true);
}


int
OwlMain(int /*argc*/, char* /*argv*/ [])
{
  return TStateApp().Run();
}
```



APPLIED IMAGE . Inc
1653 East Main Street
Rochester, NY 14609 USA
Phone: 716/482-0300
Fax: 716/288-5989