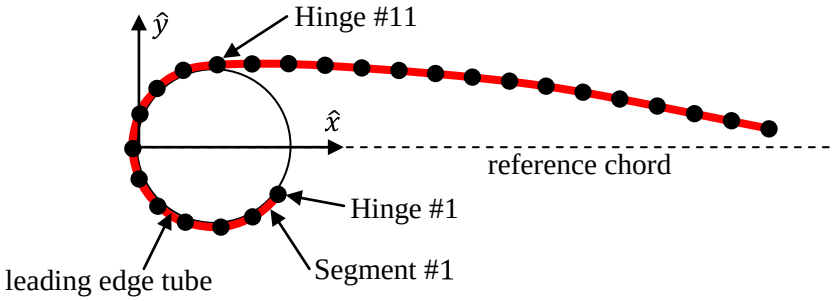
Preliminary design of a high-altitude kite

A flexible membrane kite section at high and low altitudes

In an earlier paper, titled *The shape of a flexible membrane airfoil in a uniform two-dimensional flow*, I examined a flexible membrane whose leading and trailing edges were mathematical points. The leading and trailing edges were held a fixed distance apart and the membrane was allowed to take on whatever shape was consistent with the steady airflow. In this paper, I will extend the analysis to include more realistic leading and trailing edges.

I will assume that the leading edge consists of a round tube such as shown in the following figure. I anticipate using a thin-walled aluminum tube. I will place the origin of the $\hat{x}$-$\hat{y}$ co-ordinate frame of reference (in which the membrane's profile is calculated) on the outside circumference of the leading edge tube, diametrically opposed to the ray leading to the trailing edge. The front edge of the membrane will not be attached to the tube at this origin. Instead, the front edge of the membrane will be wrapped under the tube and attached somewhere on the lower rear surface. As before, the membrane will be divided into discrete segments for analysis purposes. Also as before, I will describe the point of contact between neighbouring segments as "hinges". The entire length of the membrane will be divided into segments, including that part which is wrapped around the leading edge tube. In the figure, Hinge #1 marks the front edge of the membrane. The membrane is wrapped under the tube and then up and over its top. Tension forces in the membrane will keep it firmly in place. This configuration ensures that the nose of the kite section, or profile, will be nicely rounded and not marred by whatever mechanism is used to attach the membrane physically to the tube.

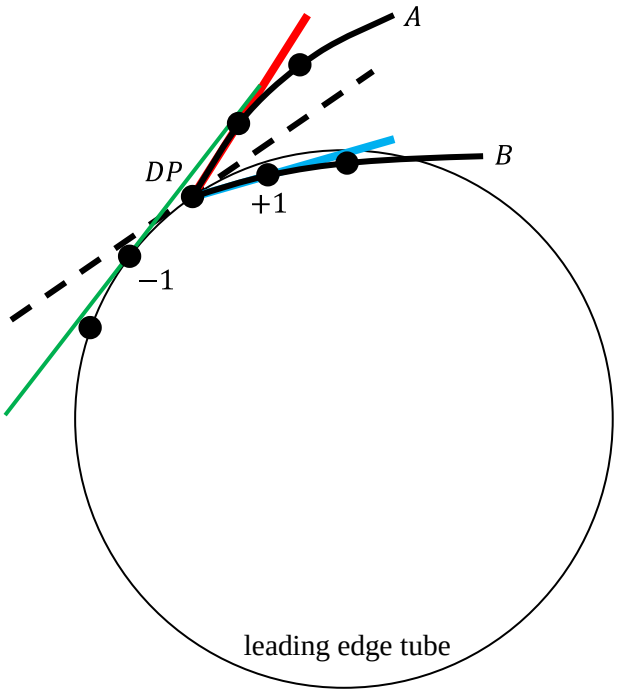


One consequence of this arrangement is that the first few segments of the membrane are tightly bound to the surface of the tube. Their position will not be determined by the airflow. In the figure, all of the hinges from #1 to #11 are held against the tube's surface. Segment #11, which follows Hinge #11, is the first one which is able to move in response to the airflow.

I will divide the flexible membrane into segments with equal surface length, just like I did in the earlier paper. Because the number of segments which actually lie on the leading edge tube will change as the shape of the membrane changes, it is convenient to include the segments which lie on the tube in the numbering scheme. The leading edge of the membrane is, as before, labeled Hinge #1 and the first segment in the chain is labeled Segment #1. It should be understood that there will be a point somewhere on the outside circumference of the leading edge tube where the flexible membrane will depart from the surface. I will call this point the “departure point”. For convenience, I will assume that the departure point always coincides with one of the hinges on the membrane. In the figure above, the departure point is Hinge #11. Segments of the membrane which are upwind of the departure point lie on the surface of the tube; segments which are downwind of the departure point are in the airstream and have air on both the upper and lower surfaces. But it is only Segments #11 and those following which are free-flying and whose shape is determined by the airflow.

From the standpoint of discretizing the membrane, wrapping part of it around the leading edge tube only makes sense if the length of the segments is relatively short compared with the diameter of the tube. In this paper, I will divide the membrane into 500 segments, which includes those segments lying on the surface of the leading edge tube. If the membrane is one meter long, for example, then each segment will be two millimeters long. If the leading edge tube has a diameter of one inch, for example, then its circumference will be equal to 79.8 millimeters, or about 40 segment lengths. That should be a satisfactory representation of a circle.

The identity of the hinge at the departure point will not be known until the shape of the membrane is finalized. It may take a couple of attempts to get things right. When we start out on a march along the membrane, the hinge which is the departure point is one more quantity we will need to guess. In the earlier paper, we needed to make two guesses before starting a march: (i) the tension force which the first segment exerts on its support and (ii) the angle between that tension force and the reference chord. We will now also need to guess which hinge to use as the starting point for the march. The following figure shows the mathematical tests I devised to ensure that the correct hinge is chosen as the departure point.

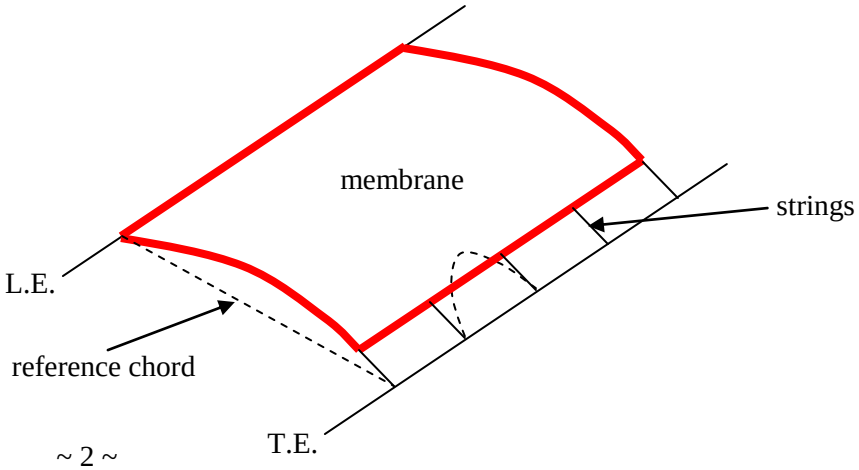


The figure shows two membranes, labeled A and B. Both membranes are being tested for departure at the hinge labeled DP. The thick red line is the slope between the proposed departure point and the next hinge in the chain of membrane A. The thick blue line is the slope between the proposed departure point and the next hinge in the chain of membrane B. The dashed black line is the slope of the leading edge tube (or circle, when seen in cross-section) at the proposed departure point.

The first test is to ensure that the slope of the membrane at the departure part is greater than the slope of the leading edge tube. Membrane B fails this test. To avoid having membrane B pass through the leading edge tube, we need to test hinge $+1$, which is further aft along the membrane, as the departure point. Membrane A passes the first test but fails the second test.

The second test is to ensure that the slope of the membrane at the proposed departure point is less than the slope of the leading edge tube at the previous hinge. In the figure, the thin green line is the slope of the leading edge tube at the previous hinge, which is labeled -1. The discontinuity where membrane A leaves the surface of the tube would be less if we used hinge -1 as the departure point.

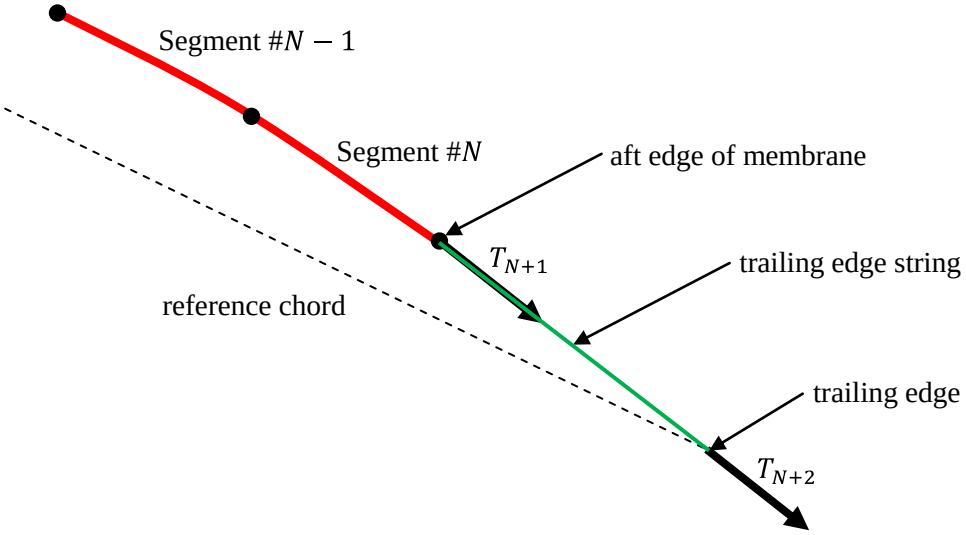
The figure to the right shows my plan for the trailing edge of the kite.



The flexible membrane itself does not extend all the way to the aft end of the reference chord, which is officially the "trailing edge" of the section. The location of the trailing edge is going to be determined by the rigid framework of the kite. Perhaps it will be the line which connects the aft ends of the ribs. The aft edge of the membrane is tied to the trailing edge by strings placed at regular intervals along the span. Physically, the aft edge of the membrane will likely be sewn to a transverse string, which in this configuration would be called a "boltrope". The tension in the aft edge of the membrane will stretch the boltrope into arch-like curves like the one shown by the dotted line. One cannot model such arches in the two-dimensional OpenFoam analysis used in this paper, because the chord-wise surface length of the membrane varies along the span. For the time being, I will simply estimate the average depth of the arches, and assume the aft edge of the membrane extends as far as that average depth. I will refer to the strings which pull back on the aft edge of the membrane as "trailing edge strings".

The trailing edge strings are going to be treated as ideal strings. They will be pulled forward by the tension force in the membrane and held back by the structure at the trailing edge. The tension force along their length will keep them straight. As the shape of the membrane changes, the angle which the trailing edge strings make with the reference chord will also change.

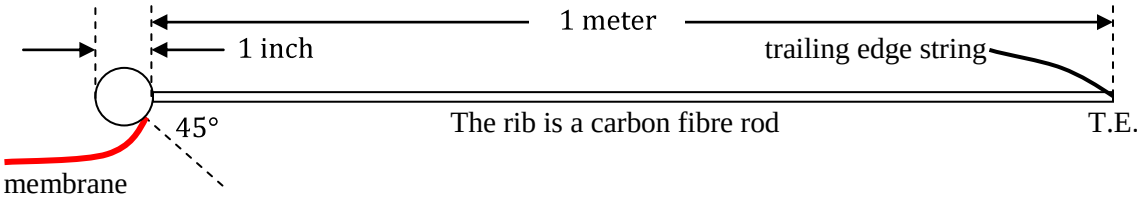
It is possible to arrange things so that the trailing edge strings can be treated as a simple extension of the flexible membrane. The following figure shows the aft end of the section in a little more detail. The cross-section shown happens to be at a span-wise location which includes a trailing edge string. I have not shown the physical structure which restrains the aft end of the trailing edge string, but have simply assumed that the trailing edge string terminates at that point. As in the earlier paper, the flexible membrane itself is divided into N discrete line segments, of which I have shown only the last two, Segment #$N-1$ and Segment #N. The hinge points between adjacent segments are shown as small black dots. The figure is intended to show how the trailing edge string transmits the tension force from the aft hinge of Segment #N to the trailing edge. In the earlier paper, I used the symbol T_j for the tension force (per unit length in the span-wise direction) which Segment #j exerts on the segment to its left. In the earlier paper, T_{N+1} was therefore the tension which the trailing edge exerted on Segment #N (and vice versa). If we treat the entire trailing edge string as an additional segment, then T_{N+1} is the tension which the trailing edge string exerts on the right-hand hinge of Segment #N and T_{N+2} is the tension which the trailing edge structure exerts on the right-hand edge of the trailing edge string.



The trailing edge string transmits the tension force from the right-hand side of Segment #N directly through to the trailing edge structure. Like the membrane itself, the trailing edge string is assumed to be

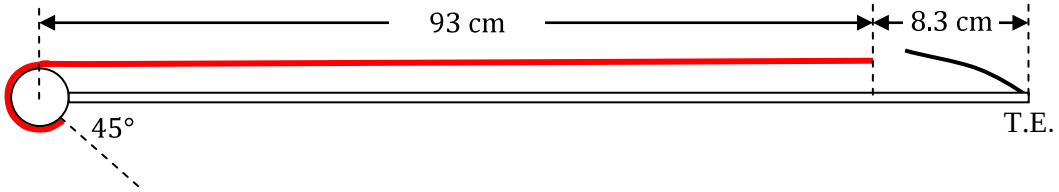
perfectly flexible, so it can only transmit tension forces. It cannot be compressed, nor can it withstand applied moments. In fact, introducing a trailing edge string is very similar to adding an additional, longer, segment to the end of the chain of segments which make up the membrane. The only difference is that there are no aerodynamic forces acting on this additional segment. (The trailing edge string will of course experience some aerodynamic drag, but I will assume the drag is negligible compared with the aerodynamic forces acting on the segments of the membrane, and negligible compared with the tension forces which counteract the aerodynamic forces.) Since the trailing edge string is not subject to aerodynamic forces, its internal tension will be constant along its length. It follows that the magnitude of tension T_{N+2} must be the same as the magnitude of tension T_{N+1}, and that both tension forces are inclined at exactly the same angle to the reference chord.

Let me start a numerical example. This example will constitute the "base case" for the OpenFoam simulations described later in this paper. The leading edge tube is an aluminum tube one inch in diameter. Thin carbon fibre rods one meter long are securely fixed onto the exterior surface of the tube. The free ends of these rods define the trailing edge of the section.



The membrane is a thin red nylon sheet. (I like red.) Its front edge is attached to the outside of the aluminum tube along a line which is 135° around the tube from its nominal leading edge (the "nose" of the section). The trailing edge string is a piece of Kevlar line attached to the free end of the carbon tube.

I will assume that the closed part of the membrane, which is to say, the nylon sheet, has a chord-wise length of 98 centimeters, after allowing for the narrow hem which will likely be needed for securement purposes at both edges. The following figure shows what happens if we wrap the nylon around the leading edge tube and pull it tight.



The nylon wraps around five-eighths of the circumference of the aluminum tube, which takes up about 5 cm of the nylon. The remaining 93 cm of nylon runs straight aft from the top of the leading edge tube. This leaves a horizontal distance of about 8.3 cm to the trailing edge. I will control the camber, or curvature, of the section by carefully setting the length of the trailing edge strings. Something near 10 cm will probably suit and is used in the base case. There is no "best" length. Factors which affect the selection of the length of the trailing edge strings include: the wind conditions, the desired angle of attack, the relative lengths of lines in the bridle, the desired margin of safety against separation of the airflow, the desired lift and lift-to-drag ratio, and so forth.

Let me return to the matter of discretization. For convenience, I will divide the entire nylon surface into 500 segments. The length of the resulting segments will be $98\text{ cm}/500 = 0.196\text{ cm}$.

The VisualBasic program used in the earlier paper to calculate the shape of the surface from a given distribution of aerodynamic forces was modified to handle the new configuration. Modifications were

made to five of the eight modules in the program. The entire program is listed below in Appendix "C", so I will mention here only the nature of the changes made.

1. In the earlier paper, the shape of the membrane used in Iteration #1 of the procedure was a single circular arc running from the leading edge of the profile to the trailing edge. The radius of the arc was selected to give the desired ratio of the length of flexible membrane to the length of the reference chord. For the current application, the starting shape (which is referred to in the program as the "seed" shape), consists of three curves: (i) a circular arc partway around the leading edge circle, (ii) a circular arc representing the free-flying part of the membrane and (iii) a straight line segment representing the trailing edge strings. The formulae used to define these three curves are described in Appendix "A" attached. These formulae are coded in the module named *SeedACrcularArc.vb*.

2. The shape calculations are carried out in the module named *OneMarchAlongMembrane.vb*. Changes were required to the original version of this module to deal with the location of the departure point and to deal with the trailing edge strings. The latter change was quite simple. The trailing edge strings are treated as the 501^{st} segment in the chain. Since the aerodynamic forces on them are being ignored, no shape-calculations need to be carried out. The tension at the right-hand hinge of the 500^{th} segment propagates straight through to the trailing edge. Conceptually, the existence of the trailing edge strings is nothing more than a translation of the trailing edge of the nylon membrane to the trailing edge of the section. The translation is in the direction of the tension force at the right-hand side of the 500^{th} nylon segment and the translation distance is equal to the length of the trailing edge strings. As described above, a guess is made at the start of each march about which hinge in the chain is the departure point. At the end of each march, a check is made to see if the selected departure point hinge is the best choice. Let's assume that Hinge #H was used as the departure point in the previous iteration. There are three possibilities which can occur when we try to calculate the revised shape:

   A. If the revised shape places Hinge #$H + 1$ physically inside the leading edge tube, then the departure point needs to be moved further aft. We should use Hinge #$H + 1$, or possibly even Hinge #$H + 2$, as the departure point in the next iteration.

   B. If the slope of the segment from Hinge #$H + 1$ to Hinge #$H + 2$ is greater than the slope of the circle at Hinge #$H + 1$, then the aerodynamic forces are trying to pull the membrane away from the surface of the tube. The departure point needs to be moved forward. We should use Hinge #$H - 1$, or possibly even Hinge #$H - 2$, as the departure point in the next iteration.

   C. If neither of the conditions in Cases A and B are met, then the hinge point we used in the previous iteration is still the best choice for the departure point.

3. Changes were made to module *WriteOpenFoamFunction.vb* which writes the text of the function (to be copied into the text of the controlDict file) which OpenFoam uses to determine how and when to print its force and moment calculations. Corresponding changes were made to module *ExtractOpenFoamForces.vb*, which reads the force and moment log file written by OpenFoam. After every 250^{th} iteration, OpenFoam writes a large number of two-line blocks, each consisting of 12 numbers, being the components of the pressure- and viscosity-induced force vectors and the pressure- and viscosity-induced moment vectors. If we let NumOfFlyingSegments be the number of free-flying nylon segments, then the number of blocks will be equal to $(2 \times \text{NumOfFlyingSegments}) + 3$. The blocks are written in the following specific order:

   A. One block being the total force and moment on the entire section,

B. NumOfFlyingSegments blocks, being the force and moment on each successive flying segment on the upper side of the nylon membrane, starting at the segment just after the departure point and ending at the one just before the trailing edge strings,

C. Another NumOfFlyingSegments blocks, being the force and moment on each successive flying segment on the lower side of the nylon membrane, starting at the segment just after the departure point and ending at the one just before the trailing edge strings,

D. One block being the total force and moment on all of the flying segments and

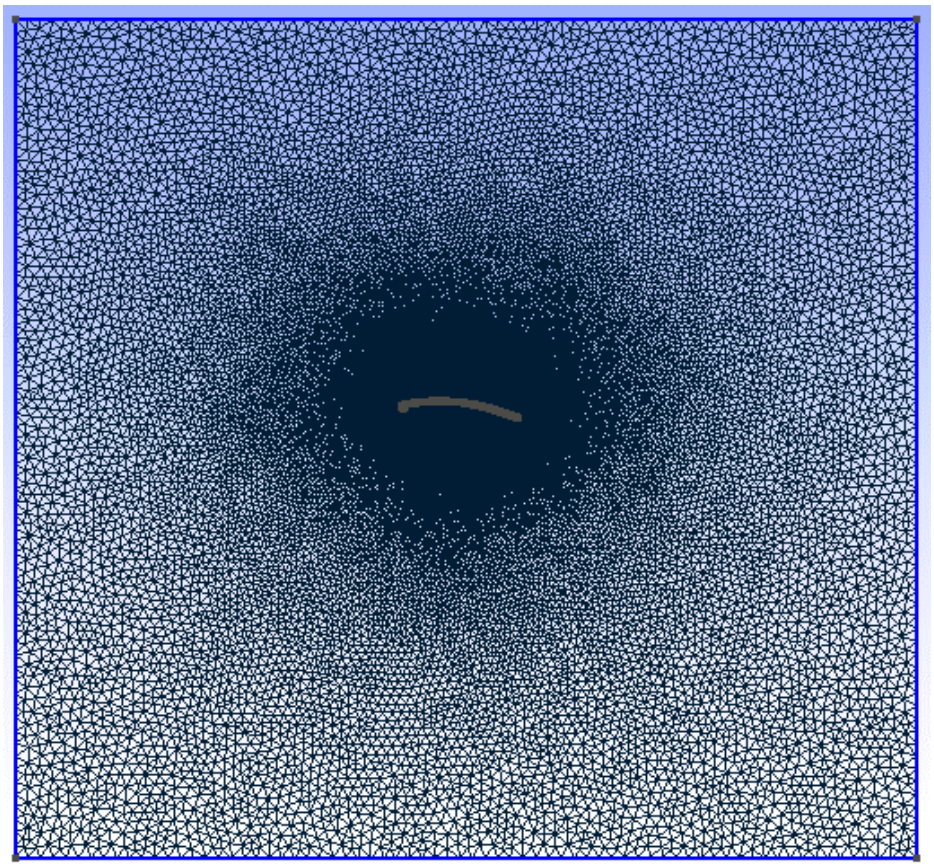
E. One block being the total force and moment on all of the segments which constitute the leading edge tube.

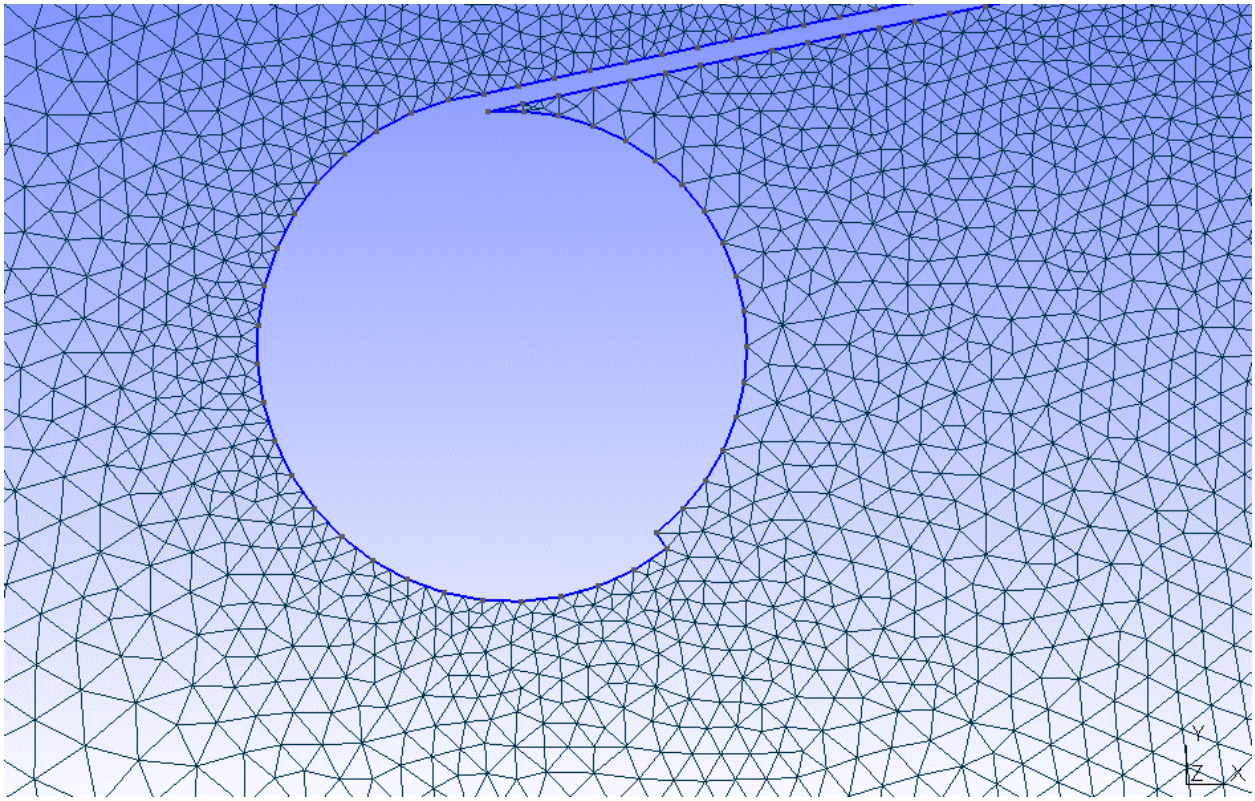
4. GMesh cannot mesh a two-dimensional region which includes in its interior a physical wall which is infinitely thin. In the earlier paper, I gave the membrane a small amount of thickness by translating the ideal shape upwards by one-half millimeter to define an upper surface and downwards by one-half millimeter to define a lower surface. Then, the two end-points were merged so that the leading and trailing edges became sharp. The module *WriteGMeshFile.vb* was modified to use a different algorithm to give the membrane a thickness of one millimeter. Now, the calculated shape of the membrane is used as the lower, or inner, surface. The upper, or outer, surface is created by translating the hinge points on the lower surface "outwards". Each hinge point (other than the last flying one, which is again brought to a sharp point) is translated by a distance of one millimeter, but it is translated in a direction which varies along the surface. The direction of translation is the average of the slopes of the two segments which join at the hinge. Creating the upper surface using this method has two advantages: (i) it allows the leading edge circle to fit exactly within the curve at the front of the membrane, and (ii) it can accommodate regions of the membrane where the slope is vertical, or reversed.

The virtual wind tunnel used by OpenFoam is the same rectangular parallelepiped as that used in the earlier paper. The top face is set a distance of three reference chord lengths above the leading edge of the profile, the bottom face is set a distance of three-and-one-half reference chord lengths below the leading edge, the upwind face, or "inlet", is set a distance of three reference chord lengths ahead of the leading edge and the downwind face, or "outlet", is set a distance of four reference chord lengths behind the leading edge. As before, the wind tunnel is given a thickness of one millimeter. The boundary conditions are based on the left and right faces being defined as "empty", the top and bottom faces being defined as "symmetryPlane"s and the inlet and outlet being defined as "patch"es. The inlet is treated as a constant velocity patch and the outlet is treated as a constant pressure patch. GMesh was instructed to create a triangular mesh with side lengths equal to 10 centimeters on the surface of the wind tunnel.

Since the airfoil is being treated as incompressible and steady, I have used OpenFoam's simpleFoam approximation of the Navier-Stokes equations. Since the airflow is two-dimensional and the object is fairly well streamlined, I have used the Spalart-Allmaras one-parameter model for the effects of turbulence. OpenFoam runs were continued until all residuals were less than 10^{-5}.

The nylon membrane has been divided into 500 segments for the purpose of calculating its shape. Only some of these, NumOfFlyingSeg in number, are actually free-flying and exposed to the air on both sides. The segments which are free-flying have both top and bottom surfaces, each of which is treated as a separate plane surface by OpenFoam. GMesh was instructed to create a triangular mesh in which each plane surface of the nylon membrane was divided in two, and constituted the bases of two neighbouring triangular prisms. I will say here in advance that the values of y^+ which resulted from this mesh size were in the range 5 - 20 for the base case. The following two figures show the mesh at the scale of the wind tunnel and, separately, in the vicinity of the leading edge tube.





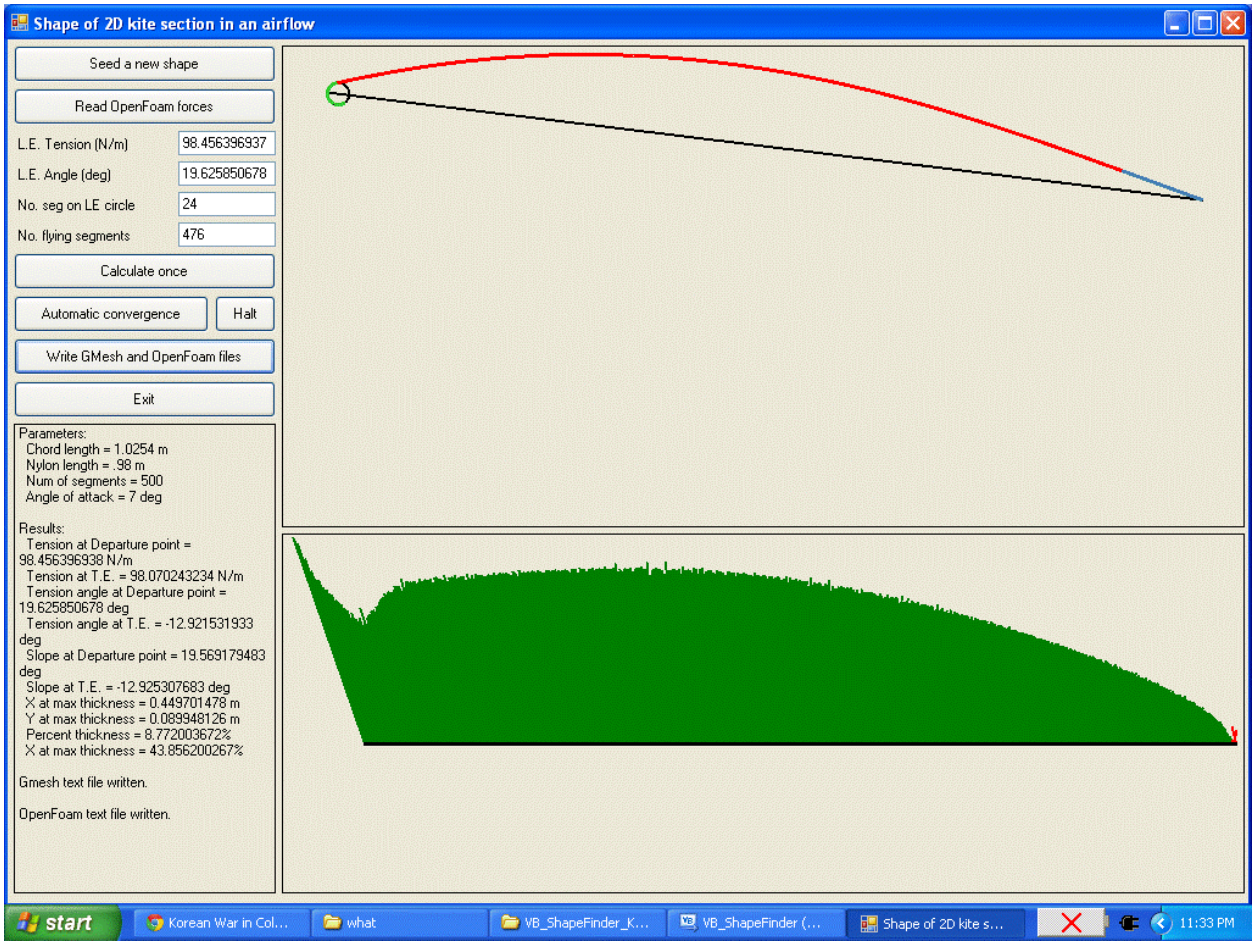
Plan of action

I want to set some direction for the study, as follows.

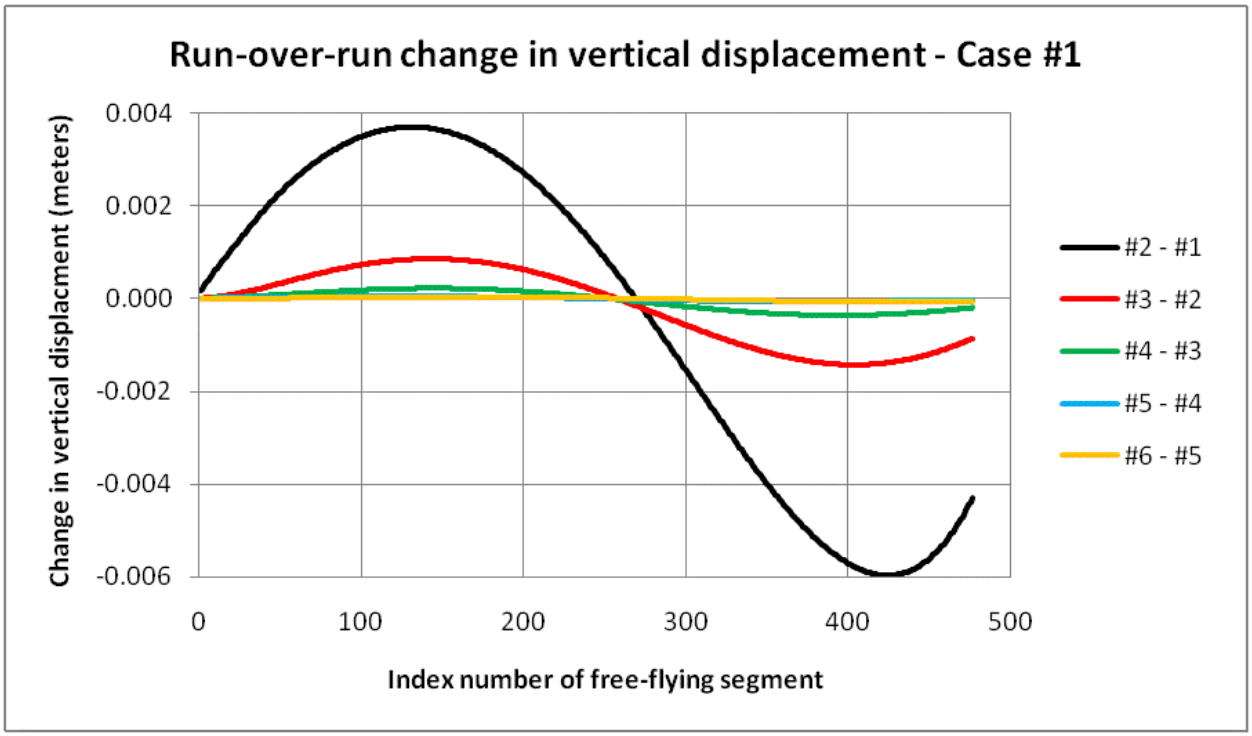
1. I will run a base case using the dimensions described above and assuming the kite is flying at or near sea level. I will assume the wind speed is 20 mph, which is a fairly stiff wind for kite flying. I do not want an angle of attack which is too small; I will use 7°.
2. I will then run the same case, with the same angle of attack and the same wind speed, but at an extremely high altitude -- 15,000 feet. I will use the air density and kinematic viscosity prescribed by the U.S. Standard Atmosphere for 15,000 feet.
3. I will then return to the base case at sea level. I will estimate weights for the various physical components.
4. I will design a two-line bridle which puts the kite into a static equilibrium at launch.
5. I will examine what happens with this bridle setting when the kite is flying at 15,000 feet..

Step #1: The base case -- 10 centimeter trailing edge strings, sea level and 20 mph wind speed

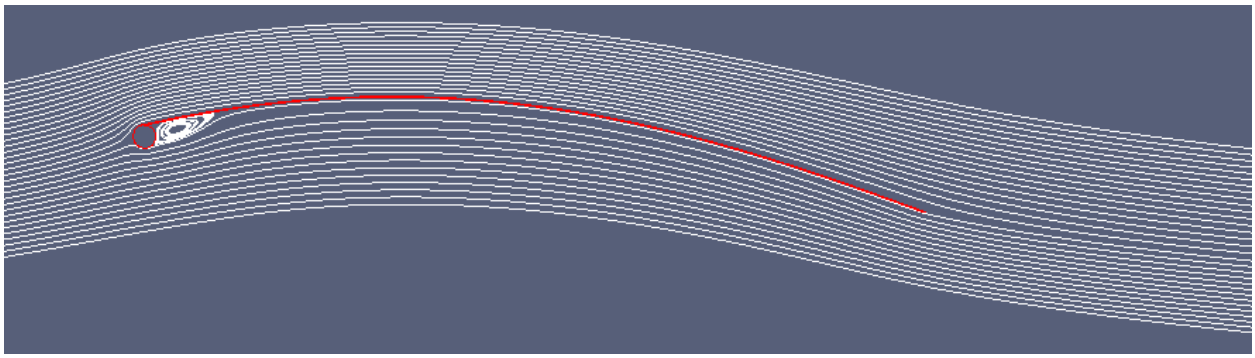
I ran five OpenFoam runs, starting with the circular arc shape described in Appendix "A" below and revising the shape after each OpenFoam run using the VisualBasic program listed in Appendix "C". The GUI after the final run was the following. The tension in the nylon membrane is just over 98 Newtons (approximately 25 pounds) per spanwise meter. The profile has a thickness / camber of 8.8% and the point of maximum thickness / camber is at 43.9% of chord.



The following graph shows how the shape converged during the course of the five OpenFoam runs. The horizontal axis represents distance along the membrane's surface starting from the departure point. The vertical axis is the change in the vertical displacement of the surface from the reference chord which takes place from one OpenFoam run to the next. As the shape converges, the run-over-run change becomes smaller and smaller.



The following picture shows some of the streamtracers around the base case profile. With the exception of a vortex behind the leading edge tube, which is expected, the pattern of the airflow resembles that of a classical rigid-construction airfoil.

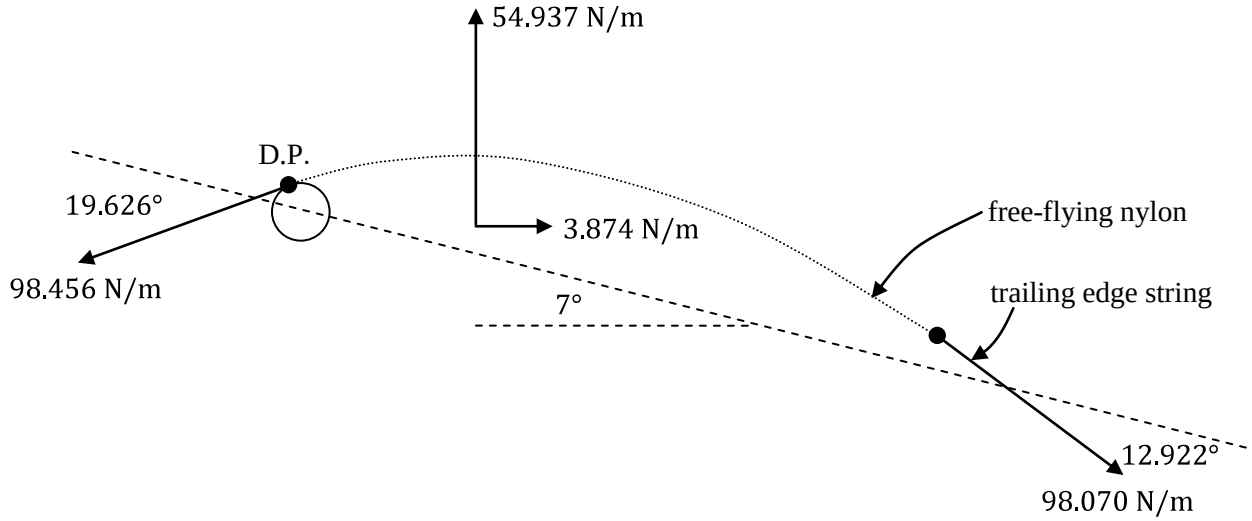


The following table summarizes the aerodynamic forces acting on the profile.

Force (N/spanwise meter)	On flying segments	On leading edge tube	Total	Lift and drag
$\hat{x}$-direction due to pressure	3.498	-0.320	3.178	Drag = 3.582
$\hat{x}$-direction due to viscosity	0.376	0.028	0.404	
$\hat{y}$-direction due to pressure	54.973	1.522	56.495	Lift = 56.474
$\hat{y}$-direction due to viscosity	-0.037	0.015	-0.021	

The lift-to-drag ratio is 56.474 / 3.582 = 15.8, including the impact of the leading edge tube.

As a reality check, it is worthwhile ensuring that the tension forces in the nylon membrane, calculated at the departure point and at the point where the membrane connects to the trailing edge strings, are consistent with the aerodynamic forces acting on the free-flying segments. This check is tantamount to checking the balance of forces acting on the nylon membrane as if it was a rigid body. The following figure shows the forces which should balance. Note that the lift and drag forces shown in this figure are those on the free-flying segments only, and do not include the forces on the leading edge tube.



The numerical values given for the aerodynamic forces are those calculated by OpenFoam during the fifth run, and which are set out in the above table. The tension magnitudes and angles are those derived when the shape was calculated after the fifth OpenFoam run, and are shown in the screenshot set out above. The sums of the forces in the vertical and horizontal directions are the following. The sums are equal to zero within the precision of the numbers used.

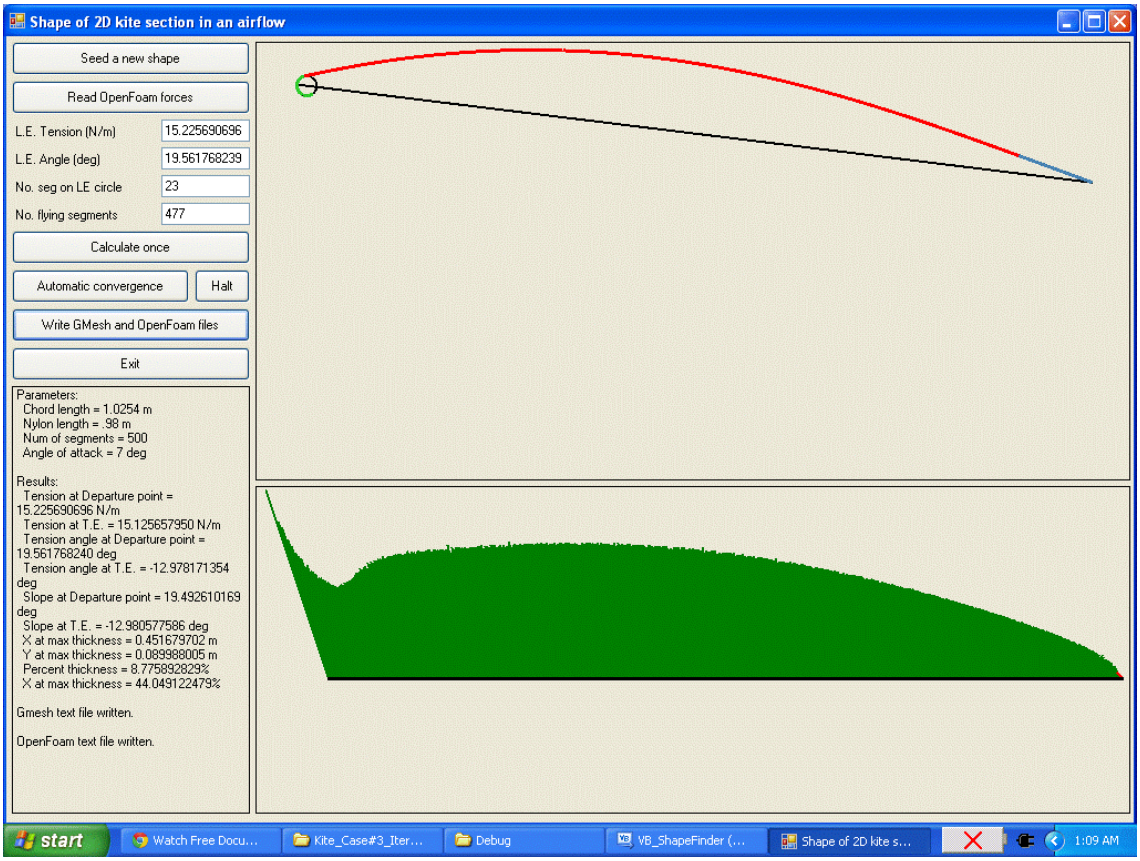
$$\text{Horizontal:} \quad \begin{bmatrix} -98.456\cos(19.626° - 7°) + \cdots \\ \cdots + 98.070\cos(12.922° + 7°) + \cdots \\ \cdots + 3.874 \end{bmatrix} = -0.00000001 \text{ N/m}$$

$$\text{Vertical:} \quad \begin{bmatrix} -98.456\sin(19.626° - 7°) + \cdots \\ \cdots - 98.070\sin(12.922° + 15°) + \cdots \\ \cdots + 54.937 \end{bmatrix} = 0.00000009 \text{ N/m}$$

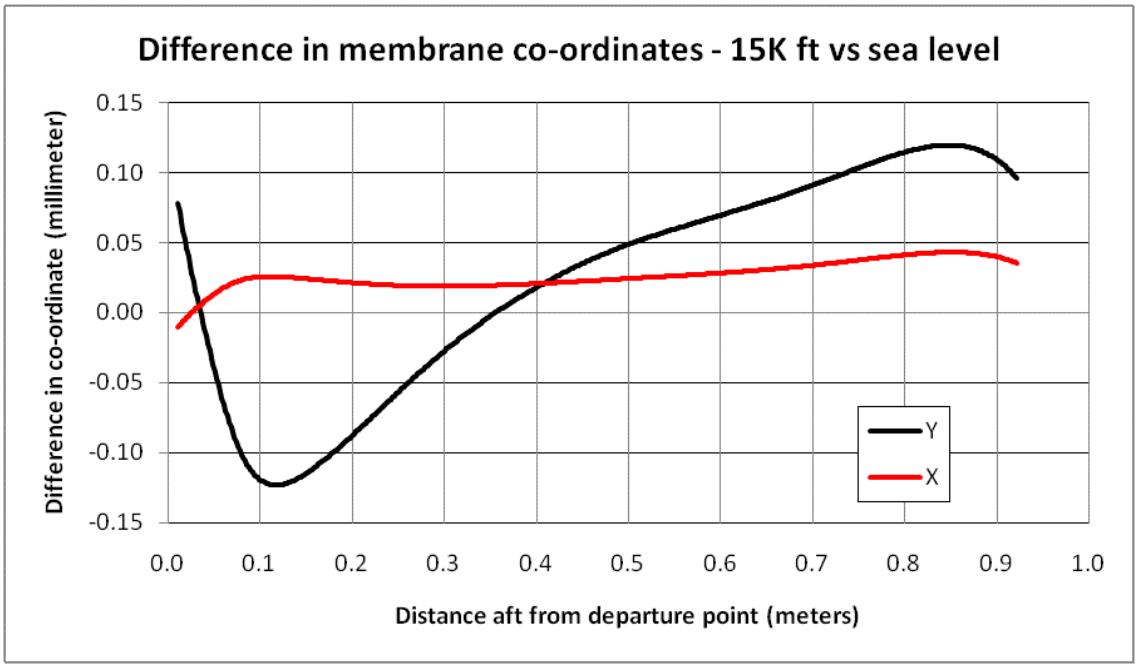
One Newton of force is very close to one-quarter pound, so the total lift of 56.474 Newtons is about 14 pounds. This is the force per meter of span. A kite with a span of six meters, or about 20 feet, would enjoy a lift of about 84 pounds.

Step #2: The same profile, angle of attack and wind speed but at 15,000 feet

In Step #2 of the analysis, I "flew" the kite at 15,000 feet. The kite has the same physical dimensions, with the camber determined by trailing edge strings 10 centimeters long. The angle of attack was left at 7° and the wind speed left at 20 mph. The following screenshot shows the shape of the profile and sets out the details of the tension forces after five OpenFoam runs.



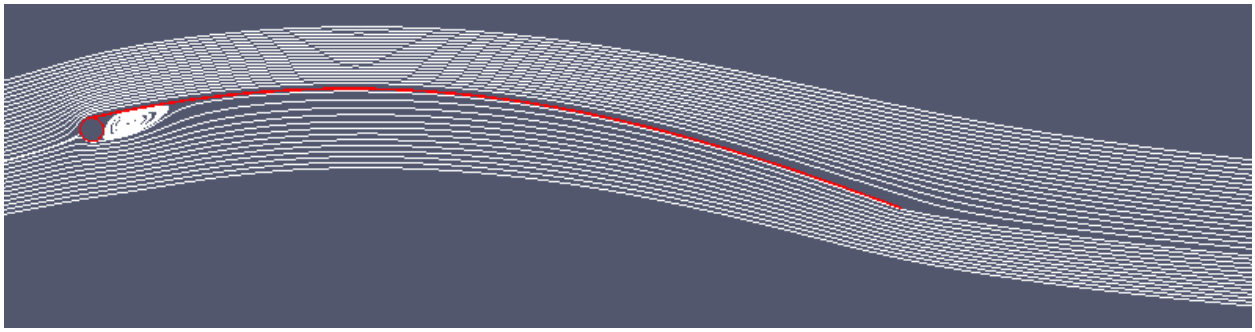
The thinness of the air at 15,000 feet means that the aerodynamic forces acting on the membrane are much less than at sea level. This translates into much lower tension forces in the membrane. Rather than the 98 N/m we saw at sea level, the tension forces are now approximately 15 N/m. Of considerable interest is how the shape of the membrane has changed from the shape at sea level. The following graph shows the difference. The horizontal axis represents distance along the reference chord starting from the departure point. The vertical axis is the difference between the (x, y) co-ordinates of a particular hinge at 15,000 feet and at sea level. The displacement in the $\hat{y}$-direction, perpendicular to the reference chord, is rendered in black. The displacement in the $\hat{x}$-direction parallel to the reference chord, is rendered in red.



The first thing to note is that the vertical scale is measured in millimeters. The most extreme displacements are almost all less than one-tenth of a millimeter. This is very small. Remember that the nylon membrane itself is assumed to be one millimeter thick, about three times thicker than the entire vertical axis in the display.

For all practical purposes, the shape of the membrane at 15,000 feet is the same as at sea level. Take care, though, both cases were run using the same wind speed. The only differences in the air are: (i) a reduction in density from 1.225 kg/m^3 at sea level to 0.1948 kg/m^3 at 15,000 feet and (ii) an increase in the kinematic viscosity from 1.4604E-5 m^2/s at sea level to 7.2998E-5 m^2/s at 15,000 feet.

The following picture shows streamtracers around the profile at the higher altitude. The pattern of the airflow is the same as in the base case, at least as far as can be discerned by eye.



The following table summarizes the aerodynamic forces at 15,000 feet.

Force (N/spanwise meter)	On flying segments	On leading edge tube	Total	Lift and drag
$\hat{x}$-direction due to pressure	0.548	-0.011	0.537	Drag = 0.645
$\hat{x}$-direction due to viscosity	0.098	0.011	0.108	
$\hat{y}$-direction due to pressure	8.487	0.218	8.705	Lift = 8.704
$\hat{y}$-direction due to viscosity	-0.008	0.006	-0.002	

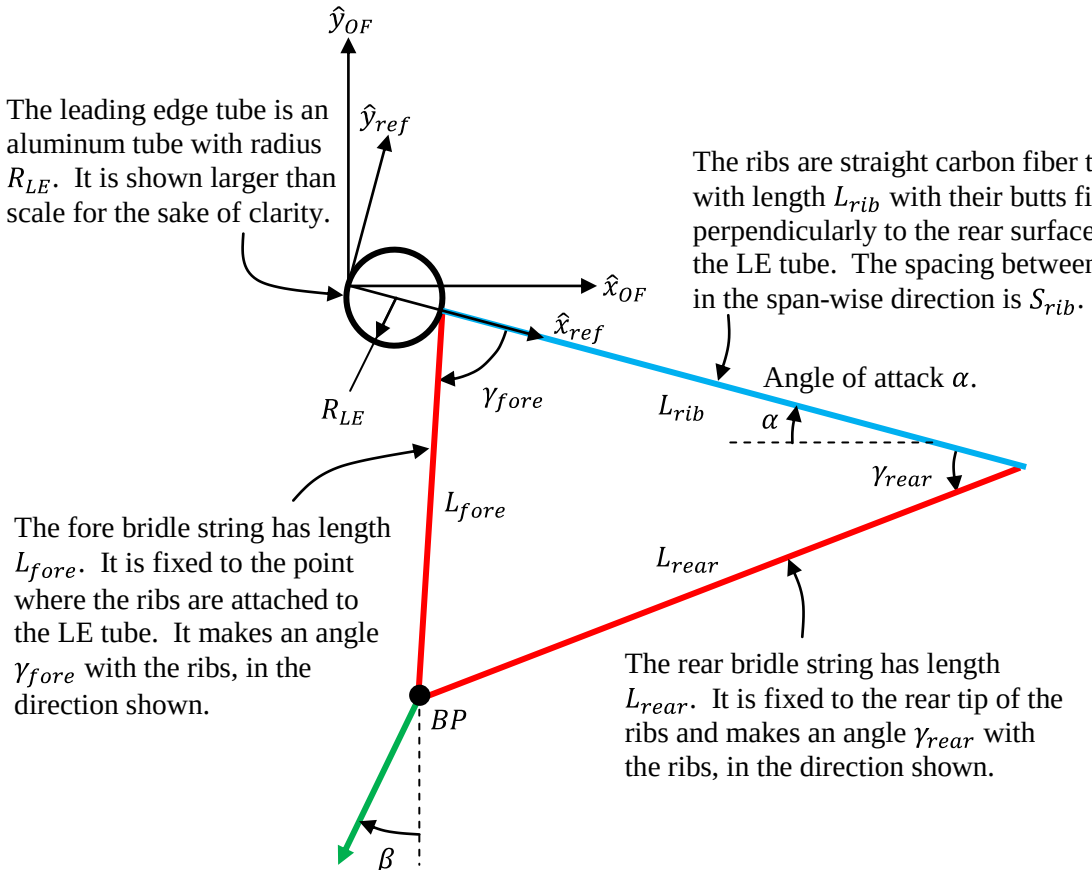
The lift-to-drag ratio is 8.704 / 0.645 = 13.5, including the impact of the leading edge tube. This is a slight reduction from the 15.8 ratio which obtained at sea level. Much more significant is the reduction in the magnitude of the lift, from 56.5 N/spanwise meter at sea level to only 8.7 N/spanwise meter at 15,000 feet.

Steps #3 and #4: Weight and balance at sea level

In Appendix "B" attached, I have made estimates of the mass of the various components which would be used to build this kite. I will not repeat those details here, other than to summarize the force of gravity (i.e., weight) on the various components, expressed in terms of Newtons per meter of span:

Component	Force of gravity	as percentage
Aluminum leading edge tube	1.62	49.8%
Carbon fibre ribs	0.23	6.9%
Nylon membrane	0.69	21.1%
Hardware	0.72	22.2%
Total	3.26	100.0%

Appendix "B" takes the matter further than this. It defines all of the forces, as well as the points-of-action of the forces, which are also needed to set up the equations for static equilibrium. This includes the dimensioning of a two-line bridle. The fore line of the bridle will be attached to the kite section at the point of intersection of the ribs and the leading edge tube. The aft line of the bridle will be attached to the aft ends of the ribs. The following figure summarizes some of the quantities defined in Appendix "B".



The leading edge tube is an aluminum tube with radius R_{LE}. It is shown larger than scale for the sake of clarity.

The ribs are straight carbon fiber tubes with length L_{rib} with their butts fixed perpendicularly to the rear surface of the LE tube. The spacing between ribs in the span-wise direction is S_{rib}.

Angle of attack α.

The fore bridle string has length L_{fore}. It is fixed to the point where the ribs are attached to the LE tube. It makes an angle γ_{fore} with the ribs, in the direction shown.

The rear bridle string has length L_{rear}. It is fixed to the rear tip of the ribs and makes an angle γ_{rear} with the ribs, in the direction shown.

The tether, shown in green, is the line which extends from the bridle point (BP) to the ground. It makes an angle β with the vertical, in the direction shown, where it is attached to the bridle point.

Our goal, at this point, is to specify a bridle which will put the kite into static equilibrium at the desired angle of attack (7°) in a 20 mph wind at sea level. These flight conditions are the ones used in the base case. In other words, we are going to trim the kite for flight. The factors we have to play with are the lengths of the two bridle lines, which are rendered in red in the figure. They are brought together at the bridle point BP, where they are attached to the tether, which is shown in green in the figure.

During flight, the two bridle lines will be stretched taut (hopefully) by the tension forces which they transmit from the kite's structure down to the bridle point. We can think of the two bridle lines and the carbon fibre rib, which is rendered in blue in the figure, as a rigid triangle. Selecting the lengths of the two bridle lines is therefore tantamount to selecting the length L_{fore} and relative angle γ_{fore} of the forward bridle line. The length L_{rear} and relative angle γ_{rear} of the after bridle line are then determined by the geometry. Fixing any two of the four variables fixes the other two.

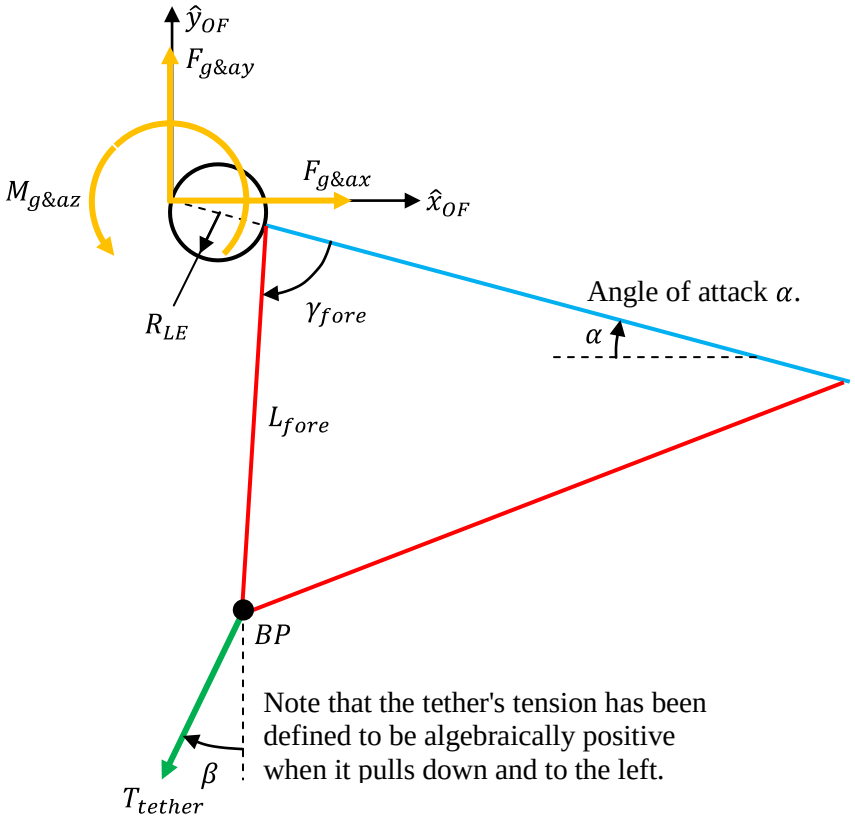
The easiest way to attack the problem of static equilibrium is to first add up the various gravitational forces and aerodynamic forces which act on the kite's section. As can be seen from a review of Appendix

"B", some of the forces are expressed in the OpenFoam frame of reference $(\hat{x}_{OF}, \hat{y}_{OF})$ and others are expressed in the kite-fixed frame of reference $(\hat{x}_{ref}, \hat{y}_{ref})$. The same holds true for the points-of-action of the forces. Adding up the forces requires that they be expressed in a common frame of reference. For the purposes of this section, I will express all of the forces in the OpenFoam frame of reference. It is also useful to gather the forces together at a common point-of-action, with a compensating mechanical moment. I have chosen to gather them together at the leading edge of the reference chord, which happens to be the origin of both frames of reference.

For a given flight condition, as has been assumed here, all of the forces acting on the kite's section are known expect for those exerted by the bridle. Adding up the gravitational and aerodynamic forces and expressing them as one makes it easier to focus on the remaining unknowns, all of which relate to the bridle. I will use the symbols $\vec{F}_{g\&a}$ and $\vec{M}_{g\&a}$ for the consolidated force and moment from all gravitational and aerodynamic ("g&a") sources. Since we are dealing with a two-dimensional case, the aggregate force will have only two components -- $F_{g\&ax}$ and $F_{g\&ay}$ -- and the moment will have only one -- $M_{g\&az}$ -- where the directions indicated in the subscripts are those of the axes in the OpenFoam frame of reference. In the base case, the sum of the gravitational and aerodynamic forces per meter of span are as follows:

Force or moment	Gravitational	Aerodynamic	Total
$F_{g\&ax}$	0.000	3.582	3.582
$F_{g\&ay}$	-3.260	56.474	53.214
$M_{g\&az}$	-0.726	22.135	21.409

The free body diagram of the kite-bridle combination looks like this. The sum of the gravitational and aerodynamic effects are rendered in orange. If they are exactly balanced by the tension in the tether, the kite will be in a static equilibrium. (This is not to say that the equilibrium will be stable. That is a different matter which we will look at in due course.) The tension force in the tether is rendered in green. We need to calculate the magnitude and direction of the tether's tension, and its point-of-action, which bring things into balance. We will find, in fact, that there is a range of possibilities at which this happens.



In order for the kite to be in static equilibrium, the tether must pull down with exactly the same magnitude as the combined gravitational and aerodynamic forces pull up. The tether must also pull towards the left with exactly the same magnitude as the combined gravitational and aerodynamic forces pull to the right. In addition, the tension in the tether must exert a mechanical moment around the leading edge which exactly counteracts the moment which the combined gravitational and aerodynamic forces impose. The first two conditions represent the balance of forces in the vertical and horizontal directions, respectively. We can write them mathematically as follows:

$$\left.\begin{array}{ll} \text{Horizontal:} & T_{tether}\sin\beta = F_{g\&ax} \\ \text{Vertical:} & T_{tether}\cos\beta = F_{g\&ay} \end{array}\right\} \quad (1)$$

These two equations can be solved to give both the magnitude and direction angle of the tether's tension.

$$\left.\begin{array}{rl} T_{tether} & = \sqrt{F_{g\&ax}^2 + F_{g\&ay}^2} \\ \beta & = \tan^{-1}\left(\dfrac{F_{g\&ax}}{F_{g\&ay}}\right) \end{array}\right\} \quad (2)$$

The numerical values for the base case are as follows:

$$\begin{array}{rll} T_{tether} & = \sqrt{3.582^2 + 53.214^2} & = 53.334\ \text{N/m} \\ \beta & = \tan^{-1}\left(\dfrac{3.582}{53.214}\right) & = 3.851° \end{array}$$

If the tether is short, the kite will be flying almost straight overhead, being less than 4° downwind from the vertical.

We have one remaining equilibrium condition, relating to the equality of moments. This condition can also be written as a mathematical expression. Note, however, that we still have two unknown quantities, being the length and angle of the forward bridle line. One equation cannot be solved uniquely for two unknowns, so we will end up with a range of possible length-angle pairs which are consistent with equilibrium.

Writing down the expression for the net moment is more difficult than doing so for the force components. Let's start with the location of the bridle point. In the kite-fixed frame of reference, the co-ordinates of the bridle point are:

$$\overrightarrow{BP} = \left(2R_{LE} + L_{fore}\sin\gamma_{fore}\right)\hat{x}_{ref} - L_{fore}\sin\gamma_{fore}\,\hat{y}_{fore} \quad (3)$$

In the kite-fixed frame of reference, the vector representing the tether's tension can be written as:

$$\vec{T}_{tether} = -T_{tether}\sin(\beta-\alpha)\,\hat{x}_{ref} - T_{tether}\cos(\beta-\alpha)\,\hat{y}_{ref} \quad (4)$$

The mechanical moment can be written as the vector cross-product of these two vectors, thus:

$$\vec{M}_{tether} = \overrightarrow{BP} \vec{\times} \vec{T}_{tether} \quad (5)$$

It is not necessary that $\overrightarrow{BP}$ and $\vec{T}_{tether}$ be transformed (rotated) into the OpenFoam frame of reference. In Equations (3) and (4), both vectors are expressed in the kite-fixed frame of reference. Since the ingredients are confined to the $\hat{x}$ and $\hat{y}$ axes, the resultant from Equation (5) will be entirely in the $\hat{z}$-direction. It happens that the $\hat{z}$-axes of the OpenFoam and kite-fixed frames of reference are identical, so the mechanical moment will have exactly the same value in both frames.

Evaluating the cross-product in the two-dimensional case is straight-forward (see Equation (B13) in Appendix "B") and is as follows:

$$\begin{aligned} M_{tetherz} &= BP_x T_{tethery} - BP_y T_{tetherx} \\ &= \begin{bmatrix} -\left(2R_{LE} + L_{fore}\cos\gamma_{fore}\right)T_{tether}\cos(\beta-\alpha) + \cdots \\ \cdots - L_{fore}\sin\gamma_{fore}\,T_{tether}\sin(\beta-\alpha) \end{bmatrix} \\ &= -\begin{Bmatrix} 2R_{LE}T_{tether}\cos(\beta-\alpha) + \cdots \\ \cdots + L_{fore}T_{tether}\left[\cos\gamma_{fore}\cos(\beta-\alpha) + \sin\gamma_{fore}\sin(\beta-\alpha)\right] \end{Bmatrix} \\ &= -\begin{bmatrix} 2R_{LE}T_{tether}\cos(\beta-\alpha) + \cdots \\ \cdots + L_{fore}T_{tether}\cos\left(\gamma_{fore}-\beta+\alpha\right) \end{bmatrix} \\ &= -T_{tether}\left[2R_{LE}\cos(\beta-\alpha) + L_{fore}\cos\left(\gamma_{fore}-\beta+\alpha\right)\right] \end{aligned} \qquad (6)$$

We want the net moment exerted on the kite-bridle body to be zero, so that:

$$\begin{aligned} & M_{tetherz} + M_{g\&az} = 0 \\ \rightarrow \quad & T_{tether}\left[2R_{LE}\cos(\beta-\alpha) + L_{fore}\cos\left(\gamma_{fore}-\beta+\alpha\right)\right] = M_{g\&az} \\ \rightarrow \quad & L_{fore}\cos\left(\gamma_{fore}-\beta+\alpha\right) = \frac{M_{g\&az}}{T_{tether}} - 2R_{LE}\cos(\beta-\alpha) \end{aligned} \qquad (7)$$

For a given flight condition, as has been assumed here, all of the quantities on the right-hand side are known. Any pair of L_{fore} and γ_{fore} which gives the same left-hand side value represents a state of static equilibrium. If we pick a value for one of the unknowns, the other can be calculated using one of the following expressions:

$$\text{pick } L_{fore} \rightarrow \quad \gamma_{fore} = \cos^{-1}\left\{\frac{1}{L_{fore}}\left[\frac{M_{g\&az}}{T_{tether}} - 2R_{LE}\cos(\beta-\alpha)\right]\right\} + \beta - \alpha \qquad (8A)$$

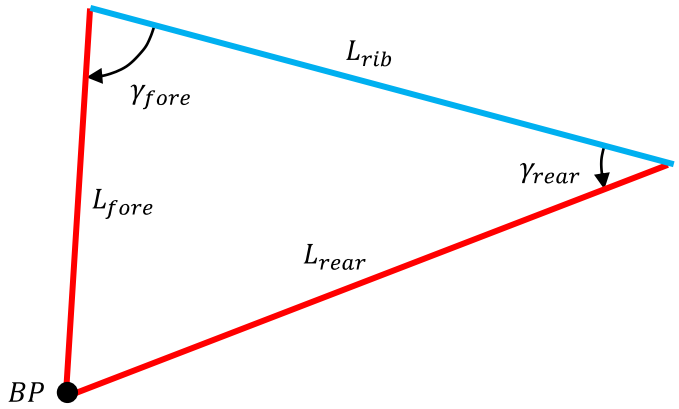
$$\text{pick } \gamma_{fore} \rightarrow \quad L_{fore} = \frac{1}{\cos\left(\gamma_{fore}-\beta+\alpha\right)}\left[\frac{M_{g\&az}}{T_{tether}} - 2R_{LE}\cos(\beta-\alpha)\right] \qquad (8B)$$

Let's work through a numerical example relating to the base case. Suppose we propose to use a forward bridle line which is four meters long, so $L_{fore} = 4$ and

$$\begin{aligned} \gamma_{fore} &= \cos^{-1}\left\{\frac{1}{4}\left[\frac{21.409}{53.334} - 2\times 0.0127\times\cos(3.851^\circ - 7^\circ)\right]\right\} + 3.851^\circ - 7^\circ \\ &= \cos^{-1}(0.0940) + 3.851^\circ - 7^\circ \\ &= 84.606^\circ + 3.851^\circ - 7^\circ \\ &= 81.457^\circ \end{aligned}$$

Follow-up task #1: Calculating the length and angle of the rear bridle line

While it is not essential that we do so now, this is a convenient place to complete the calculations related to the bridle. Now that we have calculated the details of the forward bridle line, the details of the rear bridle line can be calculated from the geometry of the following triangle.



The triangle is not a right triangle so we must use the more-general sine and cosine laws for triangles to extract the information we want. We will invoke the cosine law first, which will allow us to calculate the length of the rear bridle line:

$$L_{rear}^2 = L_{fore}^2 + L_{rib}^2 - 2L_{fore}L_{rib}\cos\gamma_{fore}$$

$$\rightarrow \quad L_{rear} = \sqrt{L_{fore}^2 + L_{rib}^2 - 2L_{fore}L_{rib}\cos\gamma_{fore}} \qquad (9)$$

Now having that length we can use the sine law to calculate angle γ_{rear}:

$$\frac{\sin\gamma_{rear}}{L_{fore}} = \frac{\sin\gamma_{fore}}{L_{rear}}$$

$$\rightarrow \quad \gamma_{rear} = \sin^{-1}\left(\frac{L_{fore}}{L_{rear}}\sin\gamma_{fore}\right) \qquad (10)$$

The numerical values for the base case, using a forward bridle length of four meters, are:

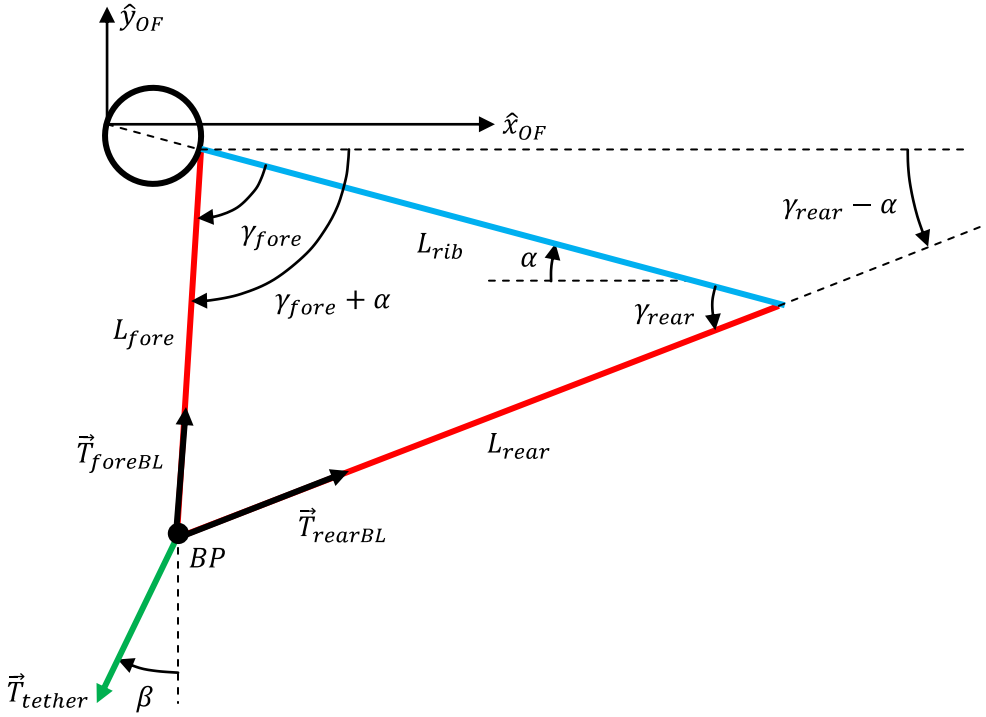
$$L_{rear} = \sqrt{4^2 + 1^2 - [2 \times 4 \times 1 \times \cos(81.457°)]} = 3.976 \text{ m}$$

$$\gamma_{rear} = \sin^{-1}\left[\frac{4}{3.976}\sin(81.457°)\right] = 84.143°$$

Follow-up task #2: Calculating the tension forces in the two bridle lines

We now have the numbers we need to calculate the tension forces in the bridle lines. Since the kite-bridle body has been put into a static equilibrium, every part of the body must be in its own individual static equilibrium. Physically, the bridle point will likely be a small metal ring to which bridle lines from the rib(s) are attached along the top side and to which the tether is tied on the bottom side. We can treat the bridle point as its own little rigid body, upon which there cannot be any net force. Only three forces act on it: the tension in the tether and the tensions in each of the bridle lines. Each of these tensions can act only along the central axis of its corresponding line. The following figure shows the geometry of the

forces acting on the bridle point. In essence, we are going to do nothing more than resolve the tether's tension into separate components along the two bridle lines.



The horizontal components of the three tension forces, expressed in the OpenFoam frame of reference, are:

$$\begin{aligned} T_{tetherx} &= -T_{tether} \sin\beta \\ T_{foreBLx} &= -T_{foreBL} \cos(\gamma_{fore} + \alpha) \\ T_{rearBLx} &= T_{rearBL} \cos(\gamma_{rear} - \alpha) \end{aligned}$$

The vertical components are:

$$\begin{aligned} T_{tethery} &= -T_{tether} \cos\beta \\ T_{foreBLy} &= T_{foreBL} \sin(\gamma_{fore} + \alpha) \\ T_{rearBLy} &= T_{rearBL} \sin(\gamma_{rear} - \alpha) \end{aligned}$$

The sum of the horizontal force components must be zero and the sum of the vertical force components must be zero. This gives us the following two equalities:

$$T_{tether} \sin\beta = -T_{foreBL} \cos(\gamma_{fore} + \alpha) + T_{rearBL} \cos(\gamma_{rear} - \alpha) \quad (11A)$$

$$T_{tether} \cos\beta = T_{foreBL} \sin(\gamma_{fore} + \alpha) + T_{rearBL} \sin(\gamma_{rear} - \alpha) \quad (11B)$$

These constitute two equations in the two unknowns T_{fore} and T_{rear}. They are most easily solved if we combine them in two separate ways and then compare the results. First, let's multiply Equation $(11A)$ by $\sin(\gamma_{fore} + \alpha)$ and multiply Equation $(11B)$ by $\cos(\gamma_{fore} + \alpha)$ and then add them. The terms in T_{foreBL} will cancel each other out, leaving:

$$T_{tether}\begin{bmatrix}\sin\beta\sin(\gamma_{fore}+\alpha)+\cdots\\ \cdots+\cos\beta\cos(\gamma_{fore}+\alpha)\end{bmatrix}=T_{rearBL}\begin{bmatrix}\cos(\gamma_{rear}-\alpha)\sin(\gamma_{fore}+\alpha)+\cdots\\ \cdots+\sin(\gamma_{rear}-\alpha)\cos(\gamma_{fore}+\alpha)\end{bmatrix}$$

$$\rightarrow \quad T_{tether}\cos[\beta-(\gamma_{fore}+\alpha)]=T_{rearBL}\sin[(\gamma_{fore}+\alpha)+(\gamma_{rear}-\alpha)]$$

$$\rightarrow \quad T_{rearBL}=T_{tether}\frac{\cos(\beta-\gamma_{fore}-\alpha)}{\sin(\gamma_{fore}+\gamma_{rear})} \qquad (12)$$

The second combination is to multiply Equation $(11A)$ by $\sin(\gamma_{rear}-\alpha)$ and to multiply Equation $(11B)$ by $\cos(\gamma_{rear}-\alpha)$ and then to subtract the latter from the former. The terms in T_{rearBL} will cancel each other out, leaving:

$$T_{tether}\begin{bmatrix}\sin\beta\sin(\gamma_{rear}-\alpha)+\cdots\\ \cdots-\cos\beta\cos(\gamma_{rear}-\alpha)\end{bmatrix}=-T_{foreBL}\begin{bmatrix}\cos(\gamma_{fore}+\alpha)\sin(\gamma_{rear}-\alpha)+\cdots\\ \cdots+\sin(\gamma_{fore}+\alpha)\cos(\gamma_{rear}-\alpha)\end{bmatrix}$$

$$\rightarrow \quad -T_{tether}\cos[\beta+(\gamma_{rear}-\alpha)]=-T_{foreBL}\sin[(\gamma_{fore}+\alpha)+(\gamma_{rear}-\alpha)]$$

$$\rightarrow \quad T_{foreBL}=T_{tether}\frac{\cos(\beta+\gamma_{rear}-\alpha)}{\sin(\gamma_{fore}+\gamma_{rear})} \qquad (13)$$

Equations (12) and (13) give the tensions in the two bridle lines, as desired. (When deriving these expressions I have made free use of the trigonometric identities $\sin(P+Q)=\sin P\cos Q+\cos P\sin Q$ and $\cos(R+S)=\cos R\cos S-\sin R\sin S$.)

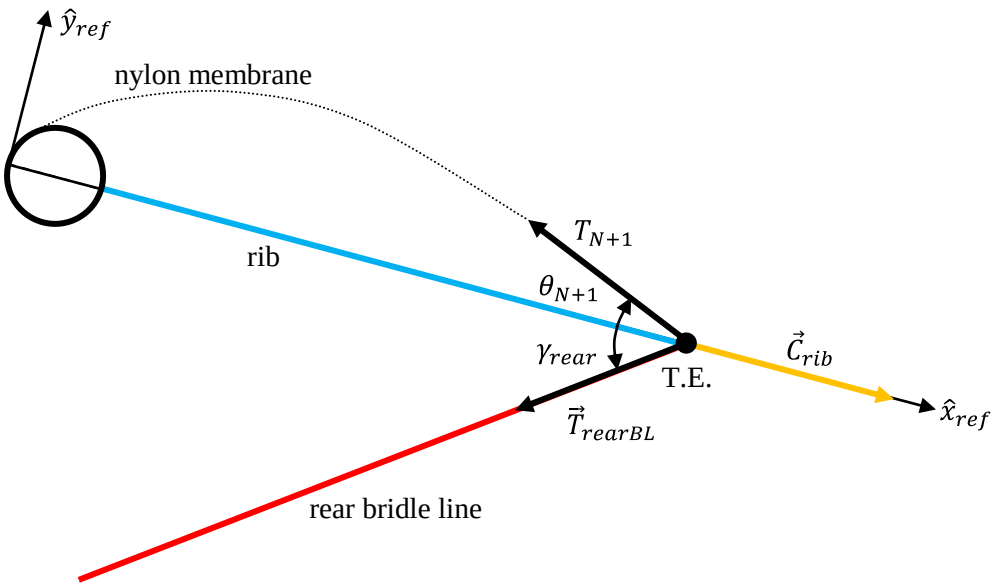
The numerical values for the base case, using a forward bridle length of four meters, are:

$$T_{rearBL} = 53.334\frac{\cos(3.851°-81.457°-7°)}{\sin(81.457°+84.143°)} = 20.161\ \text{N/m}$$

$$T_{foreBL} = 53.334\frac{\cos(3.851°+84.143°-7°)}{\sin(81.457°+84.143°)} = 33.570\ \text{N/m}$$

Follow-up task #3: Calculating the compression force in the ribs

We are not yet finished. We can estimate the force which tends to compress the ribs. Generally speaking, the tension forces in the free-flying nylon membrane are such that they pull the two ends of the ribs together. In the same way, the tension forces in the two bridle lines also tend to pull the two ends of the ribs together. This will put the ribs into a state of compression. The only transverse loading on the ribs is gravitational. This is much smaller than the other forces and I will ignore it for the purposes of this task. There will also be a small mechanical moment applied to the front end of the ribs, arising from the asymmetry of the aerodynamic forces whivh act on the leading edge tube. I will also ignore this moment for the purposes of this task. Physically, the ribs are very thin compared with their length. As a starting point, I will assume that the compression force acts along the longitudinal axis of the ribs. The following figure is a zoomed-in view of the trailing edge of the ribs, showing the principal forces which act on the very tip. I will treat the tip of the trailing edge, identified by "T.E.", as a discrete and independent body. It is subjected to the tension force $\vec{T}_{rearBL}$ in the rear bridle line, which pulls if downwards and to the left. It is subject to the tension force in the trailing edge string, which pull it upwards and to the left. The magnitude and direction of the trailing edge strings are things I described above, and are represented by T_{N+1} and θ_{N+1} where, as above, the subscript identifies the right-most hinge of the last segment of the nylon membrane. Lastly, the tip of the trailing edge has to resist the compression in the ribs. The trailing edge pushes in on the ribs; in response, the ribs push back, towards the lower left.



It is convenient to use the kite-fixed frame of reference for these calculations since the relevant angles are already defined with respect to the ribs / reference chord. If the trailing edge tip is to remain in its relative position, then the sum of the forces acting on it must be zero. The following expressions show the sum of the force components parallel to the $\hat{x}_{ref}$ axis being set equal to zero and the sum of the force components perpendicular to the $\hat{x}_{ref}$ axis being set equal to zero, respectively:

$$\begin{aligned} &\text{Parallel:} && C_{rib} = T_{N+1}\cos\theta_{N+1} + T_{rearBL}\cos\gamma_{rear} && (14A) \\ &\text{Perpendicular:} && T_{N+1}\sin\theta_{N+1} - T_{rearBL}\sin\gamma_{rear} = 0 && (14B) \end{aligned}$$

We can use Equation $(14A)$ to calculate the compression force in the ribs. The numerical values for the base case, using a forward bridle length of four meters, is:

$$\begin{aligned} C_{rib} &= (98.070\cos 12.922°) + (20.038\cos 56.129°) \\ &= 95.586 + 11.168 \\ &= 106.754\ \text{N/m} \end{aligned}$$

We can use Equation $(14B)$ to estimate the validity of our assumption to ignore the mechanical moment.

$$\begin{aligned} L.H.S. &= (98.070\sin 12.922°) - (20.038\sin 56.129°) \\ &= 21.931 - 16.637 \\ &= 5.293\ \text{N/m} \\ &\approx 0 \end{aligned}$$

I already mentioned that the happenings at the leading edge cause there to be a mechanical moment applied at the forward end of the ribs. For a static equilibrium to exist, there must be a small non-axial component of the force at the trailing edge of the ribs. Our preliminary estimate is that the non-axial force will be about 5 N/spanwise meter, which is indeed small compared with the principal force resisted by the ribs, which is a compression load of about 107 N/spanwise meter.

Step #4: Weight and balance at 15,000 feet

We will begin this step of the analysis by assuming that the base kite kite has been trimmed for flight at an angle of attack of 7° by setting the lengths of the forward and rear bridle lines to four meters and 4.079 meters, respectively. This puts the kite into a static equilibrium when it is facing a 20 mph wind at sea level. Now, we release the kite and let it ascend. What will happen?

Suppose we pay out the tether very slowly. We give the kite time to adjust to changing conditions as it climbs. We can imagine the kite passing through a continuous series of static equilibria on its way up. The flexible membrane has time to adjust its shape in response to changing wind speed and changing density and viscosity. The kite has time to adjust its angle of attack. Changes in the lift and drag have time to work their way through the dynamics of the kite and bridle and allow the kite to remain in conditions always close to equilibrium. Because the kite has mass, it cannot respond instantaneously to changes in the forces acting on it. The forces cause acceleration, the acceleration leads to speed, the speed leads to new locations and angles, and all the while these changes are occurring, the applied forces are themselves changing.

The alternative is to pay the tether out quickly, or even to let go of it entirely. Without the restraining tension of the tether, the kite will respond like a piece of paper at the mercy of the wind. It will violate one or more of the three basic assumptions on which we have based this analysis.

1. That the airflow is steady. This is a fundamental assumption of the OpenFoam simulations. If the relative airflow is unsteady -- which it will be if the kite itself is pitching or heaving -- the nylon membrane will have a constantly changing shape.
2. That the angle of attack is fixed with respect to the oncoming wind. The results we have used from the OpenFoam simulations are based on more than just steady airflow. The results are valid only for the angles of attack used. Suppose the kite is flown in a 20 mph wind, as assumed in the base case. As the kite climbs, the "effective wind" it experiences is no longer 20 mph. The upward speed of the kite must be taken into account. As seen from the kite's point-of-view, the oncoming wind has a downward component as well as the 20 mph horizontal component. From the kite's point-of-view, the effective angle of attack is less than the 7° to which it was trimmed on the ground. In a similar way, any downwind ground speed the kite may have will reduce the effective wind speed. Both of these motions -- upward climb and downrange speed -- will reduce the lift experienced by the kite, or worse.
3. We used an analysis of static equilibrium to guide our setting of the bridle. Static equilibrium is by definition steady. If changes in the prevailing conditions take place too quickly, or are too large, equilibrium is not guaranteed.

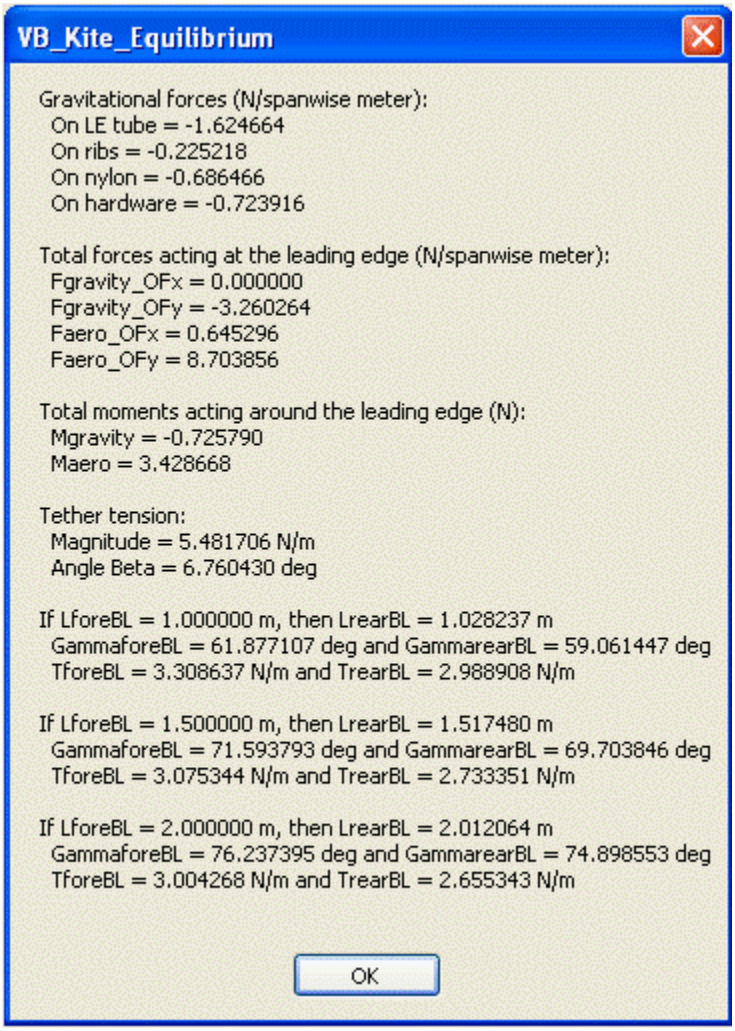
The lesson is this: changes in flight must be made slowly.

Let's assume we pay the tether out slowly and let the kite climb gradually. Could this kite ever get to an altitude of 15,000 feet? In theory, yes. The OpenFoam simulation in Step #2 above showed that the aerodynamic lift on the kite at 15,000 feet is 8.70 N/spanwise meter. In Step #3, I estimated the weight of the kite at 3.26 N/spanwise meter. The difference, of 4.44 N/spanwise meter, is the payload capacity of the kite. And, the kite does have a payload. It must be able to hold up the weight of the tether. If the tether material is light enough, we could make the wingspan of the kite great enough to support the tether.

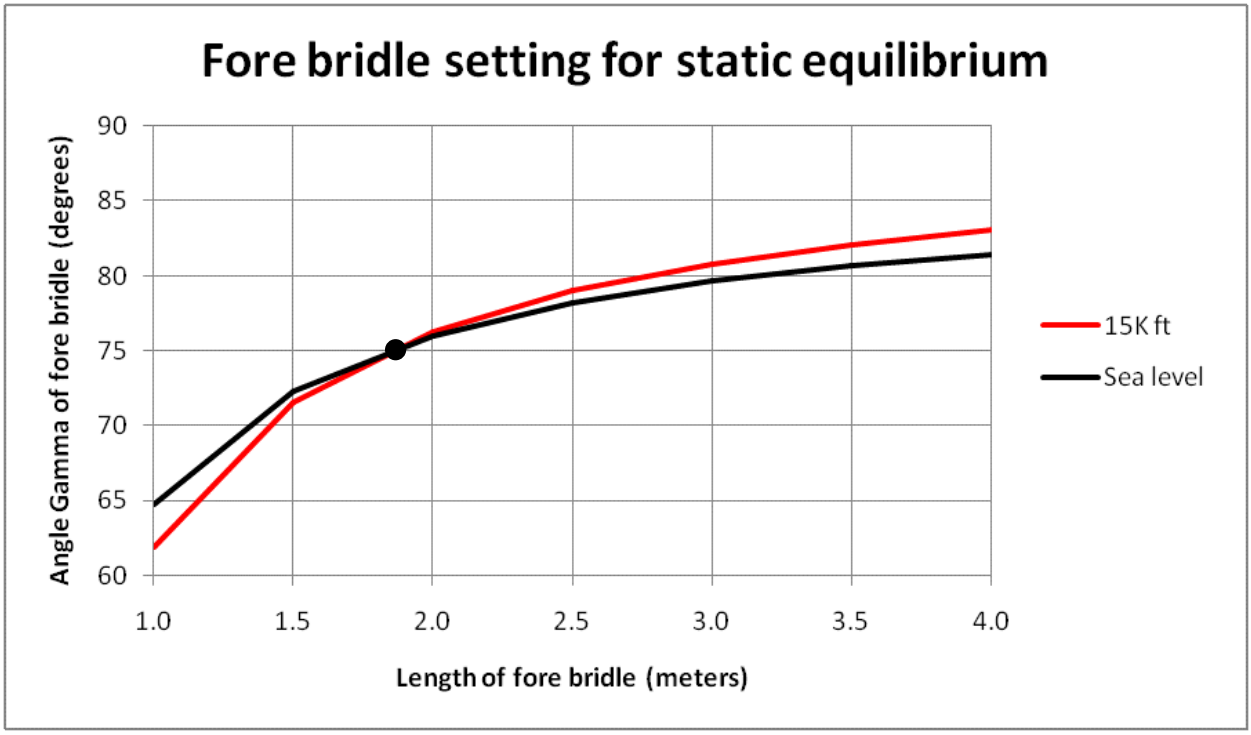
I do not want to digress at this time into a joint analysis of the kite and the tether, which would be needed to estimate the ceiling. We still have more basic things to understand. In particular, I want to see how the setting of the bridle, which we did on the ground in Step #3 above, handles flight conditions at FL150. It

is unlikely that the choice of bridle we made on the ground will still result in a 7° angle of attack at high altitude. Unfortunately, the only OpenFoam simulation we have made so far assumes that 7° angle of attack.

Rather than run more OpenFoam simulations, at other angles of attack, let's try something completely different. Let's determine if there is a bridle setting at 15,000 feet that is in the same range as the bridle settings available at sea level. If there is, then one solution is to set the bridle in such a way that it results in static equilibrium at both altitudes. In essence, I am proposing to repeat the series of calculations done in Step #3, but using the membrane tensions, etc., for 15,000 feet. To assist in the static equilibrium calculations, I wrote a short VisualBasic program which encodes the formulae developed in Step #3. the program is listed in Appendix "C" below. The principal output from the program is a listing of a range of pairs of forward bridle lengths and angles which are consistent with static equilibrium. The output message box for the 15,000 foot case is as follows.



This screenshot shows three bridle configurations which are static equilibria at 15,000 feet. The three shown are ones where the length of the forward bridle is set to one, one-and-one-half and two meters, respectively. These three points, and some other lengths also, are the red line in the following graph. The vertical axis is the angle γ_{fore} which the foward bridle line makes with the reference chord. The black line is the corresponding sets of length-angle pairs for equilibria at sea level. It is important to understand that all the points on the two lines are equilibria at the same 7° angle of attack.



The point of intesection occurs when the the length of the forward bridle is about 1.8 meters. This point is highlighted with a black dot. If the kite is trimmed on the ground to a 7° angle of attack and this particular fore bridle length, then the kite will fly at 15,000 feet at the same angle of attack.

In the next paper, I will look at the interaction between bridle length and angles of attack.

Appendices

Appendix "A" is an analysis of the equations which are used to construct a seed shape from circular arcs.

Appendix "B" describes the components of the kite in enough detail to estimate their weights. It also lays the foundations for the analysis of static equilinrum.

Appendix"C" is a listing of the VisualBasic code for the shape-calculation procedure.

Appendix "D" is a listing of the text file written by the module *WriteGMeshFile.vb* for the base case. GMesh accepts this type of text file and constructs from it a two-dimensional mesh, whigh is extruded into the third dimension by the width of the vitual wind tunnel.

Appendix "E" is a listing of the text file written by the module *WriteOpenFoamFunction.vb* for the base case. The contents of the file are 1003 function declarations in C-format. This text is copied into the appropriate spot in the system/controlDict text file, which OpenFoam uses to control how often to print the total forces and the per-segment forces.

Appendix "F" is a listing of the 11 most important files in the base case OpenFoam directory.

Appendix "G" is a listing of the VisualBasic code of a short program which automates the calculations of a static equilibrium.

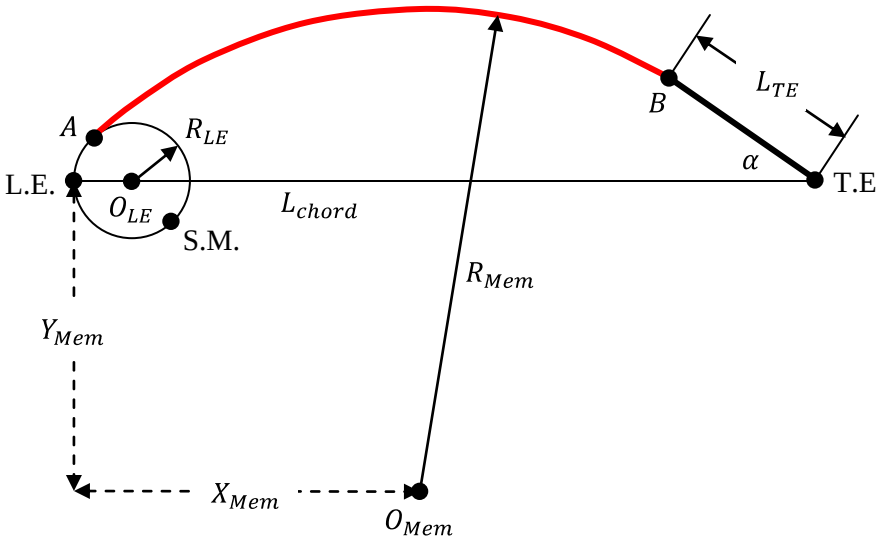
Jim Hawley
August 2014

An e-mail setting out errors or omissions would be appreciated.

Appendix "A"

<u>The equations of the circular arcs used as the seed shape</u>

The following figure defines certain features of the geometry we will use as the seed shape.



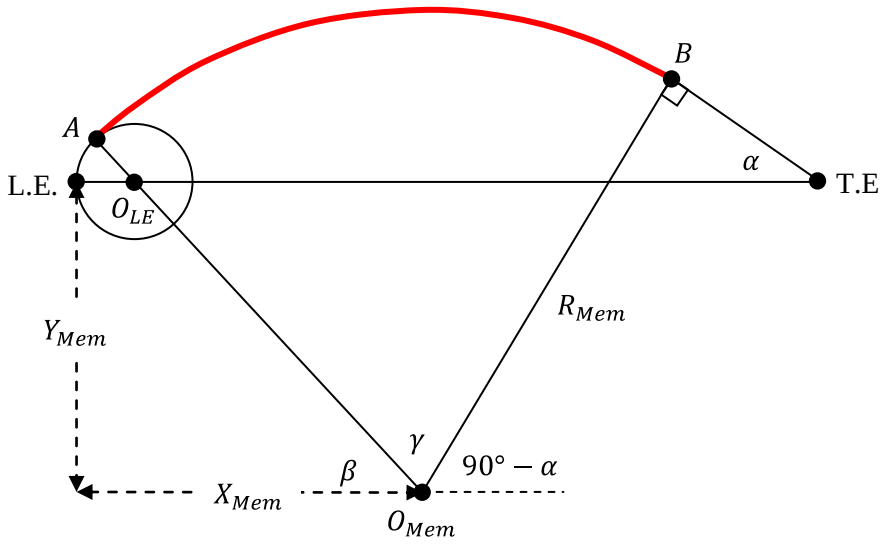
The reference chord, which has length L_{chord}, extends horizontally from the leading edge (L.E.) to the trailing edge (T.E.). The leading edge tube is represented in cross-section by the circle with radius R_{LE}. From the way we defined the leading edge, we know that the center of the leading edge circle (O_{LE}) must be a distance R_{LE} aft of the leading edge. Point A is the departure point, at which the impermeable membrane, which is shown in red, leaves the leading edge circle. The membrane must be tangent to the leading edge circle at this point A. Point S.M. is the "start of the membrane", which is pulled tight against the leading edge circle up to the departure point.

The free-flying part of the membanre, shown in red, is the arc of a circle. That circle has an unknown radius, say R_{Mem}. The center of the membrane's circle (O_{Mem}) is located at some unknown point, which we will say is a distance Y_{Mem} "below" the reference chord and a distance X_{Mem} aft of the leading edge. Resist the temptation to apply the Pythagorean Theorem -- R_{Mem} is not the hypotenuse formed by X_{Mem} and Y_{Mem}.

The aft edge of the impermeable membrane is point B. The membrane is attached to, and will be tangential to, the tailing edge string at that point. We will use the symbol L_{TE} for the length of the trailing edge string and will assume that the trailing edge string meets the reference chord at the angle α shown.

The flexible surface of interest to us is the sum of three surfaces: (i) that part of the circumference of the leading edge circle from point S.M. to the departure point A, (ii) that part of the circumference of the membrane's circle from point A to point B and (iii) the trailing edge string, with its length L_{TE}. The sum of the the first two lengths is known and constant -- it is the width of the nylon sheet. In this Appenidx I will use the S_{nylon} for the width of the nylon sheet.

We are going to define the locations of points A and B by using angles subtended at the center of the membrane's circle. The angles β and γ are defined in the following figure.



The length of the arc on the leading edge circle from the leading edge to point A is equal to:

$$\text{arc}_{LE\to A} = \frac{\beta}{2\pi} \times 2\pi R_{LE} = \beta R_{LE} \quad (A1)$$

Since the start of the membrane, point S.M., is $135° = 3\pi/4$ radians further around the bottom of the leading edge circle from the leading edge, the length of nylon which lies on the surface of the circle is equal to:

$$\text{arc}_{SM\to A} = \frac{\left(\frac{3}{4}\pi + \beta\right)}{2\pi} \times 2\pi R_{LE} = \left(\frac{3}{4}\pi + \beta\right) R_{LE} \quad (A2)$$

The length of the circular arc on the membrane's circle from point A to point B is equal to:

$$\text{arc}_{A\to B} = \frac{\gamma}{2\pi} \times 2\pi R_{Mem} = \gamma R_{Mem} \quad (A3)$$

So, we can write the total length of the nylon as follows:

$$S_{nylon} = \left(\frac{3}{4}\pi + \beta\right) R_{LE} + \gamma R_{Mem} \quad (A4)$$

Consider point A, which is the point where the leading edge circle is tangent to the membrane's circle. This can only be the case if the ray from the membrane's circle's center (O_{Mem}) to point A also passes through the center of the leading edge circle (O_{LE}). Using this fact, we can write down an equation for the vertical distance (vertical meaning in the direction perpendicular to the reference chord) from point A to the center O_{Mem}. We must have:

$$R_{Mem} \sin\beta = R_{LE} \sin\beta + Y_{Mem} \quad (A5)$$

In a similar way, we can write down an equality for the horizontal distance to the center O_{Mem} as:

$$(R_{LE} - R_{LE} \cos\beta) + R_{Mem} \cos\beta = X_{Mem} \quad (A6)$$

We should be able to solve the two equations, Equation $(A5)$ and Equation $(A6)$, for two of the unknowns. Let's solve for X_{Mem} and Y_{Mem}, thus:

$$\left.\begin{aligned} X_{Mem} &= R_{LE}(1-\cos\beta) + R_{Mem}\cos\beta \\ Y_{Mem} &= (R_{Mem} - R_{LE})\sin\beta \end{aligned}\right\} \quad (A7)$$

Philosophically, what we have done is set up the geometry and impose restrictions which ensure that the first curve (the circular arc on the leading edge circle) meets the second curve (the circualr arc on the membrane's circle) at point *A*, and that the two curves have the same slope there. We will now do the same at point B, which is the point of intersection between the second curve and the third curve (the trailing edge string which happens to be a straight line segment).

The membrane's circle is tangent to the trailing edge string at point B, which can be the case if and only if the ray from the membrane's circle's center to point B is perpendicular to the trailing edge string at point B. That imposes some restrictions on the angles at the center of the membrane's circle, in particular, that:

$$\beta + \gamma - \alpha = 90° \quad (A8)$$

The vertical distance equality here can be wrtten as:

$$\begin{aligned} & R_{Mem}\sin(90° - \alpha) = L_{TE}\sin\alpha + Y_{Mem} \\ \rightarrow \quad & R_{Mem}\cos\alpha = L_{TE}\sin\alpha + Y_{Mem} \end{aligned} \quad (A9)$$

and the corresponding horizontal distance equality as:

$$\begin{aligned} & L_{chord} = X_{Mem} + R_{Mem}\cos(90° - \alpha) + L_{TE}\cos\alpha \\ \rightarrow \quad & L_{chord} = X_{Mem} + R_{Mem}\sin\alpha + L_{TE}\cos\alpha \end{aligned} \quad (A10)$$

It is possible to solve these two equations for X_{Mem} and Y_{Mem}, just like we did before:

$$\left.\begin{aligned} X_{Mem} &= L_{chord} - R_{Mem}\sin\alpha - L_{TE}\cos\alpha \\ Y_{Mem} &= R_{Mem}\cos\alpha - L_{TE}\sin\alpha \end{aligned}\right\} \quad (A11)$$

We can equate the left-hand sides of Equations $(A7)$ and Equations $(A11)$ to get:

$$\begin{aligned} X-\text{direction:} \quad & R_{LE}(1-\cos\beta) + R_{Mem}\cos\beta = L_{chord} - R_{Mem}\sin\alpha - L_{TE}\cos\alpha \\ \rightarrow \quad & R_{Mem}(\sin\alpha + \cos\beta) = L_{chord} - L_{TE}\cos\alpha - R_{LE}(1-\cos\beta) \\ \rightarrow \quad & R_{Mem} = \frac{L_{chord} - L_{TE}\cos\alpha - R_{LE}(1-\cos\beta)}{\sin\alpha + \cos\beta} \end{aligned} \quad (A12A)$$

$$\begin{aligned} Y-\text{direction:} \quad & (R_{Mem} - R_{LE})\sin\beta = R_{Mem}\cos\alpha - L_{TE}\sin\alpha \\ \rightarrow \quad & R_{Mem}(\sin\beta - \cos\alpha) = R_{LE}\sin\beta - L_{TE}\sin\alpha \\ \rightarrow \quad & R_{Mem} = \frac{R_{LE}\sin\beta - L_{TE}\sin\alpha}{\sin\beta - \cos\alpha} \end{aligned} \quad (A12B)$$

Equation $(A12A)$ and Equation $(A12B)$ have the same left-hand side, namely, R_{mem}. We could just set the right-hand sides equal. We will do that shortly but, before we do, notice that we have another equation which we have not yet used, Equation $(A4)$, which can also be re-arranged as follows to have R_{mem} on the left-hand side. Making the re-arrangement we get:

$$(B4) \quad R_{Mem} = \frac{S_{FS} - \left(\frac{3}{4}\pi + \beta\right)R_{LE} - L_{TE}}{\gamma}$$

$$\rightarrow \quad R_{Mem} = \frac{S_{FS} - \left(\frac{3}{4}\pi + \beta\right)R_{LE} - L_{TE}}{\alpha - \beta + \frac{1}{2}\pi} \quad (A13)$$

Equations $(A12A)$, $(A12B)$ and $(A13)$ constitute three equations in the three unknowns α, β and R_{Mem}. The three equations are highly non-linear and it is unlikely that a closed-form solution could be found. Since all three equations have the same left-hand side, we could set the right-hand sides equal in two pairs, which would give two equations in the two unknowns α and β. Unfortunately, both equations alone would be even more convoluted than the three equations are separately.

Failing a closed-form solution, let's try to re-organize the three equations in some way that facilitates a numerical solution. Equation $(A12B)$ can be re-arranged to isolate angle β, as follows:

$$(A12B): \quad R_{Mem}(\sin\beta - \cos\alpha) = R_{LE}\sin\beta - L_{TE}\sin\alpha$$

$$\rightarrow \quad \sin\beta\,(R_{Mem} - R_{LE}) = R_{Mem}\cos\alpha - L_{TE}\sin\alpha$$

$$\rightarrow \quad \sin\beta = \frac{R_{Mem}\cos\alpha - L_{TE}\sin\alpha}{R_{Mem} - R_{LE}}$$

$$\rightarrow \quad \beta = \sin^{-1}\left[\frac{R_{Mem}\cos\alpha - L_{TE}\sin\alpha}{R_{Mem} - R_{LE}}\right] \quad (A14)$$

Equation $(A13)$ can be re-arranged to isolate angle α, as follows:

$$(A13): \quad \alpha - \beta + \frac{1}{2}\pi = \frac{S_{FS} - \left(\frac{3}{4}\pi + \beta\right)R_{LE} - L_{TE}}{R_{Mem}}$$

$$\rightarrow \quad \alpha = \beta + \frac{S_{FS} - \left(\frac{3}{4}\pi + \beta\right)R_{LE} - L_{TE}}{R_{Mem}} - \frac{1}{2}\pi$$

$$\rightarrow \quad \alpha = \beta\left(1 - \frac{R_{LE}}{R_{Mem}}\right) + \frac{S_{FS} - L_{TE}}{R_{Mem}} - \frac{\pi}{2}\left(1 + \frac{3R_{LE}}{2R_{Mem}}\right) \quad (A15)$$

Substituting Equation $(A14)$ into Equation $(A15)$ gives:

$$\alpha = \left(1 - \frac{R_{LE}}{R_{Mem}}\right)\sin^{-1}\left[\frac{R_{Mem}\cos\alpha - L_{TE}\sin\alpha}{R_{Mem} - R_{LE}}\right] + \frac{S_{FS} - L_{TE}}{R_{Mem}} - \frac{\pi}{2}\left(1 + \frac{3R_{LE}}{2R_{Mem}}\right) \quad (A16)$$

Our numerical procedure will use Equations $(A16)$ in alternation with the other equations, as follows:

Step #1: Make a guess for R_{Mem}. Something like $0.75 \times L_{chord}$ would be a reasonable start.

Step #2: Solve Equation $(A16)$ numerically, to find the value of α which makes the right-hand side equal to the left-hand side. It is helpful to know that angle α must always be between 0° and 90°.

Step #3: With R_{Mem} and α now known, β can be computed from Equation $(A14)$. Angle β must always be between 0° and 180°.

Step #4: With α and β now known, the right-hand side of Equation $(A12A)$ can be evaluated. This gives a comparative value for R_{Mem}.

Step #5: The difference between the initial guess for R_{Mem} in Step #1 and the comparative value from Step #4 can be used to make a better guess for R_{Mem}.

Step #6: Loop from Step #2 until the guess for R_{Mem} becomes suitably close to the comparative value.

Once α, β and R_{Mem} have all been calculated, we can tidy up the loose ends. The curves need to be discretized, or divided into discrete segments. And, the spatial co-ordinates of the end-points of these segments, which I have elsewhere called the "hinges" between the segments, need to be calculated. Let's define the following symbols:

- N_{nylon} is the total number of segments into which the nylon sheet is to be divided,
- N_{LE} is the number of nylon segments which lie on the surface of the leading edge circle and
- N_{flying} is the number of nylon segments.

All of these numbers are integers. They must satisfy:

$$N_{nylon} = N_{LE} + N_{flying} \quad (A17)$$

The length of each segment (ΔS_{nylon}) is calculated by dividing the total width of the nylon sheet by the total number of nylon segments, as follows:

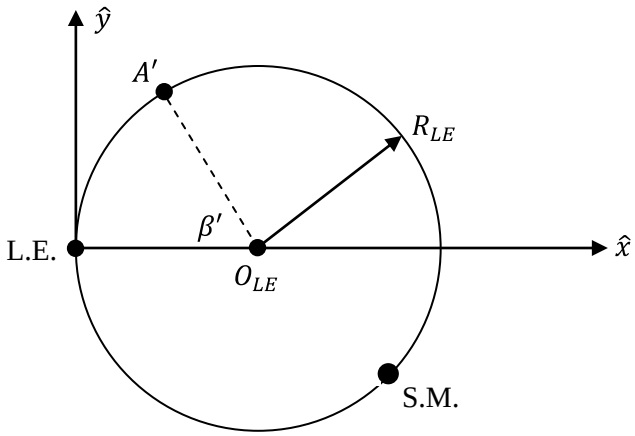
$$\Delta S_{nylon} = \frac{S_{nylon}}{N_{nylon}} \quad (A18)$$

In the general case, the lengths of the first two curves will not be exact integral multiples of the segment length. There are different ways to adjust matters to obtain intergal multiples, two of which are worth considering. One way is to divide, exactly, the length of each of the curves by the number of segments in that curve. That will mean that there are two slightly different segment lengths, neither of which is exactly equal to ΔS_{nylon}. On the other hand, this way does result in points *A* and *B* having exactly the co-ordinates that are determined by the algebra. The second way is to start at the upwind edge of each curve and work towards the downwind edge, dividing each curve into segments equal in length to ΔS_{nylon}. The segmentation will be ended at the point which is as close as possible to the downwind point determined by the algebra. This way results in the desired segment length all along the surface, but causes points *A* and *B* to be located at the nearest hinge co-ordinates, rather than where algebra says they should be.

I have chosen to use the latter approach, and have implemented it as follows.

<u>The co-ordinates of the hinges on the leading edge circle</u>

I will state the co-ordinates of the hinge points in the $\hat{x}$-$\hat{y}$ co-ordinate frame of reference centered precisely at the leading edge, with the $\hat{x}$-axis pointing down the reference chord. The leading edge circle is shown in more detail in the figure shown here.



Ideally, the departure point *A* would lie on the circumference of the leading edge circle, at an angle β measured clockwise from the leading edge. However, since the length of the

circular arc from the start of the membrane (point S.M.) to point A will likely not be an integral multiple of the segment length, it is necessary to introduce some nearby point A' which also lies on the circumference of the leading edge circle but is an integral multiple of segment lengths from point S.M. The polar angle of point A' will be angle β', which will be close to angle β but not exactly equal to it. The number of segments on the leading edge circle can be found by:

$$N_{LE} = \text{Int}\left[\frac{\left(\frac{3}{4}\pi + \beta\right)R_{LE}}{\Delta S_{nylon}}\right] \quad (A19)$$

where the Int() function returns the integer closest to its argument. This means that β' will be equal to:

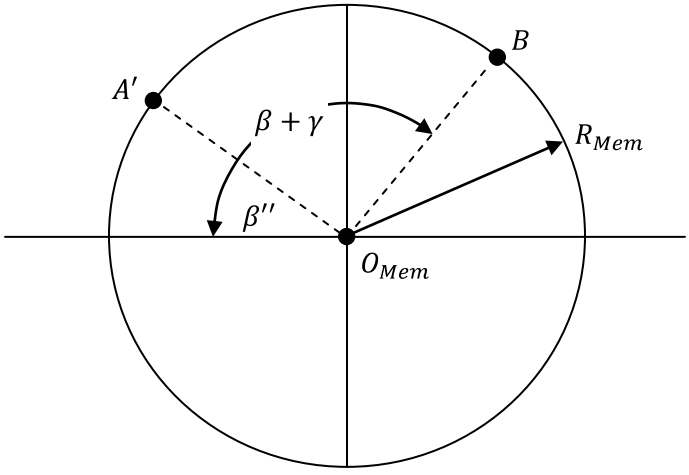
$$\beta' = \frac{N_{LE}\Delta S_{nylon}}{R_{LE}} - \frac{3\pi}{4} \quad (A20)$$

If there are N_{LE} segments along this circular arc, then $N_{LE} + 1$ hinges, or co-ordinate pairs, will be needed to describe all of the end-points. The (x, y) co-ordinates of these hinges are given by:

$$\left.\begin{aligned} x_k &= R_{LE} - R_{LE}\cos\left(k\frac{\Delta S_{nylon}}{R_{LE}}\right) \\ y_k &= R_{LE}\sin\left(k\frac{\Delta S_{nylon}}{R_{LE}}\right) \end{aligned}\right\}, \text{ for } k = 0,1,2\cdots N_{LE} \quad (A21)$$

The co-ordinates of the hinges on the membrane's circle

Point A, at angle β, is the ideal upwind edge of the second curve. However, we have to begin measuring surface lengths along the membrane's circle from point A', not from point A. The membrane's circle is shown in more detail in the following figure. The displacement of point A to point A' changes the subtended angle at the center of the membrane's circle. The subtended angle will no longer be β. Nor will it be β', which is the subtended angle as seen from the center of the leading edge circle. We will use the symbol β'' for the new subtended angle as seen from the center of the membrane's circle.



We can calculate angle β'' using the co-ordinates of point A', which we will know after we have discretized curve #1. If the co-ordinates of point A' are $(x_{A'}, y_{A'})$, then angle β'' is given by:

$$\beta'' = \tan^{-1}\left(\frac{Y_{Mem} + y_{A'}}{X_{Mem} - x_{A'}}\right) \quad (A22)$$

The question now is: how many segments with length ΔS_{nylon} can be placed along the circumference between point A' and point B? This depends on the angle between the two points as seen from the center of the circle O_{Mem}. Because we have adjusted the location of point A, the subtended angle is no longer γ, but is now $\gamma + \beta - \beta''$. The closest integral number of segments which will fit is given by:

$$N_{flying} = \text{Int}\left[\frac{(\gamma + \beta - \beta'')R_{Mem}}{\Delta S_{nylon}}\right]$$

Substituting γ from Equation $(A8)$ allows us to write this as:

$$N_{flying} = \text{Int}\left[\frac{(90° + \alpha - \beta'')R_{Mem}}{\Delta S_{nylon}}\right] \quad (A23)$$

Of course, this will modify slightly the position of point B. In its adjusted position, which will be the nearest hinge point, we will call it point B'. There are N_{flying} segments between points A' and B' but, since the co-ordinates of the hinge at point A' have already been determined, we will only need to add N_{flying} more hinges.

Recall that the co-ordinates of the membrane's circle's center are $(X_{Mem}, -Y_{Mem})$. We can write down the (x_k, y_k) co-ordinates of the new hinges as follows:

$$\left.\begin{aligned} x_j &= X_{Mem} - R_{Mem}\cos\left(\beta'' + k\frac{\Delta S_{nylon}}{R_{Mem}}\right) \\ y_j &= R_{Mem}\sin\left(\beta'' + k\frac{\Delta S_{nylon}}{R_{Mem}}\right) - Y_{Mem} \end{aligned}\right\}, \text{ for } k = 1,2\cdots N_{flying} \quad (A24)$$

<u>The co-ordinates of hinges on the trailing edge string</u>

What remains is the trailing edge string, which now runs from point B' to the trailing edge of the kite section. Point B' is not likely to be exactly the distance L_{TE} from the trailing edge, nor is the distance likely to be an integral multiple of the segment length. And, the angle which point B' makes with respect to the reference chord will no longer be exactly equal to α.

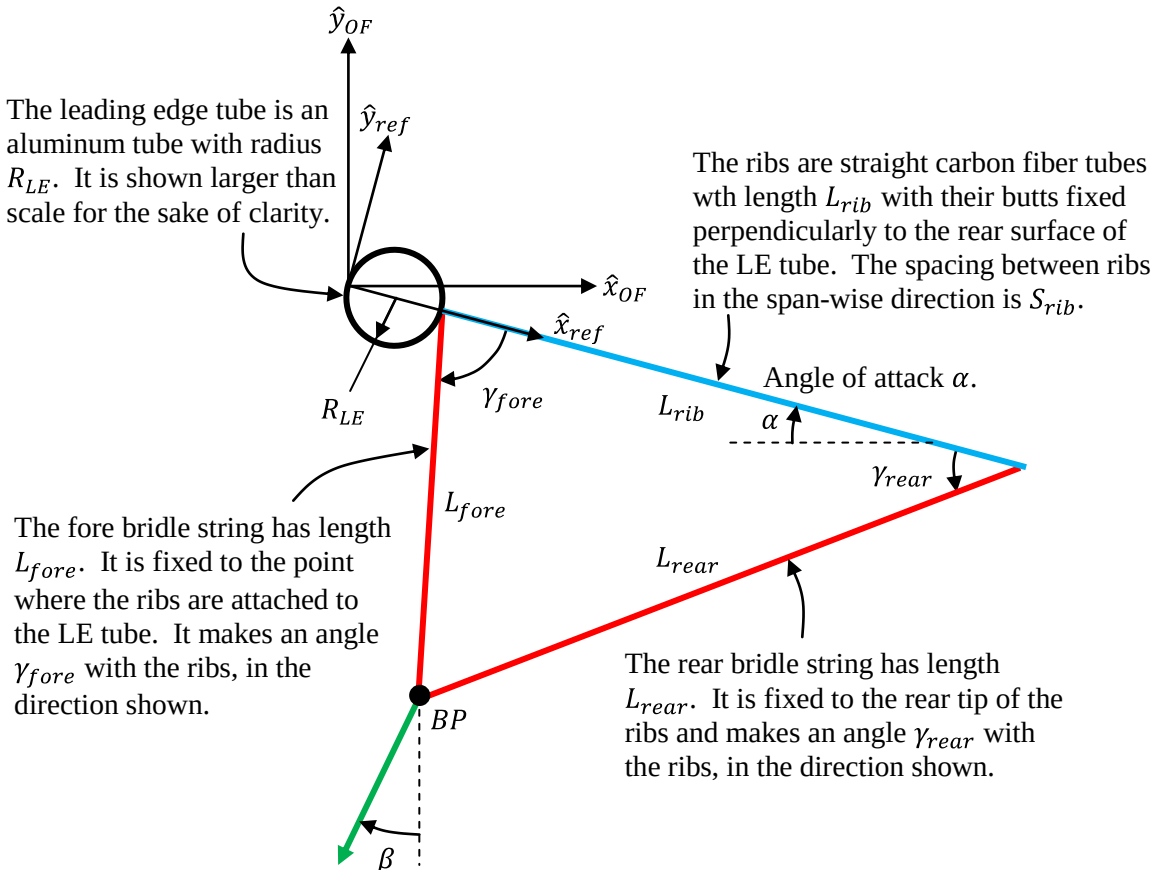
Fortunately, it is not necessary for us to discretize the trailing edge string. We have assumed that no aerodynamic forces are exerted on these segments. The OpenFoam simulaton is carried out ignoring this string, so it is not even defined in the physical model used by OpenFoam. is line. We can treat the trailing each string as a single line segment. We know the co-ordinates of the upwind end, which is point B', and the co-ordinates of the downwind end, which is the aft end of the reference chord. We do not need anything more.

Appendix "B"

Equations of static equilibrum in flght

This Appendix will develop the equations of static equilibrium for the kite in steady flight. It will not delve into questions of stability. To the extent possible, the symbols used here match the variables used in the computer code which automates the calculations.

The support structure and basic geometry



The leading edge tube is an aluminum tube with radius R_{LE}. It is shown larger than scale for the sake of clarity.

The ribs are straight carbon fiber tubes wth length L_{rib} with their butts fixed perpendicularly to the rear surface of the LE tube. The spacing between ribs in the span-wise direction is S_{rib}.

Angle of attack α.

The fore bridle string has length L_{fore}. It is fixed to the point where the ribs are attached to the LE tube. It makes an angle γ_{fore} with the ribs, in the direction shown.

The rear bridle string has length L_{rear}. It is fixed to the rear tip of the ribs and makes an angle γ_{rear} with the ribs, in the direction shown.

The tether, shown in green, is the line which extends from the bridle point (BP) to the ground. It makes an angle β with the vertical, in the direction shown, where it is attached to the bridle point.

I will use two co-ordinate frames of reference. The $\hat{x}_{ref}$-$\hat{y}_{ref}$ co-ordinate system is fixed to the reference chord, and the $\hat{x}_{ref}$ axis is coincident with the ribs. The $\hat{x}_{OF}$-$\hat{y}_{OF}$ co-ordinate system is the one used by OpenFoam, wth the $\hat{x}_{OF}$ axis parallel to the direction of the wind and the $\hat{y}_{OF}$ axis vertical to the surface of the Earth. The two co-ordinate systems have the same origin, being the point on the front side of the leading edge tube directly opposite to the attachment point of the ribs.

Measurements stated in one of the frames of reference can easily be re-stated into the other frame of reference by means of rotation through the angle of attack α, as follows. Such measurements can be position co-ordinates or vector components.

$$\left.\begin{aligned} x_{ref} &= x_{OF}\cos\alpha - y_{OF}\sin\alpha \\ y_{ref} &= x_{OF}\sin\alpha + y_{OF}\cos\alpha \end{aligned}\right\} \quad (B1A)$$

OR

$$\left.\begin{aligned} x_{OF} &= x_{ref}\cos\alpha + y_{ref}\sin\alpha \\ y_{OF} &= -x_{ref}\sin\alpha + y_{ref}\cos\alpha \end{aligned}\right\} \quad (B1B)$$

The force of gravity acting on the leading edge tube

In addition to the radius R_{LE} of the leading edge tube, we will need to know the following parameters to calculate the force of gravity F_{gLE} on the tube:

- t_{LE} is the thickness of the leading edge tube,
- ρ_{LE} is the mass denisty of the material from which the tube is made and
- g is the gravitational acceleration, which we will assume does not diminish with altitude.

The force of gravity acting on the leading edge tube is:

$$\vec{F}_{gLE} = -\rho_{LE}\pi[R_{LE}^2 - (R_{LE} - t_{LE})^2]g\hat{y}_{OF} \quad (B2)$$

πR_{LE}^2 is the cross-sectional area of the leading edge tube, or the area of the circle representing the tube when it is viewed from the end. The similar term $\pi(R_{LE} - t_{LE})^2$ is the cross-sectonal area of the inside of the tube, which is to say, the air inside the tube. The difference between these two areas is the cross-sectional area which is filled with material, in our case, aluminum. This area, when multiplied by the density ρ_{LE}, is the mass of a one-meter length of the leading edge tube (assuming all the calculations are carried out in S.I. units). This mass, when mulitplied by the gravitational acceleration g, is the force of gravity acting on a one-meter length of the tube. From the definition of the two co-ordinates frames of reference, we know that the "downwards" direction, in which gravity pulls, is the negative $\hat{y}_{OF}$ axis. I have taken the liberty of expressing the force of gravity per span-wise meter as a vector, by mulitplying its magnitude by the direction $-\hat{y}_{OF}$.

We may as well be accurate about where this force acts. Because the tube is symmetrical, we can add up the gravitational forces which act on all the separate little bits of volume which make up the tube and represent them collectively by a single point force acting at the center-of-gravity, which in this case is the central axis of the tube. I will use the symbol POA_{gLE} for the point-of-action of gravitational force on the leading edge tube. This point is most easily described as a vector in the frame of reference fixed to the reference chord, as follows:

$$\overrightarrow{POA}_{gLE} = R_{LE}\hat{x}_{ref} \quad (B3)$$

The fact that this force is expressed in one frame of reference while its point of action is expressed in the other does not present any problem. We can use Equations $(A1)$ to transform either vector into the othe frame of reference.

Note that the force of gravity has been calculated as so much force per meter of span. A significant assumption we made before carrying out the OpenFoam simulations was that the airflow over the airfoil can be modelled in two dimensions. Only a wing with an infinite span has the property that the airflow is the same at each span-wise location. The actual airflow might be pretty close to the predicted airflow at the mid-span of a real wing, but the airflow does become increasingly subject to end-effects as one gets

closer and closer to a wing tip. As a first approximation, we will assume that the kite's span is large enough compared to the chord that it makes sense to talk about quantities on a "per meter of span" basis.

The force of gravity acting on the ribs

The ribs were not included in the model of the kite which was simulated by OpenFoam. The implict assumption which justifies excluding the ribs is that they are placed so far apart, and/or that they are so narrow, that they have negligible impact on the average pattern of the airflow. In reality, the ribs are neither small nor far apart. I envision placing ribs every two feet along the span. I also envision (since I already have some of them on hand) that the ribs will be carbon fiber tubes 6mm in diameter and one meter long. These ribs weigh 14 grams each. The additional parameters we need to know to calculate the force of gravity on the ribs are the following:

- L_{rib} is the length of each rib which, in our base case, will be one meter,
- m_{rib} is the mass of each rib which, in our case, will be 14 grams and
- S_{rib} is the spacing between ribs. In our case, the spacing will be two feet, or 0.6096 meters.

The force of gravity acting on the ribs, per meter of span, is:

$$\vec{F}_{gRib} = -\left(\frac{m_{rib}}{S_{rib}}\right) g \hat{y}_{OF} \quad (B4)$$

Dividing the mass of each rib m_{rib} by the spacing S_{rib} gives the equivalent mass per meter of span. Multiplying this mass by the gravitational acceleration g gives the force of gravity per meter of span. As before, I have been specific about the direction of this force.

Since the carbon tubes are uniform in construction along their length, the centers of gravity of the ribs will be their geometric midpoints. We can write the points-of-action for the force of gravity acting on the ribs as follows:

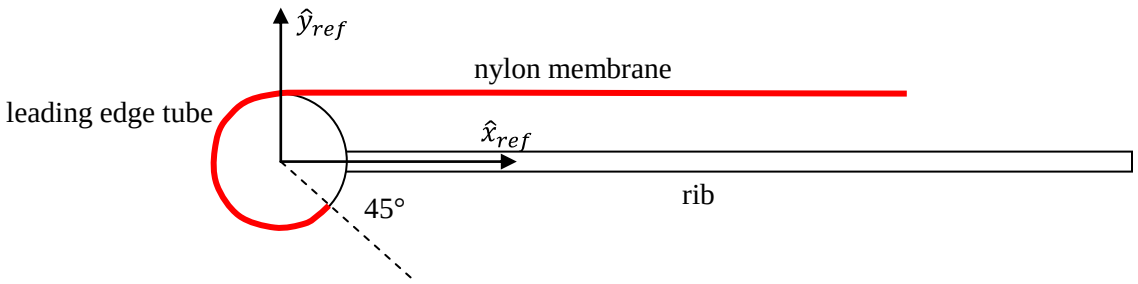
$$\overrightarrow{POA}_{gRib} = \left(2R_{LE} + \tfrac{1}{2}L_{rib}\right)\hat{x}_{ref} \quad (B5)$$

Bear in mind that the forward end of each one-meter long rib is attached to the outside rear of the leading edge tube, so the full diameter of the leading edge tube needs to be added when calculating the location of the midpoint.

The force of gravity acting on the nylon membrane

The OpenFoam simulation assumed that the chord-wise surface width of the nylon membrane was 98 centimeters. This length was derived from an unhemmed sheet one meter wide in which one-centimeter hems were sewn into both fore and aft edges.

Obviously, the shape of the membrane changes as flight conditions change, so the exact spatial distribution of the nylon, and therefore its weight, cannot be calculated until the shape is finalized. Waiting is not convenient nor is it necessary for these rough preliminary calculations. It is more expedient to make a simplifying assumption which will allow us to treat the nylon's weight distribution as being known *a priori*. I think the following assumed shape will be good enough for our purposes.



Recall that the physical model of the kite section used by OpenFoam has the front edge of the nylon membrane tucked underneath the leading edge tube, and attached to the tube along a line just below the rib-to-tube joints. More precisely, we said that the attachment line was located 135° around the bottom of the leading edge tube from the nose. For weight calculations, we will assume that the nylon is wrapped over the top of the leading edge tube and pulled straight towards the rear, where "straight" in this context means parallel to the ribs.

We can assume that the nylon membrane has a uniform and constant mass per unit surface area. A heavy ripstop nylon suitable for a large kite will have a mass density ρ_{nylon} in the order of 70 grams per square meter of surface area. Calculating the gravitational force (i.e., weight) per span-wise meter of nylon is easy -- it is simply 70 grams mulitplied by the gravitational acceleration.

Calculating the location of the center of gravity is more tricky. One could do a detailed geometrical study of the shape shown in the figure above. One could integrate along the known shape of the cross-section to calculate exact location of the center-of-gravity. Instead, I will do a rough approximation. Then I will explain why a more detailed study of this geomtery is pointless. The rough approximation is this.

1. Near the front, a length of membrane equal to the circumference of the leading edge tube ($2\pi R_{LE}$) is assumed to be wrapped completely around the tube. This length of nylon has a mass of $\rho_{nylon} 2\pi R_{LE}$ per meter of span, and experiences a force of gravity equal to $\rho_{nylon} 2\pi R_{LE} g$ per meter of span. We will assume that the point-of-action of this force is the center of the leading edge tube.
2. We also know the length of the remainder of the nylon membrane. It is the width of the orginal nylon sheet W_{nylon} less the length $2\pi R_{LE}$ already accounted for around the leading edge tube. (The impact of hemming is ignored.) This length has a mass equal to $\rho_{nylon}\left(W_{nylon} - 2\pi R_{LE}\right)$ per meter of span and experiences a force of gravity of $\rho_{nylon}\left(W_{nylon} - 2\pi R_{LE}\right) g$ per meter of span. We will assume that the point-of-action of this force is the midpoint of the rib. Note that the midpoint of the rib is not the same as the midpoint of the horizontal part of the nylon. The trailing edge strings tend to bias the membrane forward with repect to the midpoint.

The force of gravity acting on the two parts of the nylon membrane can be written as follows:

$$\left.\begin{aligned} \vec{F}_{gNylon1} &= -\rho_{nylon} 2\pi R_{LE} g \hat{y}_{OF} \\ \vec{F}_{gNylon2} &= -\rho_{nylon}\left(W_{nylon} - 2\pi R_{LE}\right) g \hat{y}_{OF} \end{aligned}\right\} \quad (B6)$$

with the corresponding points-of-action being:

$$\left.\begin{aligned} \overrightarrow{POA}_{gNylon1} &= R_{LE}\hat{x}_{ref} \\ \overrightarrow{POA}_{gNylon2} &= \left(2R_{LE} + \tfrac{1}{2}L_{rib}\right)\hat{x}_{ref} \end{aligned}\right\} \quad (B7)$$

Now let me explain why a detailed study might not give any better answer than the rough approximation. The membrane -- or at least the free-flying part of the membrane -- does not "rest" on the kite's structure. It is supported directly by the airflow, not by the leading edge tube or the ribs. The correct treatment of the weight of the free-flying segments of the membrane would be to include the gravitational force acting on each segment during the march along the membrane which computes the shape. This can be done. There is no restriction on the types of forces that can be taken into account during the marching process.

However, just because the airflow supports the free-flying membrane does not mean that the kite's structure does not feel the effects. One consequence of accounting for the force of gravity on the segments durng the marching process will be that the inter-segment tensions will be changed (reduced). The tensions which the free-flying part of the membrane exerts on its front and rear support points will be changed (also reduced). These reductions will reduce the net upward lift which the membrane exerts on the structure, as compared to the no-gravity case. This reduction in the net upward lift is logically identical to the straight-forward addition of gravitational weight. In fact, when the components of the kite are analyzed collectively as a rigid body, the net upward lift calculated by both methods should be exactly the same. The only difference (more study is needed) may be the ratio of the tensions at the departure point and on the trailing edge strings. A change in this ratio would affect the overall mechanical moment experienced by the kite, but it would not change the net vertical or horizontal forces.

The weight of miscellaneous fixtures

Some additional hardware is going to be needed. The front edge of the nylon membrane must be attached to the leading edge tube. This could a some kind of track, a system of hooks and grommets or perhaps even Duct tape. Eyebolts or D-rings are going to be needed to attach the fore bridle lines to the ribs and/or the leading edge tube. The trailing edge has a similar requirement. The trailing edge strings will need to be tied off at the rear end-tips of the ribs. If a boltrope is used to strengthen the rear edge of the nylon membrane, the some means will be needed to secure it to the end-tips. Better yet would be a bolt-and-nut mechanism at the end-tips which allows the membrane tension to be adjusted in pre-flight. More rings are going to be needed to attach the rear bridle lines to the end-tips.

We wll try to keep each bit of hardware as light as possible. Whatever hardware is needed wll have to be repeated at each rib, so the totals will mount up. At this moment all I can do is guess or, as the alttrnative rationale, to set target weights. In the base case, I will allow for one ounce ($m_{foreHW} = 30$ grams) at the front of each rib and one-half ounce ($m_{rearHW} = 15$ grams) at the rear tip of each rib. These masses must be scaled up by the factor $1/S_{rib}$ to represent masses per meter of span. I will write the weight of these pieces of harware ("HW") as follows:

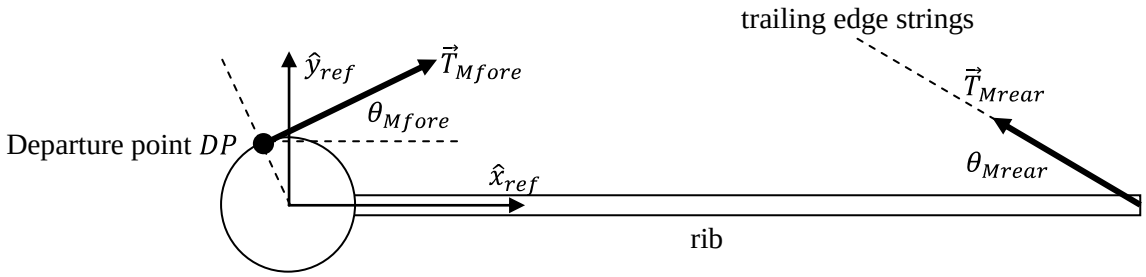
$$\left.\begin{aligned} \vec{F}_{gforeHW} &= -\left(\frac{m_{foreHW}}{S_{rib}}\right) g\hat{y}_{OF} \\ \vec{F}_{grearHW} &= -\left(\frac{m_{rearHW}}{S_{rib}}\right) g\hat{y}_{OF} \end{aligned}\right\} \quad (B8)$$

with the corresponding points-of-action being:

$$\left.\begin{aligned} \overrightarrow{POA}_{gforeHW} &= 2R_{LE}\hat{x}_{ref} \\ \overrightarrow{POA}_{grearHW} &= (2R_{LE} + L_{rib})\hat{x}_{ref} \end{aligned}\right\} \quad (B9)$$

The aerodynamic forces on the membrane

The ultimate product of the OpenFoam simulation, supported by the shape-calculation procedure, are the tension forces which the free-flying part of the membrane exerts on the departure point (at the front end) and on the trailing edge strings (at the rear end). The following figure shows how we will represent these forces in the weight-and-balance analysis which is the subject of this appendix. The directions of the tension forces $\vec{T}_{Mfore}$ and $\vec{T}_{Mrear}$ are given by the angles θ_{Mfore} and θ_{Mrear}, respectvely, which are algebraically positive in the directions shown.

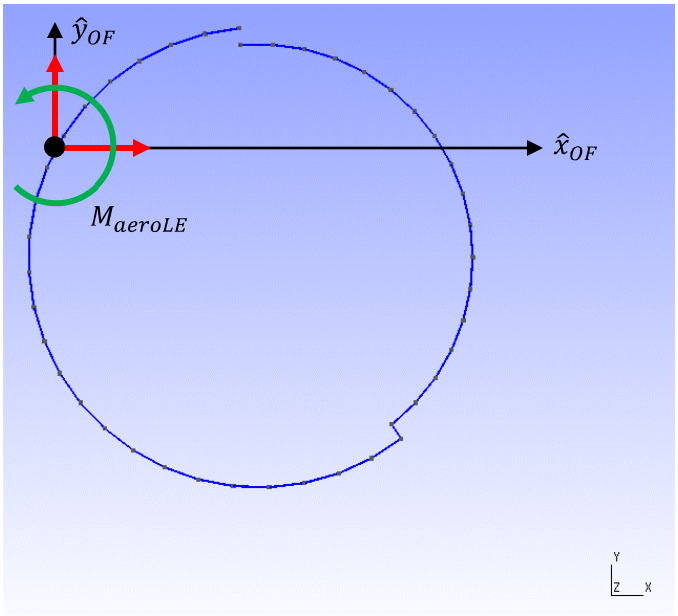


The shape-calculation procedure computes the location of the departure point assuming that the membrane is divided into a large number of segments, say, 500. A certain number of those segments lie on the leading edge tube; the remainder are said to be free-flying. The number of segments lying on the leading edge tube determines the percentage of the tube's circumference which is covered, and thus the location of the departure point. For our purposes, here, it is not necessary to refer back to the segmentation of the membrane to find out where the departure point is located. Instead we can simply treat the departure point as the point at which the forward tension force is tangent to the leading edge tube. In the frame of reference of the reference chord, then, the points-of-action of the two tension forces are as follows:

$$\left.\begin{aligned}\overrightarrow{POA}_{Mfore} &= \left(R_{LE} - R_{LE}\cos\theta_{Mfore}\right)\hat{x}_{ref} + \left(R_{LE}\sin\theta_{Mfore}\right)\hat{y}_{ref}\\ \overrightarrow{POA}_{Mrear} &= (2R_{LE} + L_{rib})\hat{x}_{ref}\end{aligned}\right\} \quad (B10)$$

The aerodynamic forces on the leading edge tube

Our primary interest in this study is, of course, the flying part of the membrane. However, the leading edge tube is both an important part of the physical structure and the most significant source of drag, so it cannot be ignored. Fortunately, the model used in the OpenFoam simulation includes the outline of the leading edge tube, so OpenFoam calculates the forces for us directly. The figure shown here is taken from the GMesh program whch creates the mesh within which OpenFoam carries out its numerical work. The figure shows the outline (the thin blue line) of the leading edge tube OpenFoam uses. The outline is made up of straight line segments. Small blacks dots identfy the hinges between the segments.



The heavy black dot is the origin of the $\hat{x}_{OF}$-$\hat{y}_{OF}$ co-ordinate frame of reference. As previously described this origin is the projection of the ribs through the leading edge tube. It appears off-center because the kite has been rotated clockwise by the angle of attack. Note that the orign does not coincide wth one of the hinges between the segments. The segments were constructed so that there would be an integral number of them between the start of the membrane point (the abrupt change in radius at the lower right) and the departure point (where the outine stops near the top). There is no requirement that the leading edge conincide with a hinge.

The two components of the aerodynamic force on the leading edge tube are shown by the red arrows. I will use the symbol $\vec{F}_{aeroLE}$ for this force, so its two components will be $F_{aeroLEx}$ and $F_{aeroLEy}$ stated in OpenFoam's frame of reference.

The aerodynamic effects include a mechanical moment whose magnitude M_{aeroLE} I have identified in the figure. While the tube is nominally a circle, it is not exactly a circle. The thickness of the membrane only obtains over part's of the tube's circumference. Furthermore, the airflow is not at all symmetrical around the tube. Together, these factors cause the airflow to exert a moment on the tube, tending to rotate it. The convention for the directon of the moment -- counter-clockwise -- is shown by the circular green arrow in the figure. The direction of the axis of rotation is used as the mathematical drection of the moment. In this case, the axis is parallel to the positive $\hat{z}_{OF}$ axis, which points out of the page.

The three magnitudes, $F_{aeroLEx}$, $F_{aeroLEy}$ and M_{aeroLE}, can be read directly from the log text fle of the OpenFoam runs. Since the virtual wind tunnel used by OpenFoam is only one millimeter thick, the figures written by OpenFoam need to be multiplied by 1,000 so that they represent the forces and moment per one meter of span.

<u>The force exerted by the tether</u>

The tether extends from the winch on the ground up to the bridle point, where all of the bridle lines are collected at one point. The tether will be assumed to be a perfectly flexible line, so the tension force it carries is coincident with the longitudinal axis of the tether. I will use the symbol $\vec{F}_{Tether}$ for the tension force in the tether at the bridle point. The figures above define the angle β as the angle with respect to the vertical, in the upwind drection, which the tether makes at the bridle point. We can write:

$$\vec{F}_{Tether} = -F_{Tether} \sin\beta\, \hat{x}_{OF} - F_{Tether} \cos\beta\, \hat{y}_{OF} \quad (B11)$$

The location of the bridle point, which is the point-of-action of the tether's tension, can be found by trigonometry, and is:

$$\overrightarrow{POA}_{Tether} = \left(2L_{RE} + L_{fore} \cos\gamma_{fore}\right)\hat{x}_{ref} - L_{fore} \sin\gamma_{fore}\, \hat{y}_{ref} \quad (B12)$$

In this expression, L_{fore} is the length of the forward bridle line and γ_{fore} is the angle which the forward bridle line makes with the ribs, in the direction defined in the figures above.

<u>The internal forces</u>

There are two other sets of forces which we will want to know, but which we do not have to account for explicitly at this time.

When the kite is in stable flight, the ribs will be in compression. The ends of each rib are drawn together by the tensions exerted by the membrane and the tensions exerted by the bridle lines. The ribs may also be subject to a mechanical moment. In due course, we will want to calculate the magnitude of this compressive force. This is the "load" which the ribs must withstand and determines how strong the ribs must be structurally.

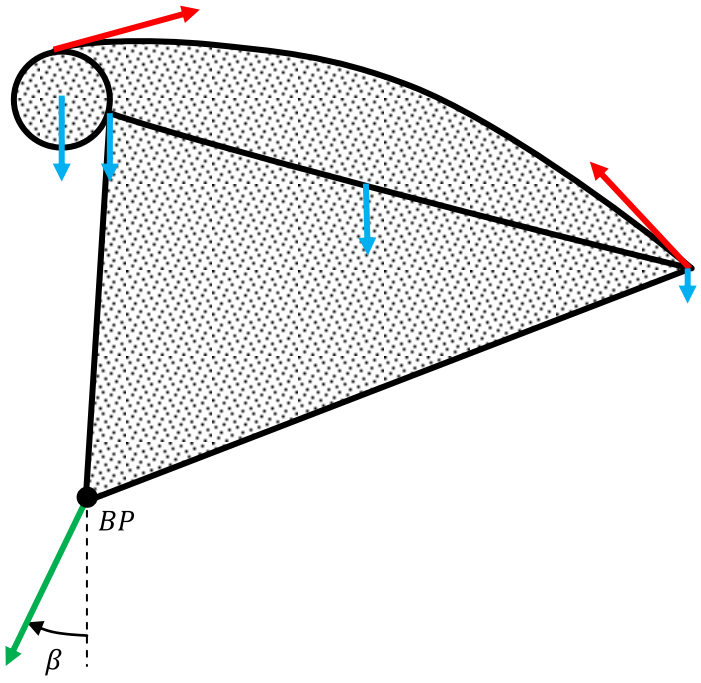
But, we do not need to calculate the compression force just yet. For the weight-and-balance analysis, we will treat the entire kite, including the ribs, as a single rigid body. The kite's equilibrium is determined by its response to external forces. Whatever is happening inside the ribs is an internal matter and does not affect the kite's overall behaviour.

We are alao going to want to know the tension forces in the fore and rear bridle lines. The ratio of the tensions in these two lines is an important ingredient in the stability of the kite. If we assume the bridle lines are perfect strings (the usual case), then they can sustain only tension. They cannot be compressed. Should the tension in one of the bridle lines be reduced to zero, control over the kite will be lost. Indeed, any flight condition which causes the tension in either of the bridle lines to get too small is a dangerous flight condition, nearing the point where it flies out of control.

Like the ribs, though, we do not need to calculate the tension forces in the bridle lines at this time. If the kite is stable, so that both bridle lines are taut, then they can also be treated as part of a single rigid body, upon which a test of equilibrium can be applied. (As a subsequent exercise, the tensions in both bridle lines must be calculated, to ensure that they are indeed taut.)

Static equilibrium of the kite and bridle

Let's begin by treating the entire kite -- structural framework, flexible membrane and bridle -- as a rigid body. So long as none of the component parts is moving relative to any of the other component parts, it does not matter how the parts are attached to one another. The only thing that matters is whether the configuration as a whole is in balance with the external forces. The following figure shows the object I am considering to be rigid. Everything filled with a dotted hatch is part of the rigid body. The external forces consist of: (i) the tension forces exerted by the extremities of nylon membrane (shown in red), (ii) the gravitational forces acting at four different locations (shown in blue) and (iii) the tension force in the tether (shown in green) which keeps the whole thing from blowing away. For the sake of clarity, I have not repeated the dimensions or the names of the forces.
For ready reference, I repeat in the following table the ten forces quantified above and, in a separate table after that, their points-of-action.



Force / moment	Expression for force / moment

Gravity on LE tube	$\vec{F}_{gLE} = -\rho_{LE}\pi[R_{LE}^2 - (R_{LE} - t_{LE})^2]g\hat{y}_{OF}$
Gravity on ribs	$\vec{F}_{gRib} = -\left(\frac{m_{rib}}{S_{rib}}\right)g\hat{y}_{OF}$
Gravity on nylon - Part I	$\vec{F}_{gNylon1} = -\rho_{nylon}2\pi R_{LE}g\hat{y}_{OF}$
Gravity on nylon - Part II	$\vec{F}_{gNylon2} = -\rho_{nylon}(W_{nylon} - 2\pi R_{LE})g\hat{y}_{OF}$
Gravity on front end hardware	$\vec{F}_{gforeHW} = -\left(\frac{m_{foreHW}}{S_{rib}}\right)g\hat{y}_{OF}$
Gravity on rear end hardware	$\vec{F}_{grearHW} = -\left(\frac{m_{rearHW}}{S_{rib}}\right)g\hat{y}_{OF}$
Membrane tension on front end	$\vec{T}_{Mfore} = T_{Mfore}\cos\theta_{Mfore}\,\hat{x}_{ref} + T_{Mfore}\sin\theta_{Mfore}\,\hat{y}_{ref}$
Membrane tension on rear end	$\vec{T}_{Mrear} = -T_{Mrear}\cos\theta_{Mrear}\,\hat{x}_{ref} + T_{Mrear}\sin\theta_{Mrear}\,\hat{y}_{ref}$
Aerodynamic forces on LE tube	$\vec{F}_{aeroLE} = F_{aeroLEx}\hat{x}_{OF} + F_{aeroLEy}\hat{y}_{OF}$
Aerodynamic moment on LE tube	$\vec{M}_{aeroLE} = M_{aeroLEx}\hat{z}_{OF}$
Tension in tether	$\vec{F}_{Tether} = -F_{Tether}\sin\beta\,\hat{x}_{OF} - F_{Tether}\cos\beta\,\hat{y}_{OF}$

Force / moment	**Expression for point-of-action**
Gravity on LE tube	$\overrightarrow{POA}_{gLE} = R_{LE}\hat{x}_{ref}$
Gravity on ribs	$\overrightarrow{POA}_{gRib} = \left(2R_{LE} + \frac{1}{2}L_{rib}\right)\hat{x}_{ref}$
Gravity on nylon - Part I	$\overrightarrow{POA}_{gNylon1} = R_{LE}\hat{x}_{ref}$
Gravity on nylon - Part II	$\overrightarrow{POA}_{gNylon2} = \left(2R_{LE} + \frac{1}{2}L_{rib}\right)\hat{x}_{ref}$
Gravity on front end hardware	$\overrightarrow{POA}_{gforeHW} = 2R_{LE}\hat{x}_{ref}$
Gravity on rear end hardware	$\overrightarrow{POA}_{grearHW} = (2R_{LE} + L_{rib})\hat{x}_{ref}$
Membrane tension on front end	$\overrightarrow{POA}_{Mfore} = (R_{LE} - R_{LE}\cos\theta_{Mfore})\hat{x}_{ref} + (R_{LE}\sin\theta_{Mfore})\hat{y}_{ref}$
Membrane tension on rear end	$\overrightarrow{POA}_{Mrear} = (2R_{LE} + L_{rib})\hat{x}_{ref}$
Aerodynamic forces on LE tube	$\vec{0}$
Aerodynamic moment on LE tube	$\vec{0}$
Tension in tether	$\overrightarrow{POA}_{Tether} = (2L_{RE} + L_{fore}\cos\gamma_{fore})\hat{x}_{ref} - L_{fore}\sin\gamma_{fore}\,\hat{y}_{ref}$

If the kite is in a static equilibrium, the sum of all external forces acting on it must be zero. Why? Suppose that, after adding up all the external forces, the net force was not zero. In accordance with Newton's Law -- that force equals mass multiplied by acceleration -- the kite would start to accelerate in the direction of the net force. Acceleration causes changes in speed which in turn cause changes in position -- completely contrary to the defnition of an equilibrium. No there cannot be any net force.

Since we are examining the kite in a two-dimensional plane, it is customary to be more specific and to say that there cannot be any net force in a given direction and that there cannot be any net force in the drection perpendicular to that given direction. It is almost always the case that a set of perpendicular axes have already been established in the two-dimensional plane, so it is almost always convenient to use those two axes as the two directions along which to test the sums of the forces. That is the case here -- we already have a set of perpendicular axes we can use for this purpose. Indded we are blessed. We have

two sets of perpendicular axes, the $\hat{x}_{OF}$-$\hat{y}_{OF}$ set and the $\hat{x}_{ref}$-$\hat{y}_{ref}$ set. We can use either one. If the net force is zero along both axes of one set of axes, it will be zero along both axes of any other set of axes.

In order to add up the forces on an "apples-to-apples" basis, we do have to express them all in the same co-ordinate frame of reference. As I pointed out above, we can use the rotation equations in Equation (1) for this purpose. Actually, it is not clear to me which co-ordinate frame of reference is better. "Better" in this context means the frame of reference in which the algebra is easiest.

Newon's Law has a rotational counterpart, that torque is equal to the moment of inertia multiplied by the rotational acceleration. This is the basis for the second condition of static equilibrium, that there cannot be any net moment exerted on the kite. (Aside: "Torque" and "moment" are the same thing, the tendency of something to rotate when subject to an off-center force or a pair of non-colinear forces. It is more precise but by no means mandatory, to use the word torque in connection with objects which do start to rotate and to use the word moment in connection with objects which are prevented from moving.) As with the forces, the thought experiment is this. If there was a non-zero net moment, the kite would be subjected to a rotational tendency, and it would start to rotate -- taking it out of the equilibrium state.

In a two-dimesional analysis like this one, all rotation takes place in the plane. Objects rotate either clockwise or counter-clockwise, so the axis of rotation is either out-of-the-page (parallel to the positive $\hat{z}$-axis) or into-the-page (parallel to the negative $\hat{z}$-axis).

In total, then, we will have three equations which describe the static equilibrium. There will be equations for the two components of the force and a third equation for the moment.

A review of the expressions for the forces and moment in the first table above will show that we know the values of all variables except for four. The four unknown variables, or simply "unknowns", are the following:

- the magniude of the tension force F_{tether} in the tether,
- the angle β which the tether makes with the vertical at the bridle point,
- the length L_{fore} of the forward bridle line and
- the angle γ_{fore} which the forward bridle line makes with the reference chord.

All four of these unknowns have to do with the way the kite is attached to the tether. That is entirely consistent with our experience and intuition about how we go about trimming a kite before a flight. The prevailing wind speed (and stiffness of the membrane) causes the kite to fly in a particular attitude. We can adjust the attitude (angle of attack) by adjusting the point at which the tether is tied to the bridle.

Our set of three equations is such that we will be able to solve them for three unknowns. But not four. In order to calculate a complete numerical solution we are going to have to add one more bit of information. Here's how I intend to proceed.

I am going to solve the equations in such an order that the tag-end unknown is the angle γ_{fore} of the forward bridle line. I am then going to complete the solution for a range of different angles. (Philosophically, I am going to add the fourth bit of information simply by setting the fourth value.) In any event, the result of these calculations will be set of the equilibrium points which are possible at the the different bridle angles chosen.

I have chosen to move towards the solution in this way for two reasons.

Firstly, setting the bridle so that the kite is in static equilibrium on the ground is only the beginning. Once the kite is sent aloft, we want it to fly higher and higher, and we want the kite to continue to be in equilibrium at the higher altitudes. We will repeat the equilibrium calculations at higher altitudes. We wll compare the effect of the bridles angles at the two altitudes to see if equilibrium is possible at both altitudes.

Secondly, the attitude (angle of attack) of the kite will change as it climbs. This will affect many things, one of the most important being the lift the kite generates. We will trim the kite on the ground for a certain angle of attack. The important question is: what will be its angle of attack at altitude? We will repeat the equilinrium Once again, knowing the effect of the bridle angle is a good place to start.

The algebra of the moment equation

Carrying out the sum of the components of the forces acting in one direction or the other is pretty straight-forward. Summing the moments is a little more tricky.

Simply put, the magnitude of the moment exerted by a given force is the magnitude of the force multiplied by the distance from the proposed axis of rotation to the line-of-acton of the force. I have used the phrase "proposed axis of rotation" because the moment exerted by a given force can be calculated around any axis. In two dimensions like, we have here, a rotation axis reduces to a point, being the rotation axis into or out of the page viewed end on. But that rotation axis or rotation point can b anywhere we want it. It could be the leading edge, the departure point, the line along which the ribs are attached to the leading edge tube, the bridle point or anywhere else. In my calculations, I chose to use the bridle point as the rotation axis, simply because I find t easier to think about the kite and its bridle as a rigid body, which rotates around the bridle point until it comes to an equilibrium.

Now, it is easy to calculate a moment when the given force is perpendicular to the line drawn from the rotation axis to the point-of-action of the force. The magnitude of the moment is simply the magnitude of the force multiplied by the dstance between the axis and the point-of-acton. If the line-of-action of the given force is at an oblique angle with respect to the axis-to-POA, then some trigonometry is needed. When the force and the point-of-action are expressed as vectors in a three-dimensional space, the vector cross-product can be used as an alternative to thinking through a lot of trigonometry. When the vectors are confined to a two-dimensional space, as they are here, the cross-product is reduced in scope. In its simplest form, the cross-product for calculating a moment in only two dimensions can be written as:

$$M_z = xF_y - yF_x \quad (B13)$$

where F_x and F_y are the components of the given force along the two perpendicular axes, x and y are the components of the line segment drawn from the proposed rtation axis / point to the point-of-action of the force and M_z is the resulting moment. If the resulting moment is algebraically positive, then the moment is parallel to the positive $\hat{z}$-axis, or counter-clockwise. If the resulting moment is algebraically negative, then the moment is parallel to the negative $\hat{z}$-axis, or clockwise. (Aside: A whole bunch of trigonometry is built into the simple expression in Equation $(B13)$. Be ye thankful.)

Appendix "C"

Listing of the Visual Basic program used to calculate the kite section's shape

The following program was developed in the Visual Basic 2010 Express version of Visual Basic. It consists of a main form (*Form1*) and eight modules.

Listing of *Form1*

```
Option Strict On
Option Explicit On

' Calculates the shape of a 2D membrane subject to a given distribution of forces.
' The membrane is wrapped around a leading edge tube and has trailing edge strings.

Public Class Form1
    Inherits System.Windows.Forms.Form

    Public Sub New()
        InitializeComponent()
        With Me
            Name = ""
            Text = "Shape of 2D kite section in an airflow"
            FormBorderStyle = Windows.Forms.FormBorderStyle.FixedSingle
            Size = New Drawing.Size(1024, 740)
            CenterToScreen()
            Visible = True
            Controls.Add(buttonSeedShape) : buttonSeedShape.BringToFront()
            Controls.Add(buttonReadForces) : buttonReadForces.BringToFront()
            Controls.Add(labelTension) : labelTension.BringToFront()
            Controls.Add(tbTension) : tbTension.BringToFront()
            Controls.Add(labelAngle) : labelAngle.BringToFront()
            Controls.Add(tbAngle) : tbAngle.BringToFront()
            Controls.Add(labelNumSegOnLECircle) : labelNumSegOnLECircle.BringToFront()
            Controls.Add(tbNumSegOnLECircle) : tbNumSegOnLECircle.BringToFront()
            Controls.Add(labelNumOfFlyingSeg) : labelNumOfFlyingSeg.BringToFront()
            Controls.Add(tbNumOfFlyingSeg) : tbNumOfFlyingSeg.BringToFront()
            Controls.Add(buttonCalculate) : buttonCalculate.BringToFront()
            Controls.Add(buttonAuto) : buttonAuto.BringToFront()
            Controls.Add(buttonHalt) : buttonHalt.BringToFront()
            Controls.Add(buttonWriteFiles) : buttonWriteFiles.BringToFront()
            Controls.Add(buttonExit) : buttonExit.BringToFront()
            Controls.Add(TextArea) : TextArea.BringToFront()
            Controls.Add(MembranePlotArea) : MembranePlotArea.BringToFront()
            Controls.Add(ForcePlotArea) : ForcePlotArea.BringToFront()
            PerformLayout()
        End With
        Initialization()
    End Sub

    '//////////////////////////
    '// Data entry - Case #1 //
    '//////////////////////////
    ' Section parameters
    Public ChordLength As Double = 1.0254       ' Reference chord length, meters
    Public NylonLength As Double = 0.98         ' Length of nylon sheet, meters
    Public TEStringLength As Double = 0.1       ' Length of trailing edge string, meters
```

```
Public LEDiameter As Double = 0.0254        ' Diameter of LE tube, meters
Public Altitude As Double = 0               ' Should be 0 or 15000, feet
' General parameters
Public NumNylonSeg As Int32 = 500           ' Number of segments in nylon
Public AngleAttackDeg As Double = 7         ' Angle of attack, degrees
Public GuessTension As Double = 110         ' GUESS Tension at leading edge, N/m
Public GuessTheta0Deg As Double = 6         ' GUESS Angle at leading edge, degrees
Public GuessNumSegOnLE As Int32 = 24        ' GUESS number of segments on LE circle
Public GuessNumOfFlyingSeg As Int32 = 476   ' GUESS number of free-flying segments

''//////////////////////////
''// Data entry - Case #3 //
''//////////////////////////
'' Section parameters
'Public ChordLength As Double = 1.0254       ' Reference chord length, meters
'Public NylonLength As Double = 0.98         ' Length of nylon sheet, meters
'Public TEStringLength As Double = 0.1       ' Length of trailing edge string, meters
'Public LEDiameter As Double = 0.0254        ' Diameter of LE tube, meters
'Public Altitude As Double = 15000           ' Should be 0 or 15000, feet
'' General parameters
'Public NumNylonSeg As Int32 = 500           ' Number of segments in nylon
'Public AngleAttackDeg As Double = 7         ' Angle of attack, degrees
'Public GuessTension As Double = 20          ' GUESS Tension at leading edge, N/m
'Public GuessTheta0Deg As Double = 16        ' GUESS Angle at leading edge, degrees
'Public GuessNumSegOnLE As Int32 = 23        ' GUESS number of segments on LE circle
'Public GuessNumOfFlyingSeg As Int32 = 477   ' GUESS number of free-flying segments

'///////////////////////////////////
'// Definition of other variables //
'///////////////////////////////////
Public NumSegOnLECircle As Int32            ' Num of nylon segments on L.E. circle
Public NumOfFlyingSeg As Int32              ' Num of free-flying nylon segments
Public X(10000) As Double                   ' X-co-ordinates of all hinges, meters
Public Y(10000) As Double                   ' X-co-ordinates of all hinges, meters
Public Tension(10000) As Double             ' Tension at the hinges, Newtons
Public ThetaRad(10000) As Double            ' Angles at the hinges, radians
Public PsiRad(10000) As Double              ' Slopes of the segments, radians
Public FperpC(10000) As Double              ' Segment forces, perpendicular to chord
Public FtangC(10000) As Double              ' Segment forces, parallel to chord
Public Muncorrected(10000) As Double        ' Segment moments, as per OpenFoam
Public DeltaS As Double                     ' Slant height of the segments
Public XTE As Double                        ' X-co-ordinate of T.E., meters
Public YTE As Double                        ' Y-co-ordinate of T.E., meters
Public AutoOn As Boolean                    ' Flag for automatic convergence
Public AngleAttackRad As Double             ' Angle of attack, radians
Public MsgStr As String                     ' General purpose message string
Public ParameterStr As String = ""          ' String containing parameters
Public BasicTextAreaContents As String      ' Mirror copy of previous TextArea

Public FxPtop(10000) As Double              ' Forces read from OpenFoam log file
Public FyPtop(10000) As Double
Public MzPtop(10000) As Double
Public FxVtop(10000) As Double
Public FyVtop(10000) As Double
Public MzVtop(10000) As Double
Public FxPbot(10000) As Double
Public FyPbot(10000) As Double
Public MzPbot(10000) As Double
```

```
Public FxVbot(10000) As Double
Public FyVbot(10000) As Double
Public MzVbot(10000) As Double
Public FxTop(10000) As Double               ' Sum of pressure and viscous forces
Public FyTop(10000) As Double
Public MzTop(10000) As Double
Public FxBot(10000) As Double
Public FyBot(10000) As Double
Public MzBot(10000) As Double
Public FxNet(10000) As Double               ' Net force on segments in X-direction
Public FyNet(10000) As Double               ' Net force on segments in Y-direction
Public MzNet(10000) As Double               ' Net moment on segments in Z-direction

'////////////////////
'// Initialization //
'////////////////////
Public Sub Initialization()
    ' Calculated parameters
    DeltaS = NylonLength / NumNylonSeg
    AngleAttackRad = AngleAttackDeg * Math.PI / 180
    XTE = ChordLength * Math.Cos(AngleAttackRad)
    YTE = -ChordLength * Math.Sin(AngleAttackRad)
    ' Prepare textboxes for display purposes
    tbTension.Text = Trim(Str(GuessTension))
    tbAngle.Text = Trim(Str(GuessTheta0Deg))
    tbNumSegOnLECircle.Text = Trim(Str(GuessNumSegOnLE))
    tbNumOfFlyingSeg.Text = Trim(Str(GuessNumOfFlyingSeg))
    Me.Refresh()
End Sub

'////////////////////////////////////////////////////////////////////////////////
'// Controls
'////////////////////////////////////////////////////////////////////////////////

Public WithEvents buttonSeedShape As New Windows.Forms.Button With _
    {.Size = New Drawing.Size(215, 30), _
     .Location = New Drawing.Point(5, 5), _
     .Text = "Seed a new shape", _
     .TextAlign = ContentAlignment.MiddleCenter}

Public WithEvents buttonReadForces As New Windows.Forms.Button With _
    {.Size = New Drawing.Size(215, 30), _
     .Location = New Drawing.Point(5, 40), _
     .Text = "Read OpenFoam forces", _
     .TextAlign = ContentAlignment.MiddleCenter}

Public labelTension As New Windows.Forms.Label With _
    {.Size = New Drawing.Size(135, 20), _
     .Location = New Drawing.Point(5, 75), _
     .Text = "L.E. Tension (N/m)", .TextAlign = ContentAlignment.MiddleLeft}

Public tbTension As New Windows.Forms.TextBox With _
    {.Size = New Drawing.Size(80, 20), _
     .Location = New Drawing.Point(140, 75), _
     .Text = "", .TextAlign = HorizontalAlignment.Left}

Public labelAngle As New Windows.Forms.Label With _
    {.Size = New Drawing.Size(135, 20), _
```

```
     .Location = New Drawing.Point(5, 100), _
     .Text = "L.E. Angle (deg)", .TextAlign = ContentAlignment.MiddleLeft}

Public tbAngle As New Windows.Forms.TextBox With _
    {.Size = New Drawing.Size(80, 20), _
     .Location = New Drawing.Point(140, 100), _
     .Text = "", .TextAlign = HorizontalAlignment.Left}

Public labelNumSegOnLECircle As New Windows.Forms.Label With _
    {.Size = New Drawing.Size(135, 20), _
     .Location = New Drawing.Point(5, 125), _
     .Text = "No. seg on LE circle", .TextAlign = ContentAlignment.MiddleLeft}

Public tbNumSegOnLECircle As New Windows.Forms.TextBox With _
    {.Size = New Drawing.Size(80, 20), _
     .Location = New Drawing.Point(140, 125), _
     .Text = "", .TextAlign = HorizontalAlignment.Left}

Public labelNumOfFlyingSeg As New Windows.Forms.Label With _
    {.Size = New Drawing.Size(135, 20), _
     .Location = New Drawing.Point(5, 150), _
     .Text = "No. flying segments", .TextAlign = ContentAlignment.MiddleLeft}

Public tbNumOfFlyingSeg As New Windows.Forms.TextBox With _
    {.Size = New Drawing.Size(80, 20), _
     .Location = New Drawing.Point(140, 150), _
     .Text = "", .TextAlign = HorizontalAlignment.Left}

Public WithEvents buttonCalculate As New Windows.Forms.Button With _
    {.Size = New Drawing.Size(215, 30), _
     .Location = New Drawing.Point(5, 175), _
     .Text = "Calculate once", .TextAlign = ContentAlignment.MiddleCenter}

Public WithEvents buttonAuto As New Windows.Forms.Button With _
    {.Size = New Drawing.Size(160, 30), _
     .Location = New Drawing.Point(5, 210), _
     .Text = "Automatic convergence", .TextAlign = ContentAlignment.MiddleCenter}

Public WithEvents buttonHalt As New Windows.Forms.Button With _
    {.Size = New Drawing.Size(50, 30), _
     .Location = New Drawing.Point(170, 210), _
     .Text = "Halt", .TextAlign = ContentAlignment.MiddleCenter}

Public WithEvents buttonWriteFiles As New Windows.Forms.Button With _
    {.Size = New Drawing.Size(215, 30), _
     .Location = New Drawing.Point(5, 245), _
     .Text = "Write GMesh and OpenFoam files", _
     .TextAlign = ContentAlignment.MiddleCenter}

Public WithEvents buttonExit As New Windows.Forms.Button With _
    {.Size = New Drawing.Size(215, 30), _
     .Location = New Drawing.Point(5, 280), _
     .Text = "Exit", .TextAlign = ContentAlignment.MiddleCenter}

Public TextArea As New Windows.Forms.Label With _
    {.Size = New Drawing.Size(215, 385), _
    .Location = New Drawing.Point(5, 315), _
    .TextAlign = ContentAlignment.TopLeft, _
```

```
    .BorderStyle = BorderStyle.FixedSingle}

Public MembranePlotArea As New Windows.Forms.Panel With _
    {.Size = New Drawing.Size(790, 395), _
    .Location = New Drawing.Point(225, 5), _
    .BorderStyle = BorderStyle.FixedSingle}

Public MembraneBitmap As New Bitmap(790, 395)

Public ForcePlotArea As New Windows.Forms.Panel With _
    {.Size = New Drawing.Size(790, 295), _
    .Location = New Drawing.Point(225, 405), _
    .BorderStyle = BorderStyle.FixedSingle}

Public ForceBitmap As New Bitmap(790, 295)

'////////////////////////////////////////////////////////////////////////////////
'// Handlers
'////////////////////////////////////////////////////////////////////////////////

Public Sub buttonSeedShape_Click() Handles buttonSeedShape.MouseClick
    ParameterStr =
        "Parameters:" & vbCrLf & _
        "  Num of segments = " & Trim(Str(NumNylonSeg)) & vbCrLf & _
        "  Chord length = " & Trim(Str(ChordLength)) & " m" & vbCrLf & _
        "  Nylon length = " & Trim(Str(NylonLength)) & " m" & vbCrLf & _
        "  TE string length = " & Trim(Str(TEStringLength)) & " m" & vbCrLf & _
        "  LE tube diameter = " & Trim(Str(LEDiameter)) & " m"
    TextArea.Text = "Seeding a new shape with:" & vbCrLf & ParameterStr
    TextArea.Refresh()
    ' Find the circular arcs
    Dim RetString As String = ""
    SeedACircularArc.SeedACircularArc( _
        NumNylonSeg, ChordLength, NylonLength, _
        LEDiameter / 2, TEStringLength, X, Y, _
        NumSegOnLECircle, NumOfFlyingSeg, RetString)
    ' Calculate some variables for display
    Dim XatMaxThickness As Double
    Dim YatMaxThickness As Double = -1.0E+20
    For I As Int32 = 1 To (NumNylonSeg + 2) Step 1
        If (Y(I) > YatMaxThickness) Then
            XatMaxThickness = X(I)
            YatMaxThickness = Y(I)
        End If
    Next I
    Dim PcntThickness As Double = 100 * YatMaxThickness / ChordLength
    Dim PcntChordAtMaxThick As Double = 100 * XatMaxThickness / ChordLength
    ' Rotate the local X-Y co-ordinates based on the reference chord line to
    ' account for the angle of attack.
    Dim CosAlpha As Double = Math.Cos(AngleAttackDeg * Math.PI / 180)
    Dim SinAlpha As Double = Math.Sin(AngleAttackDeg * Math.PI / 180)
    Dim Temp As Double
    For I As Int32 = 1 To (NumNylonSeg + 2) Step 1
        Temp = (X(I) * CosAlpha) + (Y(I) * SinAlpha)
        Y(I) = (-X(I) * SinAlpha) + (Y(I) * CosAlpha)
        X(I) = Temp
    Next I
    ' Display text
```

```
        TextArea.Text = _
            ParameterStr & vbCrLf & vbCrLf & _
            RetString & vbCrLf & vbCrLf & _
            "Results:" & vbCrLf & _
            "  X at max thickness = " & _
                FormatNumber(XatMaxThickness, 9) & " m" & vbCrLf & _
            "  Y at max thickness = " & _
                FormatNumber(YatMaxThickness, 9) & " m" & vbCrLf & _
            "  Percent thickness = " & _
                FormatNumber(PcntThickness, 9) & "%" & vbCrLf & _
            "  X at max thickness = " & _
                FormatNumber(PcntChordAtMaxThick, 9) & "%"
        TextArea.Refresh()
        ' Display the final shape
        ' Part A: Clear the graphics
        Dim h As Graphics = Graphics.FromImage(MembraneBitmap)
        h.Clear(Control.DefaultBackColor)
        h.Dispose()
        MembranePlotArea.BackgroundImage = MembraneBitmap
        MembranePlotArea.Refresh()
        ' Part B: Paint the Bitmap
        Dim e As System.EventArgs
        RenderMembrane.RenderMembrane( _
            MembranePlotArea, e, MembraneBitmap, _
            NumNylonSeg, ChordLength, LEDiameter / 2, _
            NumSegOnLECircle, NumOfFlyingSeg, _
            AngleAttackDeg, X, Y)
        ' Part C: Display the Bitmap
        MembranePlotArea.BackgroundImage = MembraneBitmap
        MembranePlotArea.Refresh()
        ' Display the numbers of segments
        tbNumSegOnLECircle.Text = Trim(Str(NumSegOnLECircle))
        tbNumofflyingSeg.Text = Trim(Str(NumOfFlyingSeg))
    End Sub

    Public Sub buttonReadForces_Click() Handles buttonReadForces.MouseClick
        Dim TimeStepUsed As Int32
        ' Read number of segments
        NumSegOnLECircle = CInt(Val(tbNumSegOnLECircle.Text))
        NumOfFlyingSeg = CInt(Val(tbNumOfFlyingSeg.Text))
        ' Read OpenFoam log file
        ' Note that ExtractOpenFoamForces() returns NumOfFlyingSeg forces in the first
        ' NumOfFlyingSeg spots in the force and moment vectors.  The force and moment
        ' figures are as OpenFoam reported them; they are not corrected for the angle of
        ' attack.
        TextArea.Text = "Reading file ..."
        TextArea.Refresh()
        ExtractOpenFoamForces.ExtractOpenFoamForces( _
            NumNylonSeg, NumSegOnLECircle, NumOfFlyingSeg, _
            FxPtop, FyPtop, MzPtop, FxVtop, FyVtop, MzVtop, _
            FxPbot, FyPbot, MzPbot, FxVbot, FyVbot, MzVbot, _
            FxTop, FyTop, MzTop, FxBot, FyBot, MzBot, TimeStepUsed)
        ' Plot the forces
        TextArea.Text = TextArea.Text & vbCrLf & _
            "    Reading is complete." & vbCrLf & _
            "Now plotting forces ..."
        TextArea.Refresh()
        ' Part A: Clear the graphics
```

```
Dim g As Graphics = Graphics.FromImage(ForceBitmap)
g.Clear(Control.DefaultBackColor)
g.Dispose()
ForcePlotArea.BackgroundImage = ForceBitmap
ForcePlotArea.Refresh()
' Part B: Paint the Bitmap
' Note that RenderForces() assumes that the force and moment figures are saved
' in the first NumOfFlyingSeg spots in their vectors.  It also assumes they
' are stated in OpenFoam's co-ordinate frame of reference.
Dim e As System.EventArgs
RenderForces.RenderForces( _
    ForcePlotArea, e, ForceBitmap, _
    NumOfFlyingSeg, ChordLength, AngleAttackDeg * Math.PI / 180, _
    FxTop, FyTop, FxBot, FyBot)
' Part C: Display the Bitmap
ForcePlotArea.BackgroundImage = ForceBitmap
ForcePlotArea.Refresh()
' Add the pressure and viscous forces along corresponding top and bottom segments
TextArea.Text = TextArea.Text & vbCrLf & _
    "    Plotting is complete." & vbCrLf & _
    "    Time step used for forces = " & Trim(Str(TimeStepUsed)) & vbCrLf & _
    "Now adding per-segment forces ..."
TextArea.Refresh()
For I As Int32 = 1 To NumNylonSeg Step 1
    FxNet(I) = FxPbot(I) + FxVbot(I) + FxPtop(I) + FxVtop(I)
    FyNet(I) = FyPbot(I) + FyVbot(I) + FyPtop(I) + FyVtop(I)
    MzNet(I) = MzPbot(I) + MzVbot(I) + MzPtop(I) + MzVtop(I)
Next I
' Unrotate the segment forces to account for the angle of attack
TextArea.Text = TextArea.Text & vbCrLf & _
    "    Addition is complete." & vbCrLf & _
    "Now rotating forces to" & vbCrLf & _
    "    the angle of attack: " & Trim(Str(AngleAttackDeg)) & " deg ..."
TextArea.Refresh()
Dim CosAlpha As Double = Math.Cos(AngleAttackRad)
Dim SinAlpha As Double = Math.Sin(AngleAttackRad)
For I As Int32 = 1 To NumNylonSeg Step 1
    FtangC(I) = (FxNet(I) * CosAlpha) - (FyNet(I) * SinAlpha)
    FperpC(I) = (FyNet(I) * CosAlpha) + (FxNet(I) * SinAlpha)
    Muncorrected(I) = MzNet(I)
Next I
' Correct forces to unit length across span, assumed to be one millimeter
TextArea.Text = TextArea.Text & vbCrLf & _
    "    Rotation is complete." & vbCrLf & _
    "Now correcting for wind tunnel width" & vbCrLf & _
    "    Assuming wind tunnel width = 1 mm"
TextArea.Refresh()
For I As Int32 = 1 To NumNylonSeg Step 1
    FtangC(I) = FtangC(I) * 1000
    FperpC(I) = FperpC(I) * 1000
    Muncorrected(I) = Muncorrected(I) * 1000
Next I
TextArea.Text = TextArea.Text & vbCrLf & _
    "    Correction is complete." & vbCrLf & _
    "Done."
TextArea.Refresh()
' Save contents of TextArea for other routines
BasicTextAreaContents = TextArea.Text
```

```
    ' Move the force and moment figures from the first NumOfFlyinySeg spots in their
    ' vectors to the appropriate spots, leaving room for the NumSegOnLECircle entries
    ' which come first.  Decrement through the list to avoid over-writing entries.
    For I As Int32 = NumOfFlyingSeg To 1 Step -1
        Dim J = NumSegOnLECircle + I
        FtangC(J) = FtangC(I)
        FperpC(J) = FperpC(I)
        Muncorrected(J) = Muncorrected(I)
    Next I
    ' Copy the forces on the first segment of the free-flying membrane onto all
    ' the segments on the leading edge circle.  This is useful if, when everything
    ' else is done, it is necessary to reduce NumSegOnLECircle by one.
    For I As Int32 = 1 To NumSegOnLECircle Step 1
        FtangC(I) = FtangC(NumSegOnLECircle + 1)
        FperpC(I) = FperpC(NumSegOnLECircle + 1)
        Muncorrected(I) = Muncorrected(NumSegOnLECircle + 1)
    Next I
End Sub

Public Sub buttonCalculate_Click() Handles buttonCalculate.MouseClick
    GuessTension = Val(tbTension.Text)
    tbTension.Text = Trim(Str(GuessTension))
    GuessTheta0Deg = Val(tbAngle.Text)
    If (GuessTheta0Deg > 90) Then GuessTheta0Deg = 90
    If (GuessTheta0Deg < -90) Then GuessTheta0Deg = -90
    tbAngle.Text = Trim(Str(GuessTheta0Deg))
    GuessNumSegOnLE = CInt(Val(tbNumSegOnLECircle.Text))
    tbNumSegOnLECircle.Text = Trim(Str(GuessNumSegOnLE))
    NumSegOnLECircle = GuessNumSegOnLE
    NumOfFlyingSeg = CInt(Val(tbNumOfFlyingSeg.Text))
    Me.Refresh()
    AutoOn = False
    ' Note that OneMarchAlongMembrane() assumes the force and moment figures are
    ' saved in the spots in their vectors which correspond to their order on the
    ' nylon membrane, starting from its front edge.
    OneMarchAlongMembrane.OneMarchAlongMembrane( _
        NumNylonSeg, NumSegOnLECircle, _
        ChordLength, NylonLength, LEDiameter / 2, TEStringLength, _
        FtangC, FperpC, Muncorrected, _
        GuessTension, GuessTheta0Deg * Math.PI / 180, X, Y, Tension, _
        ThetaRad, PsiRad, NumOfFlyingSeg)
    ' Display the final shape
    ' Part A: Clear the graphics
    Dim g As Graphics = Graphics.FromImage(MembraneBitmap)
    g.Clear(Control.DefaultBackColor)
    g.Dispose()
    MembranePlotArea.BackgroundImage = MembraneBitmap
    MembranePlotArea.Refresh()
    ' Part C: Paint the Bitmap
    Dim e As System.EventArgs
    RenderMembrane.RenderMembrane( _
        MembranePlotArea, e, MembraneBitmap, _
        NumNylonSeg, ChordLength, LEDiameter / 2, _
        NumSegOnLECircle, NumOfFlyingSeg, _
        AngleAttackDeg, X, Y)
    ' Part D: Display the Bitmap
    MembranePlotArea.BackgroundImage = MembraneBitmap
    MembranePlotArea.Refresh()
```

```
End Sub

Public Sub buttonAuto_Click() Handles buttonAuto.MouseClick
    GuessTension = Val(tbTension.Text)
    tbTension.Text = Trim(Str(GuessTension))
    GuessTheta0Deg = Val(tbAngle.Text)
    If (GuessTheta0Deg > 90) Then GuessTheta0Deg = 90
    If (GuessTheta0Deg < -90) Then GuessTheta0Deg = -90
    tbAngle.Text = Trim(Str(GuessTheta0Deg))
    GuessNumSegOnLE = CInt(Val(tbNumSegOnLECircle.Text))
    tbNumSegOnLECircle.Text = Trim(Str(GuessNumSegOnLE))
    NumSegOnLECircle = GuessNumSegOnLE
    NumOfFlyingSeg = CInt(Val(tbNumOfFlyingSeg.Text))
    Me.Refresh()
    AutoOn = True
    ConvergeToShape.ConvergeToShape( _
        NumNylonSeg, NumSegOnLECircle, _
        ChordLength, NylonLength, LEDiameter / 2, TEStringLength, _
        FtangC, FperpC, Muncorrected, _
        GuessTension, GuessTheta0Deg, X, Y, Tension, _
        ThetaRad, PsiRad, NumOfFlyingSeg, _
        BasicTextAreaContents)
    ' Calculate some variables for display.  These are calculated before rotating
    ' the co-ordinates from the reference chord to the angle of attack.
    Dim PsiDepPtDeg As Double = PsiRad(NumSegOnLECircle + 1) * 180 / Math.PI
    Dim PsiRightDeg As Double = PsiRad(NumNylonSeg) * 180 / Math.PI
    Dim XatMaxThickness As Double
    Dim YatMaxThickness As Double = -1.0E+20
    For I As Int32 = 1 To (NumNylonSeg + 1) Step 1
        If (Y(I) > YatMaxThickness) Then
            XatMaxThickness = X(I)
            YatMaxThickness = Y(I)
        End If
    Next I
    Dim PcntThickness As Double = 100 * YatMaxThickness / ChordLength
    Dim PcntChordAtMaxThick As Double = 100 * XatMaxThickness / ChordLength
    ' Display the results.  These are displayed before rotating the co-ordinates
    ' from the reference chord to the angle of attack.
    ParameterStr = _
        "Parameters:" & vbCrLf & _
        "  Chord length = " & Trim(Str(ChordLength)) & " m" & vbCrLf & _
        "  Nylon length = " & Trim(Str(NylonLength)) & " m" & vbCrLf & _
        "  Num of segments = " & Trim(Str(NumNylonSeg)) & vbCrLf & _
        "  Angle of attack = " & Trim(Str(AngleAttackDeg)) & " deg"
    TextArea.Text = _
        ParameterStr & vbCrLf & vbCrLf & _
        "Results:" & vbCrLf & _
        "  Tension at Departure point = " & _
            FormatNumber(Tension(NumSegOnLECircle + 1), 9) & " N/m" & vbCrLf & _
        "  Tension at T.E. = " & _
            FormatNumber(Tension(NumNylonSeg + 1), 9) & " N/m" & vbCrLf & _
        "  Tension angle at Departure point = " & _
            FormatNumber(ThetaRad(NumSegOnLECircle + 1) * 180 / Math.PI, 9) & _
            " deg" & vbCrLf & _
        "  Tension angle at T.E. = " & _
            FormatNumber(ThetaRad(NumNylonSeg + 1) * 180 / Math.PI, 9) & _
            " deg" & vbCrLf & _
        "  Slope at Departure point = " & _
```

```
        FormatNumber(PsiDepPtDeg, 9) & " deg" & vbCrLf & _
    "  Slope at T.E. = " & _
        FormatNumber(PsiRightDeg, 9) & " deg" & vbCrLf & _
    "  X at max thickness = " & _
        FormatNumber(XatMaxThickness, 9) & " m" & vbCrLf & _
    "  Y at max thickness = " & _
        FormatNumber(YatMaxThickness, 9) & " m" & vbCrLf & _
    "  Percent thickness = " & _
        FormatNumber(PcntThickness, 9) & "%" & vbCrLf & _
    "  X at max thickness = " & _
        FormatNumber(PcntChordAtMaxThick, 9) & "%"
TextArea.Refresh()
' Rotate the local X-Y co-ordinates based on the reference chord line to
' account for the angle of attack.
Dim CosAlpha As Double = Math.Cos(AngleAttackDeg * Math.PI / 180)
Dim SinAlpha As Double = Math.Sin(AngleAttackDeg * Math.PI / 180)
Dim Temp As Double
For I As Int32 = 1 To (NumNylonSeg + 2) Step 1
    ' Rotate all points
    Temp = (X(I) * CosAlpha) + (Y(I) * SinAlpha)
    Y(I) = (-X(I) * SinAlpha) + (Y(I) * CosAlpha)
    X(I) = Temp
Next I
' Display the final shape
' Part A: Clear the graphics
Dim g As Graphics = Graphics.FromImage(MembraneBitmap)
g.Clear(Control.DefaultBackColor)
g.Dispose()
MembranePlotArea.BackgroundImage = MembraneBitmap
MembranePlotArea.Refresh()
' Part B: Paint the Bitmap
Dim e As System.EventArgs
RenderMembrane.RenderMembrane( _
    MembranePlotArea, e, MembraneBitmap, _
    NumNylonSeg, ChordLength, LEDiameter / 2, _
    NumSegOnLECircle, NumOfFlyingSeg, _
    AngleAttackDeg, X, Y)
' Part C: Display the Bitmap
MembranePlotArea.BackgroundImage = MembraneBitmap
MembranePlotArea.Refresh()
' Set the final values in the textboxes
tbTension.Text = Trim(Str(Tension(NumSegOnLECircle + 1)))
tbAngle.Text = Trim(Str(ThetaRad(NumSegOnLECircle + 1) * 180 / Math.PI))
'
' ********************************************************************
' ***** Check the angle at the departure point for consistency *****
' ***** All angles are in the rotated frame of reference       *****
' ********************************************************************
Dim MembraneSlopeAtDepPtDeg As Double
MembraneSlopeAtDepPtDeg = _
    (PsiRad(NumSegOnLECircle + 1) * 180 / Math.PI) - AngleAttackDeg
Dim TensionSlopeAtDepPtDeg As Double
TensionSlopeAtDepPtDeg = _
    (ThetaRad(NumSegOnLECircle + 1) * 180 / Math.PI) - AngleAttackDeg
' Calculate the LE tube slope at the departure point
Dim LETubeSlopeAtDepPtDeg As Double
Dim Run As Double
Dim Rise As Double
```

```
Run = X(NumSegOnLECircle + 1) - (0.5 * LEDiameter * Math.Cos(AngleAttackRad))
Rise = Y(NumSegOnLECircle + 1) - (-0.5 * LEDiameter * Math.Sin(AngleAttackRad))
LETubeSlopeAtDepPtDeg = (Math.Atan2(Rise, Run) * 180 / Math.PI) - 90
' Calculate the subtended angle of a typical segment on the leading edge tube
Dim SubtendedAngleRad As Double
Dim SubtendedAngleDeg As Double
SubtendedAngleRad = (NylonLength / NumNylonSeg) / (LEDiameter / 2)
SubtendedAngleDeg = SubtendedAngleRad * 180 / Math.PI
' Calculate slope before the departure point
Dim LETubeSlopeBeforeDepPtDeg As Double
LETubeSlopeBeforeDepPtDeg = LETubeSlopeAtDepPtDeg + SubtendedAngleDeg
' Calculate slope after the departure point
Dim LETubeSlopeAfterDepPtDeg As Double
LETubeSlopeAfterDepPtDeg = LETubeSlopeAtDepPtDeg - SubtendedAngleDeg
' Check if membrane leaves LE tube at too great an angle
If (MembraneSlopeAtDepPtDeg > LETubeSlopeBeforeDepPtDeg) Then
    MsgBox( _
        "**************" & vbCrLf & _
        "***Warning***" & vbCrLf & _
        "**************" & vbCrLf & _
        "Angle of membrane at the departure point = " & _
        Trim(Str(MembraneSlopeAtDepPtDeg)) & " degrees" & vbCrLf & _
        "Angle of tension at the departure point = " & _
        Trim(Str(TensionSlopeAtDepPtDeg)) & " degrees" & vbCrLf & _
        "Angle of LE tube to the left of the departure point = " & _
        Trim(Str(LETubeSlopeBeforeDepPtDeg)) & " degrees" & vbCrLf & _
        "Angle of LE tube at the departure point = " & _
        Trim(Str(LETubeSlopeAtDepPtDeg)) & " degrees" & vbCrLf & _
        "Angle of LE tube to the right of the departure point = " & _
        Trim(Str(LETubeSlopeAfterDepPtDeg)) & " degrees" & vbCrLf & vbCrLf & _
        "Slope of membrane at LE may be too high." & vbCrLf & _
        "Try reducing NumSegOnLECircle by one.  As the offset," & vbCrLf & _
        "add one to NumSegOnMembrane.  The result may be a better fit.")
    Return
    Exit Sub
End If
' Check if membrane leaves LE tube at too an angle
If (MembraneSlopeAtDepPtDeg < LETubeSlopeAfterDepPtDeg) Then
    MsgBox( _
        "************" & vbCrLf & _
        "***Error***" & vbCrLf & _
        "************" & vbCrLf & _
        "Angle of membrane at the departure point = " & _
        Trim(Str(MembraneSlopeAtDepPtDeg)) & " degrees" & vbCrLf & _
        "Angle of tension at the departure point = " & _
        Trim(Str(TensionSlopeAtDepPtDeg)) & " degrees" & vbCrLf & _
        "Angle of LE tube to the left of the departure point = " & _
        Trim(Str(LETubeSlopeBeforeDepPtDeg)) & " degrees" & vbCrLf & _
        "Angle of LE tube at the departure point = " & _
        Trim(Str(LETubeSlopeAtDepPtDeg)) & " degrees" & vbCrLf & _
        "Angle of LE tube to the right of the departure point = " & _
        Trim(Str(LETubeSlopeAfterDepPtDeg)) & " degrees" & vbCrLf & vbCrLf & _
        "Slope of membrane is too low; it intersects the LE circle." & vbCrLf & _
        "Increase NumSegOnLECircle by one and decrease" & vbCrLf & _
        "NumSegOnMembrane by one and re-run.  This is mandatory.")
    Return
    Exit Sub
End If
```

```
        ' Confirmation message regarding successful convergence
        MsgBox( _
            "Successful convergence:" & vbCrLf & _
            "Angle of membrane at the departure point = " & _
            Trim(Str(MembraneSlopeAtDepPtDeg)) & " degrees" & vbCrLf & _
            "Angle of tension at the departure point = " & _
            Trim(Str(TensionSlopeAtDepPtDeg)) & " degrees" & vbCrLf & _
            "Angle of LE tube to the left of the departure point = " & _
            Trim(Str(LETubeSlopeBeforeDepPtDeg)) & " degrees" & vbCrLf & _
            "Angle of LE tube at the departure point = " & _
            Trim(Str(LETubeSlopeAtDepPtDeg)) & " degrees" & vbCrLf & _
            "Angle of LE tube to the right of the departure point = " & _
            Trim(Str(LETubeSlopeAfterDepPtDeg)) & " degrees")
    End Sub

    Public Sub buttonHalt_Click() Handles buttonHalt.MouseClick
        AutoOn = False
    End Sub

    Public Sub buttonWriteFiles_Click() Handles buttonWriteFiles.MouseClick
        WriteGMeshFile.WriteGMeshFile( _
            NumNylonSeg, X, Y, _
            ChordLength, NylonLength, LEDiameter / 2, _
            AngleAttackDeg, NumSegOnLECircle, NumOfFlyingSeg)
        TextArea.Text = TextArea.Text & vbCrLf & vbCrLf & "Gmesh text file written."
        WriteOpenFoamFunction.WriteOpenFoamFunction(NumOfFlyingSeg, Altitude)
        TextArea.Text = TextArea.Text & vbCrLf & vbCrLf & "OpenFoam text file written."
        TextArea.Refresh()
    End Sub

    Public Sub buttonExit_Click() Handles buttonExit.MouseClick
        Application.Exit()
    End Sub

End Class
```

Listing of Module *SeedACircularArc*

```
Option Strict On
Option Explicit On

Public Module SeedACircularArc

    ' The subroutine in this module prepares a preliminary shape for the first OpenFoam
    ' run.  The routine uses the formula from the Appendix "B" of the text titled "The
    ' equations of the circular arc used as the seed shape".  This module ignores the
    ' angle of attack and carries out all calculations with the X-axis being the
    ' reference chord line.  In a similar way it ignores any offset of the leading edge
    ' to the point (XLE, YLE).

    ' The input variables are:
    '    NumNylonSeg = the number of segments into which the nylon sheet is divided
    '    ChordLength = length of reference chord, meters
    '    NylonLength = length of nylon, meters
    '    LERadius = radius of the leading edge tube, meters
    '    TELength = length of the trailing edge string, meters
```

```
' The calculated quantities which are returned to the calling procedure are:
'    X(NumNylonSeg + 2) = X-co-ordinates of all hinges, meters
'    Y(NumNylonSeg + 2) = Y-co-ordinates of all hinges, meters
'    NumSegOnLECircle = number of segments on the leading edge circle
'    NumOfFlyingSeg = number of free-flying nylon segments

Public Sub SeedACircularArc( _
    ByVal NumNylonSeg As Int32, _
    ByVal ChordLength As Double, ByVal NylonLength As Double, _
    ByVal LERadius As Double, ByVal TELength As Double, _
    ByRef X() As Double, ByRef Y() As Double, _
    ByRef NumSegOnLECircle As Int32, _
    ByRef NumOfFlyingSeg As Int32, _
    ByRef RetString As String)
    '
    Dim MaxRmemPctError As Double = Val("1e-12")
    Dim RmemPctError As Double
    Dim OldRmembrane As Double
    Dim NewRmembrane As Double
    Dim Alpha As Double
    Dim Beta As Double
    Dim lTemp As Double
    ' Make an initial guess for Rmembrane
    NewRmembrane = 0.75 * ChordLength
    Do
        ' Calculate Alpha using Equation (B16)
        SolveForAlpha(NylonLength, LERadius, TELength, NewRmembrane, Alpha)
        ' Calculate Beta using Equation (B14).  Beta must be greater than zero, but
        ' it can also be greater than 90 degrees, so include a quadrant check.
        lTemp = (NewRmembrane * Math.Cos(Alpha)) - (TELength * Math.Sin(Alpha))
        Beta = Math.Asin(lTemp / (NewRmembrane - LERadius))
        If (Beta < 0) Then
            Beta = (2 * Math.PI) - Beta
        End If
        ' Calculate Rmembrane using Equation (B12A)
        OldRmembrane = NewRmembrane
        NewRmembrane = _
            (ChordLength - _
            (TELength * Math.Cos(Alpha)) - _
            (LERadius * (1 - Math.Cos(Beta)))) / _
            (Math.Sin(Alpha) + Math.Cos(Beta))
        ' Check for convergence
        RmemPctError = Math.Abs((NewRmembrane - OldRmembrane) / OldRmembrane)
        If (RmemPctError < MaxRmemPctError) Then
            Exit Do
        End If
        ' Update the guess for Rmembrane.  If the difference is less than 10%,
        ' then feed back 75% of the error.  Otherwise, use the average.
        If (RmemPctError < 0.1) Then
            NewRmembrane = OldRmembrane + (0.75 * (NewRmembrane - OldRmembrane))
        Else
            NewRmembrane = (NewRmembrane + OldRmembrane) / 2
        End If
        ' Display the details of the current iteration
        RetString = _
            "Interim results:" & vbCrLf & _
            "  Rmembrane = " & FormatNumber(NewRmembrane, 11) & " m" & vbCrLf & _
            "  Alpha = " & FormatNumber(Alpha * 180 / Math.PI, 11) & _
```

```
            " deg" & vbCrLf & _
        "  Beta = " & FormatNumber(Beta * 180 / Math.PI, 11) & _
            " deg" & vbCrLf & _
        "  Gamma = " & FormatNumber( _
            ((Math.PI / 2) + Alpha - Beta) * 180 / Math.PI, 11) & _
            " deg" & vbCrLf & _
        "  Xmembrane = " & FormatNumber(ChordLength - _
            (NewRmembrane * Math.Sin(Alpha)) - _
            (TELength * Math.Cos(Alpha)), 11) & " m" & vbCrLf & _
        "  Ymembrane = " & FormatNumber( _
            (NewRmembrane * Math.Cos(Alpha)) - _
            (TELength * Math.Sin(Alpha)), 11) & " m"
    Form1.TextArea.Text = RetString
    Form1.TextArea.Refresh()
    ' Wait 20ms between iterations
    Threading.Thread.Sleep(20)
    ' Give other processes a chance
    Application.DoEvents()
Loop
'
' Calculate the center of the membrane's circle using Equation (B11)
Dim Xmembrane As Double
Dim Ymembrane As Double
Xmembrane = _
    ChordLength - _
    (NewRmembrane * Math.Sin(Alpha)) - _
    (TELength * Math.Cos(Alpha))
Ymembrane = (NewRmembrane - LERadius) * Math.Sin(Beta)
'
' Calculate the segmentation of the flexible material
Dim DeltaS As Double = NylonLength / NumNylonSeg
'
' Discretize curve #1 using Equations (B19) and (B21).  Remember that the nylon
' sheet is attached to the leading edge tube at a point which is 135 degrees
' around the bottom of the tube from the L.E.
NumSegOnLECircle = CInt((Beta + (0.75 * Math.PI)) * LERadius / DeltaS)
Dim DeltaTheta As Double = (Beta + (0.75 * Math.PI)) / NumSegOnLECircle
For Icurve1 As Int32 = 1 To (NumSegOnLECircle + 1) Step 1
    Dim Argument As Double = (-0.75 * Math.PI) + ((Icurve1 - 1) * DeltaTheta)
    X(Icurve1) = LERadius * (1 - Math.Cos(Argument))
    Y(Icurve1) = LERadius * Math.Sin(Argument)
Next Icurve1
'
'////////////////////////////////////////////////////////////////////////////
'// Intersections of the 1D curves
'// Bear in mind that part of the leading edge circle is not covered by the
'// membrane.  This exposed part is also part of the object placed in the virtual
'// wind tunnel.  We will need to make sure that the arc and points which
'// represent this exposed part do not intersect the curves being calculated
'// here.
'////////////////////////////////////////////////////////////////////////////
'
' Discretize curve #2 using Equations (B22) and (B23)
Dim Beta2Prime As Double = Math.Atan2( _
    Ymembrane + Y(NumSegOnLECircle + 1), Xmembrane - X(NumSegOnLECircle + 1))
NumOfFlyingSeg = CInt((((Math.PI / 2) + Alpha - Beta2Prime) * _
    NewRmembrane) / DeltaS)
For Icurve2 As Int32 = 1 To NumOfFlyingSeg Step 1
```

```
        Dim Argument As Double = Beta2Prime + (Icurve2 * DeltaS / NewRmembrane)
        X(NumSegOnLECircle + 1 + Icurve2) = _
            Xmembrane - (NewRmembrane * Math.Cos(Argument))
        Y(NumSegOnLECircle + 1 + Icurve2) = _
            (NewRmembrane * Math.Sin(Argument)) - Ymembrane
    Next Icurve2
    '
    ' There is no need to discretize the trailing edge string
    X(NumNylonSeg + 2) = ChordLength
    Y(NumNylonSeg + 2) = 0
End Sub

'////////////////////////////////////////////////////////////////////////////
'// Subroutine SolveForAlpha
'// This subroutine iterates Equation (B16) to find the value of Alpha for a given
'// value of Rmembrane.
'////////////////////////////////////////////////////////////////////////////
Private Sub SolveForAlpha( _
    ByVal lNylonLength As Double, _
    ByVal lLERadius As Double, _
    ByVal lTELength As Double, _
    ByVal lMembraneRadius As Double, _
    ByRef lAlpha As Double)
    Dim S_FS As Double = lNylonLength + lTELength
    Dim MinimumAlpha As Double
    Dim MaxAlphaError As Double = Val("1e-12")
    '
    ' Step #1: Calculate the smallest value of Alpha for which angle Beta exists.
    MinimumAlpha = Math.Atan(lTELength / lMembraneRadius)
    '
    ' Step #2: By direct search, find two values of Alpha which bound the desired
    '          solution of Equation (B16).  For one of these bounds, the RHS side of
    '          Equation (B16) will represent an algebraically positive error.  For
    '          the other bound, the RHS of Equation (B16) will represent an
    '          algebraically negative error.  Begin with AlphaLow set equal to the
    '          value of MinimumAlpha and increment it up to 90 degrees in steps of
    '          one-thousandth of a degree.  At each step, test a second value of
    '          alpha, called AlphaHigh, which is one-thousandth of a degree greater
    '          than AlphaLow.  When a pair of AlphaLow and AlphaHigh have errors of
    '          opposite sign, then the bisection search can begin.
    Dim DeltaAlpha As Double = (Math.PI / 180) / 1000
    Dim AlphaHigh As Double = MinimumAlpha
    Dim Term1, Term2, Term3, Term4, RHSHigh, ErrorHigh As Double
    Term1 = (lMembraneRadius * Math.Cos(AlphaHigh)) - _
        (lTELength * Math.Sin(AlphaHigh))
    Term2 = Math.Asin(Term1 / (lMembraneRadius - lLERadius))
    If (Term2 < 0) Then
        Term2 = (2 * Math.PI) - Term2
    End If
    Term3 = Term2 * (1 - (lLERadius / lMembraneRadius))
    Term4 = Term3 + ((S_FS - lTELength) / lMembraneRadius)
    RHSHigh = Term4 - ((Math.PI / 2) * (1 + (1.5 * lLERadius / lMembraneRadius)))
    ErrorHigh = AlphaHigh - RHSHigh
    If (Math.Abs(ErrorHigh) < MaxAlphaError) Then
        lAlpha = AlphaHigh
        Return
        Exit Sub
    End If
```

```
Dim AlphaLow, RHSLow, ErrorLow As Double
For Ialpha As Int32 = 1 To 90000 Step 1
    AlphaLow = AlphaHigh
    RHSLow = RHSHigh
    ErrorLow = ErrorHigh
    AlphaHigh = AlphaLow + DeltaAlpha
    If (AlphaHigh > (Math.PI / 2)) Then
        MsgBox("Error: Could not find bounding angles alpha.")
    End If
    Term1 = (lMembraneRadius * Math.Cos(AlphaHigh)) - _
        (lTELength * Math.Sin(AlphaHigh))
    Term2 = Math.Asin(Term1 / (lMembraneRadius - lLERadius))
    If (Term2 < 0) Then
        Term2 = (2 * Math.PI) - Term2
    End If
    Term3 = Term2 * (1 - (lLERadius / lMembraneRadius))
    Term4 = Term3 + ((S_FS - lTELength) / lMembraneRadius)
    RHSHigh = Term4 - ((Math.PI / 2) * (1 + (1.5 * lLERadius / lMembraneRadius)))
    ErrorHigh = AlphaHigh - RHSHigh
    If (Math.Abs(ErrorHigh) < MaxAlphaError) Then
        lAlpha = AlphaHigh
        Return
        Exit Sub
    End If
    If ((ErrorHigh * ErrorLow) < 0) Then
        Exit For
    End If
Next Ialpha
'
' Step #3: Use the bounding values of alpha, and bisection, to home in on
'          the value of alpha.
Dim AlphaMid, RHSMid, ErrorMid As Double
Do
    AlphaMid = (AlphaHigh + AlphaLow) / 2
    Term1 = (lMembraneRadius * Math.Cos(AlphaMid)) - _
        (lTELength * Math.Sin(AlphaMid))
    Term2 = Math.Asin(Term1 / (lMembraneRadius - lLERadius))
    If (Term2 < 0) Then
        Term2 = (2 * Math.PI) - Term2
    End If
    Term3 = Term2 * (1 - (lLERadius / lMembraneRadius))
    Term4 = Term3 + ((S_FS - lTELength) / lMembraneRadius)
    RHSMid = Term4 - ((Math.PI / 2) * (1 + (1.5 * lLERadius / lMembraneRadius)))
    ErrorMid = AlphaMid - RHSMid
    If (Math.Abs(ErrorMid) < MaxAlphaError) Then
        lAlpha = AlphaMid
        Return
        Exit Sub
    End If
    ' Bisect the region
    If ((ErrorHigh * ErrorMid) < 0) Then
        AlphaLow = AlphaMid
        RHSLow = RHSMid
        ErrorLow = ErrorMid
    Else
        AlphaHigh = AlphaMid
        RHSHigh = RHSMid
        ErrorHigh = ErrorMid
```

```
            End If
            ' Give other processes a chance to work
            Application.DoEvents()
        Loop
    End Sub

End Module
```

Listing of Module *WriteGMeshFile*

```
Option Strict On
Option Explicit On

Public Module WriteGMeshFile

    ' The X-Y co-ordinates of the membrane which are passed as arguments to the
    ' subroutine in this module include the rotation due to the angle of attack as well
    ' as any displacement given to the leading edge.

    ' The X(),Y() co-ordinates of the hinge points define the lower surface of the
    ' membrane.  The upper surface is defined by translating those points outwards in
    ' directions perpendicular to the slope of the segment immediately to the right.
    ' Note that the hinge points on the lower surface which lie on the leading edge
    ' circle are not written as vertices.  They are interior points when the leading
    ' edge circle is included in the profile.  Similarly, the trailing edge string is
    ' not used in creating the mesh.
    '
    ' That part of the circumference of the leading edge circle which is not covered by
    ' the impermeable membrane is included in the profile of the airfoil.  The exposed
    ' part of the circumference is divided as closely as possible into segments with the
    ' same length as those into which the flexible surface is divided.

    '////////////////
    '// Data entry //
    '////////////////
    ' The sides of the wind tunnel ("WT") are established with respect to the L.E. of the
    ' airfoil in terms of multiples of the chord length.
    Public WTMultipleAhead As Double = 3
    Public WTMultipleAstern As Double = 4
    Public WTMultipleAbove As Double = 3
    Public WTMultipleBelow As Double = 3.5
    Public WTDistanceAhead As Double = WTMultipleAhead * Form1.ChordLength
    Public WTDistanceAstern As Double = WTMultipleAstern * Form1.ChordLength
    Public WTDistanceAbove As Double = WTMultipleAbove * Form1.ChordLength
    Public WTDistanceBelow As Double = WTMultiplebelow * Form1.ChordLength

    ' Size of mesh on membrane.  Specify the Number of Points per Segment ("NPS").
    Public Membrane_NPS As Int32 = 2
    Public lcMembrane As Double

    ' Size of mesh on wind tunnel.  Specify the characteristic length in meters.
    Public lcWT As Double = 0.1

    ' Thicknesses
    Public MemThickness As Double = 0.001               ' Membrane thickness, in meters
    Public MemHalfThick As Double = MemThickness / 2
```

```
Public WTWidth As Double = 0.001                    ' Depth of wind tunnel, in meters
Public WTHalfWidth As Double = WTWidth / 2

' Output file name
Public GMeshTextFileName As String = "KiteMembrane.geo.txt"

'////////////////////////////////////
'// Definition of other variables //
'////////////////////////////////////
' Co-ordinates of points on the exposed part of the leading edge circle
Public NumExposedSegOnLECircle As Int32
Public P(100) As Double
Public Q(100) As Double
' Co-ordinates of points on the outer surface of the membrane
Public Xoutside(10000) As Double
Public Youtside(10000) As Double
' POINT reference indices, all on the right side of wind tunnel
Public FirstPtOnTopR As Int32     ' Index of 1st Point on flying nylon's top (LE)
Public LastPtOnTopR As Int32      ' Index of last Point on flying nylon's top (TE)
Public FirstPtOnBotR As Int32     ' Index of 1st Point on flying nylon's bottom (LE)
Public LastPtOnBotR As Int32      ' Index of last Point on flying nylon's bottom (TE)
Public FirstPtOnLECircle As Int32 ' Indices on LE circle, clockwise, from ...
Public LastPtOnLECircle As Int32  ' ... departure point + 1 to departure point - 1
' LINE reference indices, all on the right side of the wind tunnel
Public FirstLnAlngTopR As Int32   ' Index of 1st Line on flying nylon's top, LE->TE
Public LastLnAlngTopR As Int32    ' Index of last Line on flying nylon's top, LE->TE
Public FirstLnAlngBotR As Int32   ' Index of 1st Line on flying nylon's bot, LE->TE
Public LastLnAlngBotR As Int32    ' Index of last Line on flying nylon's bot, LE->TE
Public FirstLnAroundLE As Int32   ' Index of 1st line clockwise around LE circle
Public LastLnAroundLE As Int32    ' Index of last Line clockwise around LE circle
' Wind tunnel reference indices
Public FirstPtOnWTR As Int32      ' Index of first Point on wind tunnel, right side
Public FirstLnAlngWTR As Int32    ' Index of first Line around WT, right side
' LINE LOOP reference indices
Public MemLineLoop As Int32       ' Index of Line Loop around membrane, right side
Public WTLineLoop As Int32        ' Index of Line Loop around wind tunnel, right side
' SURFACE LOOP reference indices
Public WTSurface As Int32         ' Index of Plane Surface of WT, right side
' Other variables
Public AngleAttackRad As Double
Public Filewriter As System.IO.StreamWriter

Public Sub WriteGMeshFile( _
    ByVal NumNylonSeg As Int32, _
    ByRef Xinside() As Double, ByRef Yinside() As Double, _
    ByVal ChordLength As Double, _
    ByVal NylonLength As Double, _
    ByVal LERadius As Double,
    ByVal AngleAttackDeg As Double, _
    ByRef NumSegOnLECircle As Int32, _
    ByRef NumOfFlyingSeg As Int32)
    '
    ' Step #1: Calculate the characteristic length on the airfoil
    lcMembrane = NylonLength / (NumNylonSeg * Membrane_NPS)
    '
    ' Step #2: Open the output file
    Filewriter = New System.IO.StreamWriter(GMeshTextFileName)
    '
```

```
' Step #3: Write header information to the output file
Filewriter.Write( _
    "// Shape of kite membrane in a 2D airflow" & vbCrLf & _
    "//   Chord length (m) = " & Trim(Str(ChordLength)) & vbCrLf & _
    "//   Nylon length (m) = " & Trim(Str(NylonLength)) & vbCrLf & _
    "//   L.E. radius (m) = " & Trim(Str(LERadius)) & vbCrLf & _
    "Mesh.RandomFactor = 1e-11;" & vbCrLf & _
    "Geometry.AutoCoherence = 1;" & vbCrLf & _
    "Geometry.HighlightOrphans = 1;" & vbCrLf & _
    "Geometry.MatchGeomAndMesh = 1;" & vbCrLf & _
    "Geometry.SnapX = 0;" & vbCrLf & _
    "Geometry.SnapY = 0;" & vbCrLf & _
    "Geometry.SnapZ = 0;" & vbCrLf & _
    "Geometry.Tolerance = 1e-15;" & vbCrLf & _
    "//" & vbCrLf & _
    "// Parameters" & vbCrLf & _
    "WTDistanceAhead = " & Trim(Str(WTDistanceAhead)) & ";" & vbCrLf & _
    "WTDistanceAstern = " & Trim(Str(WTDistanceAstern)) & ";" & vbCrLf & _
    "WTDistanceAbove = " & Trim(Str(WTDistanceAbove)) & ";" & vbCrLf & _
    "WTDistanceBelow = " & Trim(Str(WTDistanceBelow)) & ";" & vbCrLf & _
    "lcWT = " & Trim(Str(lcWT)) & ";" & vbCrLf & _
    "WTWidth = " & Trim(Str(WTWidth)) & ";" & vbCrLf & _
    "WTHalfWidth = " & Trim(Str(WTHalfWidth)) & ";" & vbCrLf & _
    "MembraneThickness = " & Trim(Str(MemThickness)) & ";" & vbCrLf & _
    "Membrane_NPS = " & Trim(Str(Membrane_NPS)) & ";" & vbCrLf & _
    "lcMembrane = " & Trim(Str(lcMembrane)) & ";" & vbCrLf)
'
' Step #4: Calculate the co-ordinates of all points on the outside surface of the
'          nylon membrane.  All points on the nylon membrane, including those
'          which lie on the surface of the leading edge circle, are translated
'          outwards and upwards.
For I As Int32 = 1 To (NumNylonSeg + 1) Step 1
    Dim Rise As Double = Yinside(I + 1) - Yinside(I)
    Dim Run As Double = Xinside(I + 1) - Xinside(I)
    Dim Angle As Double = Math.Atan2(Rise, Run)
    Xoutside(I) = Xinside(I) + (MemThickness * Math.Cos(Angle + (Math.PI / 2)))
    Youtside(I) = Yinside(I) + (MemThickness * Math.Sin(Angle + (Math.PI / 2)))
Next I
'
' Step #5: Calculate the co-ordinates of all points on the exposed surface of the
'          leading edge circle.  The exposed arc length is discretized into
'          segments with the same length as those on the nylon membrane.  The
'          new co-ordinates are stored in the vectors P() and Q(), which are
'          zero-based.  Note that P(0) and Q(0) are the co-ordinates of the front
'          edge of the nylon (the "start of membrane").  Similarly,
'          P(NumExposedSegOnLECircle) and Q(NumExposedSegOnLECircle) are the
'          co-ordinates of the departure point.  Both of these points are already
'          included in the membrane's list of points so they will not be
'          re-declared in the GMesh text file.  Remember that the Xinside() and
'          Yinside() co-ordinates have already been corrected for the angle of
'          attack and for the leading edge offset, if any.  The vector dot-
'          -product in two dimensions is used to calculate the exposed angle.
Dim LECircleCenterX As Double
Dim LECircleCenterY As Double
LECircleCenterX = LERadius * Math.Cos(AngleAttackDeg * Math.PI / 180)
LECircleCenterY = -LERadius * Math.Sin(AngleAttackDeg * Math.PI / 180)
Dim RunToDeparturePoint As Double
Dim RiseToDeparturePoint As Double
```

```
Dim RunToStartOfMembrane As Double
Dim RiseToStartOfMembrane As Double
RunToDeparturePoint = Xinside(NumSegOnLECircle + 1) - LECircleCenterX
RiseToDeparturePoint = Yinside(NumSegOnLECircle + 1) - LECircleCenterY
RunToStartOfMembrane = Xinside(1) - LECircleCenterX
RiseToStartOfMembrane = Yinside(1) - LECircleCenterY
Dim ExposedAngle As Double
ExposedAngle = Math.Acos( _
    (RunToDeparturePoint * RunToStartOfMembrane / (LERadius * LERadius)) + _
    (RiseToDeparturePoint * RiseToStartOfMembrane / (LERadius * LERadius)))
Dim ExposedCircumference As Double
ExposedCircumference = ExposedAngle * LERadius
Dim DeltaS As Double
DeltaS = NylonLength / Form1.NumNylonSeg
NumExposedSegOnLECircle = CInt(ExposedCircumference / DeltaS)
Dim DeltaTheta As Double
DeltaTheta = ExposedAngle / NumExposedSegOnLECircle
Dim ExposedStartAngle As Double
ExposedStartAngle = Math.Atan2(RiseToStartOfMembrane, RunToStartOfMembrane)
For I As Int32 = 0 To NumExposedSegOnLECircle Step 1
    Dim lAngle As Double = ExposedStartAngle + (I * DeltaTheta)
    P(I) = LECircleCenterX + (LERadius * Math.Cos(lAngle))
    Q(I) = LECircleCenterY + (LERadius * Math.Sin(lAngle))
Next I
'
' Step #6: Write vertices on the membrane's upper/outer surface, from the outside
'          of the nylon membrane at the departure point to the outside of the
'          nylon membrane immediately forward of its aft edge.  The trailing edge
'          of the nylon membrane will not be represented as a square edge having
'          the nominal membrane thickness.  Instead, we will represent the
'          trailing edge as a sharp edge, being the mid-point of the membrane.
'          This final point is not written during this Step #6.  Instead, it will
'          be the final point written in the next step, when we write the points
'          along the bottom surface.  Note that these particular points are
'          written to the GMesh file first to make it easier to keep track of the
'          segments on which we want to know the specific aerodynamic forces.
Filewriter.Write( _
   "//" & vbCrLf & _
   "// Nylon membrane's upper surface, departure point to aft edge" & vbCrLf)
FirstPtOnTopR = 1
LastPtOnTopR = FirstPtOnTopR - 1
For I As Int32 = 1 To NumOfFlyingSeg Step 1
    Dim J As Int32 = NumSegOnLECircle + I
    LastPtOnTopR = LastPtOnTopR + 1
    Filewriter.Write( _
        "Point(" & Trim(Str(LastPtOnTopR)) & ") = { " & _
        FormatNumber(Xoutside(J), 10) & ", " & _
        FormatNumber(Youtside(J), 10) & ", " & _
        "-WTHalfWidth, lcMembrane };" & vbCrLf)
Next I
'
' Step #7: Write vertices on the membrane's lower/inner surface, from the inside
'          of the nylon membrane at the departure point to the sharp point at the
'          midpoint of the nylon membrane at its aft edge.  These particular
'          points are written to the GMesh file second to make it easier to keep
'          track of the segments on which we want to know the specific
'          aerodynamic forces.
```

```
Filewriter.Write( _
    "//" & vbCrLf & _
    "// Nylon membrane's lower surface, departure point to aft edge" & vbCrLf)
FirstPtOnBotR = LastPtOnTopR + 1
LastPtOnBotR = FirstPtOnBotR - 1
For I As Int32 = 1 To (NumOfFlyingSeg + 1) Step 1
    Dim J As Int32 = NumSegOnLECircle + I
    LastPtOnBotR = LastPtOnBotR + 1
    If (I <> (NumOfFlyingSeg + 1)) Then
        Filewriter.Write( _
            "Point(" & Trim(Str(LastPtOnBotR)) & ") = { " & _
            FormatNumber(Xinside(J), 10) & ", " & _
            FormatNumber(Yinside(J), 10) & ", " & _
            "-WTHalfWidth, lcMembrane };" & vbCrLf)
    Else
        Filewriter.Write( _
            "Point(" & Trim(Str(LastPtOnBotR)) & ") = { " & _
            FormatNumber(0.5 * (Xinside(J) + Xoutside(J)), 10) & ", " & _
            FormatNumber(0.5 * (Yinside(J) + Youtside(J)), 10) & ", " & _
            "-WTHalfWidth, lcMembrane };" & vbCrLf)
    End If
Next I
'
' Step #8: Write vertices on the exposed part of the leading edge circle,
'          clockwise, starting from the point on the circle immediately after the
'          departure point and counting down to, and including, the "start of
'          membrane" point.
FirstPtOnLECircle = LastPtOnBotR + 1
LastPtOnLECircle = FirstPtOnLECircle - 1
Filewriter.Write( _
   "//" & vbCrLf & _
   "// Exposed points on the L.E. circle, clockwise" & vbCrLf)
For I As Int32 = (NumExposedSegOnLECircle - 1) To 0 Step -1
    LastPtOnLECircle = LastPtOnLECircle + 1
    Filewriter.Write( _
        "Point(" & Trim(Str(LastPtOnLECircle)) & ") = { " & _
        FormatNumber(P(I), 10) & ", " & _
        FormatNumber(Q(I), 10) & ", " & _
        "-WTHalfWidth, lcMembrane };" & vbCrLf)
Next I
'
' Step #9: Write vertices on the covered part of the leading edge circle, on the
'          outside of the nylon membrane.  The first point is at the radial
'          location of the "start of membrane" point, but displaced outwards by
'          the thickness of the membrane.  The following points are listed
'          clockwise around the bottom and then the front side of the leading
'          edge circle.  The last point listed is the one immediately forward of
'          the departure point.
Filewriter.Write( _
   "//" & vbCrLf & _
   "// Covered points on the L.E. circle, continued clockwise" & vbCrLf)
For I As Int32 = 1 To NumSegOnLECircle Step 1
    LastPtOnLECircle = LastPtOnLECircle + 1
    Filewriter.Write( _
        "Point(" & Trim(Str(LastPtOnLECircle)) & ") = { " & _
        FormatNumber(Xoutside(I), 10) & ", " & _
        FormatNumber(Youtside(I), 10) & ", " & _
        "-WTHalfWidth, lcMembrane };" & vbCrLf)
```

```
Next I
'
' Step #10: Lines along nylon membrane's upper/outer surface, from the departure
'           point to the aft edge.  At the last segment, the upper surface
'           slopes down to meet the lower surface at a sharp point.
Dim FromPoint As Int32
Dim ToPoint As Int32
Filewriter.Write( _
    "//" & vbCrLf & _
    "// Lines along nylon membrane's upper surface, from LE to TE" & vbCrLf)
FirstLnAlngTopR = LastPtOnLECircle + 1
LastLnAlngTopR = FirstLnAlngTopR - 1
For I As Int32 = 1 To NumOfFlyingSeg Step 1
    FromPoint = FirstPtOnTopR + I - 1
    If (I < NumOfFlyingSeg) Then
        ToPoint = FirstPtOnTopR + I
    Else
        ToPoint = LastPtOnBotR
    End If
    LastLnAlngTopR = LastLnAlngTopR + 1
    Filewriter.Write( _
        "Line(" & Trim(Str(LastLnAlngTopR)) & _
        ") = {" & Trim(Str(FromPoint)) & _
        ", " & Trim(Str(ToPoint)) & "};" & vbCrLf)
Next I
'
' Step #11: Lines along nylon membrane's lower/inner surface, from the departure
'           point to the aft edge.
Filewriter.Write( _
    "//" & vbCrLf & _
    "// Lines along membrane's bottom surface, from LE to TE" & vbCrLf)
FirstLnAlngBotR = LastLnAlngTopR + 1
LastLnAlngBotR = FirstLnAlngBotR - 1
For I As Int32 = 1 To NumOfFlyingSeg Step 1
    FromPoint = FirstPtOnBotR + I - 1
    ToPoint = FirstPtOnBotR + I
    LastLnAlngBotR = LastLnAlngBotR + 1
    Filewriter.Write( _
        "Line(" & Trim(Str(LastLnAlngBotR)) & _
        ") = {" & Trim(Str(FromPoint)) & _
        ", " & Trim(Str(ToPoint)) & "};" & vbCrLf)
Next I
'
' Step #12: Lines around the leading edge circle, clockwise
Filewriter.Write( _
    "//" & vbCrLf & _
    "// Lines around the leading edge circle, clockwise" & vbCrLf)
FirstLnAroundLE = LastLnAlngBotR + 1
LastLnAroundLE = FirstLnAroundLE - 1
For I As Int32 = 0 To (NumSegOnLECircle + NumExposedSegOnLECircle) Step 1
    If (I = 0) Then
        FromPoint = FirstPtOnBotR
    Else
        FromPoint = FirstPtOnLECircle + I - 1
    End If
    If (I <> (NumSegOnLECircle + NumExposedSegOnLECircle)) Then
        ToPoint = FirstPtOnLECircle + I
    Else
```

```
            ToPoint = FirstPtOnTopR
        End If
        LastLnAroundLE = LastLnAroundLE + 1
        Filewriter.Write( _
            "Line(" & Trim(Str(LastLnAroundLE)) & _
            ") = {" & Trim(Str(FromPoint)) & _
            ", " & Trim(Str(ToPoint)) & "};" & vbCrLf)
    Next I
    '
    ' Step #13: Line Loop around the entire section, clockwise
    Filewriter.Write( _
        "//" & vbCrLf & _
        "// Line Loop around the entire section, clockwise" & vbCrLf)
    MemLineLoop = LastLnAroundLE + 1
    Dim NumbersAcrossPage As Int32 = 9
    Filewriter.Write("Line Loop(" & Trim(Str(MemLineLoop)) & ") = {")
    ' List Lines on upper/outer surface first, from LE to TE
    For I As Int32 = 1 To NumOfFlyingSeg Step 1
        Dim LineIndex As Int32 = FirstLnAlngTopR + I - 1
        If (NumbersAcrossPage > 8) Then
            Filewriter.Write(vbCrLf & "     " & Trim(Str(LineIndex)) & ",")
            NumbersAcrossPage = 1
        Else
            Filewriter.Write(" " & Trim(Str(LineIndex)) & ",")
            NumbersAcrossPage = NumbersAcrossPage + 1
        End If
    Next I
    ' List Lines on lower/inner surface, from TE to LE, reversed in sign
    For I As Int32 = 1 To NumOfFlyingSeg Step 1
        Dim LineIndex As Int32 = LastLnAlngBotR + 1 - I
        If (NumbersAcrossPage > 8) Then
            Filewriter.Write(vbCrLf & "     " & Trim(Str(-LineIndex)) & ",")
            NumbersAcrossPage = 1
        Else
            Filewriter.Write(" " & Trim(Str(-LineIndex)) & ",")
            NumbersAcrossPage = NumbersAcrossPage + 1
        End If
    Next I
    ' List Lines around leading edge tube
    For I As Int32 = 1 To (NumSegOnLECircle + NumExposedSegOnLECircle + 1) Step 1
        Dim LineIndex As Int32 = FirstLnAroundLE + I - 1
        If (I = (NumSegOnLECircle + NumExposedSegOnLECircle + 1)) Then
            Filewriter.Write(" " & Trim(Str(LineIndex)) & "};" & vbCrLf)
        Else
            If (NumbersAcrossPage > 8) Then
                Filewriter.Write(vbCrLf & "     " & Trim(Str(LineIndex)) & ",")
                NumbersAcrossPage = 1
            Else
                Filewriter.Write(" " & Trim(Str(LineIndex)) & ",")
                NumbersAcrossPage = NumbersAcrossPage + 1
            End If
        End If
    Next I
    '
    ' Step #14: Points at the corners of the wind tunnel
    Filewriter.Write( _
         "//" & vbCrLf & _
         "// Points at the corners of the wind tunnel" & vbCrLf)
```

```
FirstPtOnWTR = MemLineLoop + 1
Filewriter.Write("Point(" & Trim(Str(FirstPtOnWTR)) & ") = " & _
    "{-WTDistanceAhead, WTDistanceAbove, -WTHalfWidth, lcWT};" & vbCrLf)
Filewriter.Write("Point(" & Trim(Str(FirstPtOnWTR + 1)) & ") = " & _
    "{WTDistanceAstern, WTDistanceAbove, -WTHalfWidth, lcWT};" & vbCrLf)
Filewriter.Write("Point(" & Trim(Str(FirstPtOnWTR + 2)) & ") = " & _
    "{WTDistanceAstern, -WTDistanceBelow, -WTHalfWidth, lcWT};" & vbCrLf)
Filewriter.Write("Point(" & Trim(Str(FirstPtOnWTR + 3)) & ") = " & _
    "{-WTDistanceAhead, -WTDistanceBelow, -WTHalfWidth, lcWT};" & vbCrLf)
'
' Step #15: Lines along the edges of the wind tunnel, clockwise
Filewriter.Write( _
     "//" & vbCrLf & _
     "// Lines along the edges of the wind tunnel, clockwise" & vbCrLf)
FirstLnAlngWTR = MemLineLoop + 1
Filewriter.Write("Line(" & Trim(Str(FirstLnAlngWTR)) & ") = {" & _
    Trim(Str(FirstPtOnWTR)) & ", " & _
    Trim(Str(FirstPtOnWTR + 1)) & "};" & vbCrLf)
Filewriter.Write("Line(" & Trim(Str(FirstLnAlngWTR + 1)) & ") = {" & _
    Trim(Str(FirstPtOnWTR + 1)) & ", " & _
    Trim(Str(FirstPtOnWTR + 2)) & "};" & vbCrLf)
Filewriter.Write("Line(" & Trim(Str(FirstLnAlngWTR + 2)) & ") = {" & _
    Trim(Str(FirstPtOnWTR + 2)) & ", " & _
    Trim(Str(FirstPtOnWTR + 3)) & "};" & vbCrLf)
Filewriter.Write("Line(" & Trim(Str(FirstLnAlngWTR + 3)) & ") = {" & _
    Trim(Str(FirstPtOnWTR + 3)) & ", " & _
    Trim(Str(FirstPtOnWTR)) & "};" & vbCrLf)
'
' Step #16: Line Loop around the wind tunnel, directed outwards
Filewriter.Write( _
     "//" & vbCrLf & _
     "// Line Loop around the wind tunnel, directed outwards" & vbCrLf)
WTLineLoop = FirstLnAlngWTR + 4
Filewriter.Write("Line Loop(" & Trim(Str(WTLineLoop)) & ") = {" & _
     Trim(Str(FirstLnAlngWTR)) & ", " & _
     Trim(Str(FirstLnAlngWTR + 1)) & ", " & _
     Trim(Str(FirstLnAlngWTR + 2)) & ", " & _
     Trim(Str(FirstLnAlngWTR + 3)) & "};" & vbCrLf)
'
' Step #17: Plane Surface on the wind tunnel, right side, directed outwards
Filewriter.Write( _
     "//" & vbCrLf & _
     "// Plane Surface on the wind tunnel, right side," & vbCrLf & _
     "// excluding the hole left by the membrane." & vbCrLf)
WTSurface = WTLineLoop + 1
Filewriter.Write("Plane Surface(" & Trim(Str(WTSurface)) & ") = {" & _
    Trim(Str(WTLineLoop)) & ", " & Trim(Str(MemLineLoop)) & "};" & vbCrLf)

'////////////////////////////////////////////////////////////////////////////////
'////////////////////////////////////////////////////////////////////////////////
'// Extrusion of the right-hand side into the left-hand side ////////////////////
''////////////////////////////////////////////////////////////////////////////////
'////////////////////////////////////////////////////////////////////////////////

' Step #18: Extrude the Plane Surface of the wind tunnel in the Z-direction
Filewriter.Write( _
    "//" & vbCrLf & _
    "// Extrude Plane Surface of the wind tunnel in the Z-direction" & vbCrLf)
```

```
Filewriter.Write("NewWT[] = " & _
    "Extrude { 0 , 0 , WTWidth } {" & vbCrLf & _
    "    Surface{" & Trim(Str(WTSurface)) & "};" & vbCrLf & _
    "    Layers{1};" & vbCrLf & _
    "    Recombine; };" & vbCrLf)

'////////////////////////////////////////////////////////////////////////////
'////////////////////////////////////////////////////////////////////////////
'// Define Physical Surfaces ////////////////////////////////////////////////
'// It is necessary to review NewWT[] in GMesh to determine the order of the
'// segments.
'////////////////////////////////////////////////////////////////////////////
'////////////////////////////////////////////////////////////////////////////

' Step #19:  Define Physical Surfaces on the membrane for OpenFoam's use
Filewriter.Write( _
    "//" & vbCrLf & _
    "// Physical Surfaces on the membrane (for OpenFoam's use)" & vbCrLf & _
    "//" & vbCrLf & _
    "// Part #A -- All segments on LE circle" & vbCrLf)
Dim NumPlaneSurfaces As Int32
NumPlaneSurfaces = NumSegOnLECircle + NumExposedSegOnLECircle + 1
For I As Int32 = 1 To NumPlaneSurfaces Step 1
    Filewriter.Write( _
        "Physical Surface(""LETube." & Trim(Str(I)) & _
        """) = { NewWT[" & Trim(Str(5 + I)) & "] };" & vbCrLf)
Next I
Filewriter.Write( _
    "//" & vbCrLf & _
    "// Part #B -- Lower/inner surface of membrane" & vbCrLf)
Dim SurfaceNumber As Int32
For I As Int32 = 1 To NumOfFlyingSeg Step 1
    SurfaceNumber = NumSegOnLECircle + NumExposedSegOnLECircle + 1 + I
    Filewriter.Write( _
        "Physical Surface(""SegmentOnBot." & Trim(Str(I)) & _
        """) = { NewWT[" & Trim(Str(5 + SurfaceNumber)) & "] };" & vbCrLf)
Next I
Filewriter.Write( _
    "//" & vbCrLf & _
    "// Part #C -- Upper/outer surface of membrane" & vbCrLf)
For I As Int32 = 1 To NumOfFlyingSeg Step 1
    SurfaceNumber = _
        NumSegOnLECircle + NumExposedSegOnLECircle + _
        (2 * (NumOfFlyingSeg + 1)) - I
    Filewriter.Write( _
        "Physical Surface(""SegmentOnTop." & Trim(Str(I)) & _
        """) = { NewWT[" & Trim(Str(5 + SurfaceNumber)) & "] };" & vbCrLf)
Next I
'
' Step #20: Define Physical Surfaces on the wind tunnel for OpenFoam's use
Filewriter.Write( _
     "//" & vbCrLf & _
     "// Physical Surfaces on the wind tunnel (for OpenFoam's use)" & vbCrLf)
Filewriter.Write( _
    "Physical Surface(""LeftWall"") = { NewWT[0] };" & vbCrLf & _
    "Physical Surface(""Top"") = { NewWT[2] };" & vbCrLf & _
    "Physical Surface(""Outlet"") = { NewWT[3] };" & vbCrLf & _
    "Physical Surface(""Bottom"") = { NewWT[4] };" & vbCrLf & _
```

```
            "Physical Surface(""Inlet"") = { NewWT[5] };" & vbCrLf & _
            "Physical Surface(""RightWall"") = { " & _
            Trim(Str(WTSurface)) & " };" & vbCrLf)
        '
        ' Step #21: Define the Physical Volume for OpenFoam's use
        Filewriter.Write( _
             "//" & vbCrLf & _
             "// Define the Physical Volume for OpenFoam's use" & vbCrLf)
        Filewriter.Write("Physical Volume(""Internal"") = { NewWT[1] };" & vbCrLf)
        '
        ' Step #22: Conclude
        Filewriter.Close()
    End Sub

End Module
```

Listing of Module *WriteOpenFoamFunction*

```
Option Strict On
Option Explicit On

Public Module WriteOpenFoamFunction

    ' The subroutine in this module writes a text file which defines the forces which
    ' OpenFoam should print.  The default, as listed here, is:
    '
    ' At the end of every iteration, the total forces acting on the entire airfoil
    ' section are printed.
    '
    ' At the end of every 250th iteration, the following forces are printed:
    ' 1.    The total forces acting on the airfoil section.
    ' 2.    The forces acting on the top of each of the NumOfFlyingSeg free-flying
    '       segments, listed from the L.E. to the T.E.
    ' 3.    The forces acting on the bottom of each of the NumOfFlyingSeg free-flying
    '       segments, listed from the L.E. to the T.E.
    ' 4.    The total forces acting on the free-flying segments.  This is the sum of
    '       the forces in groups 2 and 3, and is a useful check for manual calculations.
    ' 5.    The total forces acting on the L.E. tube.  This is the difference between the
    '       forces in groups 1 and 4.  This is a necessary ingredient for the check of
    '       the overall balance of forces.

    ' The text in the file must be copied into the system/controlDict file in the
    ' OpenFoam case directory.

    ' The default file name is "OpenFoamFunction.txt".  If this program is run in Debug
    ' mode, this file will be created inside the Bin/Debug directory in the project's
    ' directory.

    '////////////////
    '// Data entry //
    '////////////////
    Public AirDensity As String
    Public OpenFoamTextFileName As String = "OpenFoamFunction.txt"

    Public Sub WriteOpenFoamFunction( _
        ByVal NumOfFlyingSeg As Int32, _
```

```
ByVal Altitude As Double)

' Step #1: Set the density for force calculations
If (Altitude = 0) Then
    AirDensity = "1.225"
Else
    If (Altitude = 15000) Then
        AirDensity = "0.1948"
    Else
        MsgBox("Unknown altitude supplied to WriteOpenFoamFunction")
        Exit Sub
    End If
End If

' Step #2: Open the output file
Dim Filewriter As System.IO.StreamWriter
Filewriter = New System.IO.StreamWriter(OpenFoamTextFileName)

' Step #3: Write header information to the output file
Filewriter.Write("//" & vbCrLf & _
    "// Function to print forces exerted on a flexible membrane." & vbCrLf & _
    "// The total force is printed every iteration." & vbCrLf & _
    "// The force on each segment is printed every 250 iterations." & vbCrLf & _
    "//" & vbCrLf & _
    "functions" & vbCrLf & _
    "{" & vbCrLf)

' Step #4: Write the total force at the end of every iteration
Filewriter.Write("  TotalForceOnMembrane" & vbCrLf & _
        "  {" & vbCrLf & _
        "    type                 forces;" & vbCrLf & _
        "    functionObjectLibs   ( ""libforces.so"" );" & vbCrLf & _
        "    patches              " & _
        "( ""SegmentOnTop.*"" ""SegmentOnBot.*"" ""LETube.*"" );" & vbCrLf & _
        "    rhoName              rhoInf;" & vbCrLf & _
        "    pName                p;" & vbCrLf & _
        "    UName                U;" & vbCrLf & _
        "    log                  true;" & vbCrLf & _
        "    rhoInf               " & AirDensity & ";" & vbCrLf & _
        "    CofR                 ( 0 0 0 );" & vbCrLf & _
        "    outputControl        timeStep;" & vbCrLf & _
        "    outputInterval       1;" & vbCrLf & _
        "  }" & vbCrLf)

' Step #5: Write forces exerted on segments on top of the membrane
For I As Int32 = 1 To NumOfFlyingSeg Step 1
    Filewriter.Write("  ForceOnTopSegment#" & Trim(Str(I)) & vbCrLf & _
        "  {" & vbCrLf & _
        "    type                 forces;" & vbCrLf & _
        "    functionObjectLibs   ( ""libforces.so"" );" & vbCrLf & _
        "    patches              ( ""SegmentOnTop." & _
        Trim(Str(I)) & """ );" & vbCrLf & _
        "    rhoName              rhoInf;" & vbCrLf & _
        "    pName                p;" & vbCrLf & _
        "    UName                U;" & vbCrLf & _
        "    log                  true;" & vbCrLf & _
        "    rhoInf               " & AirDensity & ";" & vbCrLf & _
        "    CofR                 ( 0 0 0 );" & vbCrLf & _
```

```
        "    outputControl        timeStep;" & vbCrLf & _
        "    outputInterval       250;" & vbCrLf & _
        "  }" & vbCrLf)
Next I

' Step #6: Write forces exerted on segments on bottom of the membrane
For I As Int32 = 1 To NumOfFlyingSeg Step 1
    Filewriter.Write("  ForceOnBottomSegment#" & Trim(Str(I)) & vbCrLf & _
        "  {" & vbCrLf & _
        "    type                 forces;" & vbCrLf & _
        "    functionObjectLibs   ( ""libforces.so"" );" & vbCrLf & _
        "    patches              ( ""SegmentOnBot." & _
        Trim(Str(I)) & """ );" & vbCrLf & _
        "    rhoName              rhoInf;" & vbCrLf & _
        "    pName                p;" & vbCrLf & _
        "    UName                U;" & vbCrLf & _
        "    log                  true;" & vbCrLf & _
        "    rhoInf               " & AirDensity & ";" & vbCrLf & _
        "    CofR                 ( 0 0 0 );" & vbCrLf & _
        "    outputControl        timeStep;" & vbCrLf & _
        "    outputInterval       250;" & vbCrLf & _
        "  }" & vbCrLf)
Next I

' Step #7: Write total of forces exerted on free-flying segments
Filewriter.Write("  TotalForceOnFreeFlyingSegments" & vbCrLf & _
        "  {" & vbCrLf & _
        "    type                 forces;" & vbCrLf & _
        "    functionObjectLibs   ( ""libforces.so"" );" & vbCrLf & _
        "    patches              " & _
        "( ""SegmentOnTop.*"" ""SegmentOnBot.*"" );" & vbCrLf & _
        "    rhoName              rhoInf;" & vbCrLf & _
        "    pName                p;" & vbCrLf & _
        "    UName                U;" & vbCrLf & _
        "    log                  true;" & vbCrLf & _
        "    rhoInf               " & AirDensity & ";" & vbCrLf & _
        "    CofR                 ( 0 0 0 );" & vbCrLf & _
        "    outputControl        timeStep;" & vbCrLf & _
        "    outputInterval       250;" & vbCrLf & _
        "  }" & vbCrLf)

' Step #8: Write total of forces exerted on the leading edge tube
Filewriter.Write("  TotalForceOnLETube" & vbCrLf & _
        "  {" & vbCrLf & _
        "    type                 forces;" & vbCrLf & _
        "    functionObjectLibs   ( ""libforces.so"" );" & vbCrLf & _
        "    patches              " & _
        "( ""LETube.*"" );" & vbCrLf & _
        "    rhoName              rhoInf;" & vbCrLf & _
        "    pName                p;" & vbCrLf & _
        "    UName                U;" & vbCrLf & _
        "    log                  true;" & vbCrLf & _
        "    rhoInf               " & AirDensity & ";" & vbCrLf & _
        "    CofR                 ( 0 0 0 );" & vbCrLf & _
        "    outputControl        timeStep;" & vbCrLf & _
        "    outputInterval       250;" & vbCrLf & _
        "  }" & vbCrLf)
```

```
        ' Step #9: Write trailer information to the output file
        Filewriter.Write("};" & vbCrLf)

        ' Step #10: Conclude
        Filewriter.Close()
    End Sub

End Module
```

Listing of Module *RenderMembrane*

```
Option Strict On
Option Explicit On

Public Module RenderMembrane

    ' The subroutine in this module draws the membrane on a given Bitmap, which the
    ' calling procedure pastes into a label control on the terminal screen.  The
    ' leading edge tube and reference chord line are rendered in black.  The part of the
    ' nylon sheet which lies on the surface of the leading edge tube is rendered in
    ' green.  the free-flying part of the nylon sheet is rendered in red.  The trailing
    ' edge string is rendered in blue.  The membrane is displayed at the specified angle
    ' of attack.

    ' Note that the locations of the points (hinges) are given as absolute X- and Y-
    ' co-ordinates, including their rotation by the angle of attack.  The angle of attack
    ' is passed to the subroutine for the sole purpose of determining the location of
    ' the trailing edge.

    ' The leading edge tube is discretized into 360 segments for plotting purposes only.
    ' It is plotted before the nylon membrane, so the green and red lines will appear on
    ' top in those places where the membrane lies on the surface of the leading edge
    ' tube.

    ' The input variables are:
    '    NumNylonSeg = the number of segments into which the nylon sheet is divided
    '    ChordLength = length of chord, meters
    '    LERadius = radius of the leading edge tube
    '    NumSegOnLECicle = number of segments on the leading edge circle
    '    NumOfFlyingSeg = number of free-flying nylon segments
    '    AngleAttackDeg = angle of attack, degrees
    '    XLE = X-co-ordinate of the leading edge, meters
    '    YLE = Y-co-ordinate of the leading edge, meters
    '    X(NumNylonSeg + 2) = X-co-ordinates of all hinges, meters
    '    Y(NumNylonSeg + 2) = Y-co-ordinates of the hinges, meters

    ' The output variable is:
    '    MembraneBitmap

    Public Sub RenderMembrane( _
        ByVal sender As System.Object, ByVal e As System.EventArgs, _
        ByRef MembraneBitmap As Bitmap, _
        ByVal NumNylonSeg As Int32, _
        ByVal ChordLength As Double, ByVal LERadius As Double, _
        ByVal NumSegOnLECircle As Int32, _
        ByVal NumOfFlyingSeg As Int32, _
```

```
ByVal AngleAttackDeg As Double, _
ByVal X() As Double, ByVal Y() As Double)
'
' Calculated values
Dim AngleAttackRad As Double = AngleAttackDeg * Math.PI / 180
Dim CosAlpha As Double = Math.Cos(AngleAttackRad)
Dim SinAlpha As Double = Math.Sin(AngleAttackRad)
Dim XTE As Double = ChordLength * CosAlpha
Dim YTE As Double = -ChordLength * SinAlpha
'
' Discretize the leading edge tube.  Use new vectors P() and Q() for the
' co-ordinates on its surface.
Dim P(360), Q(360) As Double
For I As Int32 = 1 To 360 Step 1
    ' Lay out 360 points around the L.E. circle
    P(I) = LERadius - (LERadius * Math.Cos((I - 1) * Math.PI / 180))
    Q(I) = LERadius * Math.Sin((I - 1) * Math.PI / 180)
    ' Rotate the circle to the correct angle of attack
    Dim Temp As Double
    Temp = (P(I) * CosAlpha) + (Q(I) * SinAlpha)
    Q(I) = (-P(I) * SinAlpha) + (Q(I) * CosAlpha)
    P(I) = Temp
Next I
'
' Find the extreme X- and Y-values to be plotted, in meters
Dim xMax As Single = -1.0E+20
Dim xMin As Single = 1.0E+20
Dim yMax As Single = -1.0E+20
Dim yMin As Single = 1.0E+20
' Test all points on the flexible surface
For I As Int32 = 1 To (NumNylonSeg + 2) Step 1
    If (X(I) > xMax) Then xMax = CSng(X(I))
    If (X(I) < xMin) Then xMin = CSng(X(I))
    If (Y(I) > yMax) Then yMax = CSng(Y(I))
    If (Y(I) < yMin) Then yMin = CSng(Y(I))
Next I
' Test all points on the leading edge circle
For I As Int32 = 1 To 360 Step 1
    If (P(I) > xMax) Then xMax = CSng(P(I))
    If (P(I) < xMin) Then xMin = CSng(P(I))
    If (Q(I) > yMax) Then yMax = CSng(Q(I))
    If (Q(I) < yMin) Then yMin = CSng(Q(I))
Next I
' Test the trailing edge
If (XTE > xMax) Then xMax = CSng(XTE)
If (XTE < xMin) Then xMin = CSng(XTE)
If (YTE > yMax) Then yMax = CSng(YTE)
If (YTE < yMin) Then yMin = CSng(YTE)
'
' Calculate the appropriate scaling factor, in pixels per meter.
' Leave a 5% margin all around the display.
Dim HorAvailPxls As Double = MembraneBitmap.Width
Dim VerAvailPxls As Double = MembraneBitmap.Height
Dim DeltaXReal As Double = (xMax - xMin) * 1.1
Dim DeltaYReal As Double = (yMax - yMin) * 1.1
Dim SFPixelsPerRealUnit As Double
If ((HorAvailPxls / DeltaXReal) < (VerAvailPxls / DeltaYReal)) Then
    SFPixelsPerRealUnit = HorAvailPxls / DeltaXReal
```

```
Else
    SFPixelsPerRealUnit = VerAvailPxls / DeltaYReal
End If
'
' Express the location and dimensions of the bitmap in meters
Dim bmLeftRealUnit As Double = xMin - (0.05 * (xMax - xMin))
Dim bmTopRealUnit As Double = yMax + (0.05 * (yMax - yMin))
Dim bmWidthRealUnit As Double = DeltaXReal
Dim bmHeightRealUnit As Double = DeltaYReal
'
' Draw the reference chord line
Dim g As Graphics = Graphics.FromImage(MembraneBitmap)
Dim ThisPen As New Drawing.Pen(Color.Black, 2)
Dim StartX As Single
Dim StartY As Single
Dim StopX As Single
Dim StopY As Single
StartX = CSng((0 - bmLeftRealUnit) * SFPixelsPerRealUnit)
StartY = CSng((bmTopRealUnit - 0) * SFPixelsPerRealUnit)
StopX = CSng((XTE - bmLeftRealUnit) * SFPixelsPerRealUnit)
StopY = CSng((bmTopRealUnit - YTE) * SFPixelsPerRealUnit)
g.DrawLine(ThisPen, StartX, StartY, StopX, StopY)
'
' Draw the leading edge circle
For I As Int32 = 1 To 359 Step 1
    StartX = CSng((P(I) - bmLeftRealUnit) * SFPixelsPerRealUnit)
    StartY = CSng((bmTopRealUnit - Q(I)) * SFPixelsPerRealUnit)
    StopX = CSng((P(I + 1) - bmLeftRealUnit) * SFPixelsPerRealUnit)
    StopY = CSng((bmTopRealUnit - Q(I + 1)) * SFPixelsPerRealUnit)
    g.DrawLine(ThisPen, StartX, StartY, StopX, StopY)
Next I
StartX = CSng((P(360) - bmLeftRealUnit) * SFPixelsPerRealUnit)
StartY = CSng((bmTopRealUnit - Q(360)) * SFPixelsPerRealUnit)
StopX = CSng((P(1) - bmLeftRealUnit) * SFPixelsPerRealUnit)
StopY = CSng((bmTopRealUnit - Q(1)) * SFPixelsPerRealUnit)
g.DrawLine(ThisPen, StartX, StartY, StopX, StopY)
'
' Draw the part of the membrane on the leading edge circle
ThisPen = New Drawing.Pen(Color.LimeGreen, 3)
For I As Int32 = 1 To NumSegOnLECircle Step 1
    StartX = CSng((X(I) - bmLeftRealUnit) * SFPixelsPerRealUnit)
    StartY = CSng((bmTopRealUnit - Y(I)) * SFPixelsPerRealUnit)
    StopX = CSng((X(I + 1) - bmLeftRealUnit) * SFPixelsPerRealUnit)
    StopY = CSng((bmTopRealUnit - Y(I + 1)) * SFPixelsPerRealUnit)
    g.DrawLine(ThisPen, StartX, StartY, StopX, StopY)
Next I
'
' Draw the free-flying part of the nylon membrane
ThisPen = New Drawing.Pen(Color.Red, 3)
For I As Int32 = 1 To NumOfFlyingSeg Step 1
    Dim J As Int32 = NumSegOnLECircle + I
    StartX = CSng((X(J) - bmLeftRealUnit) * SFPixelsPerRealUnit)
    StartY = CSng((bmTopRealUnit - Y(J)) * SFPixelsPerRealUnit)
    StopX = CSng((X(J + 1) - bmLeftRealUnit) * SFPixelsPerRealUnit)
    StopY = CSng((bmTopRealUnit - Y(J + 1)) * SFPixelsPerRealUnit)
    g.DrawLine(ThisPen, StartX, StartY, StopX, StopY)
Next I
'
```

```
        ' Draw the trailing edge string
        ThisPen = New Drawing.Pen(Color.SteelBlue, 3)
        StartX = CSng((X(NumNylonSeg + 1) - bmLeftRealUnit) * SFPixelsPerRealUnit)
        StartY = CSng((bmTopRealUnit - Y(NumNylonSeg + 1)) * SFPixelsPerRealUnit)
        StopX = CSng((X(NumNylonSeg + 2) - bmLeftRealUnit) * SFPixelsPerRealUnit)
        StopY = CSng((bmTopRealUnit - Y(NumNylonSeg + 2)) * SFPixelsPerRealUnit)
        g.DrawLine(ThisPen, StartX, StartY, StopX, StopY)
        g.Dispose()
    End Sub

End Module
```

Listing of Module *ExtractOpenFoamForces*

```
Option Strict On
Option Explicit On

Public Module ExtractOpenFoamForces

    ' The subroutine in this module reads a log file written during the OpenFoam job
    ' and extracts from the file the forces acting on the individual segments of the
    ' membrane.  The code shown here reads the forces assuming that they were written
    ' in the format used by the sister module WriteOpenFoamFunction.
    '
    ' Only the forces on the free-flying segments of the membrane are read from the
    ' file.  There are NumOfFlyingSeg segments on the upper/outer surface and a similar
    ' number of segments on the lower/inner surface.

    ' The original forces are returned in 12 vectors:
    '    FxPtop(NumOfFlyingSeg) for X-direction pressure forces on top-side segments
    '    FyPtop(NumOfFlyingSeg) for Y-direction pressure forces on top-side segments
    '    MzPtop(NumOfFlyingSeg) for Z-direction pressure moment on top-side segments
    '    FxVtop(NumOfFlyingSeg) for X-direction viscous forces on top-side segments
    '    FyVtop(NumOfFlyingSeg) for Y-direction viscous forces on top-side segments
    '    MzVtop(NumOfFlyingSeg) for Z-direction viscous moment on top-side segments
    '    FxPbot(NumOfFlyingSeg) for X-direction pressure forces on bottom-side segments
    '    FyPbot(NumOfFlyingSeg) for Y-direction pressure forces on bottom-side segments
    '    MzPbot(NumOfFlyingSeg) for Z-direction pressure moment on bottom-side segments
    '    FxVbot(NumOfFlyingSeg) for X-direction viscous forces on bottom-side segments
    '    FyVbot(NumOfFlyingSeg) for Y-direction viscous forces on bottom-side segments
    '    MzVbot(NumOfFlyingSeg) for Z-direction viscous moment on bottom-side segments
    ' The summed forces are also returned, in six vectors:
    '    FxTop(NumOfFlyingSeg) for X-direction total force on top-side segments
    '    FyTop(NumOfFlyingSeg) for Y-direction total force on top-side segments
    '    MzTop(NumOfFlyingSeg) for Z-direction total moment on top-side segments
    '    FxBot(NumOfFlyingSeg) for X-direction total force on bottom-side segments
    '    FyBot(NumOfFlyingSeg) for Y-direction total force on bottom-side segments
    '    MzBot(NumOfFlyingSeg) for Z-direction total moment on bottom-side segments
    ' In each vector, the segments are ordered from the L.E. to the T.E.

    ' The default file name for the OpenFoam log file is "ofLog.txt".  If this program
    ' is run in Debug mode, then the OpenFoam log file should be copied into the
    ' bin/Debug directory in the project's directory before execution.

    ' This subroutine parses the OpenFoam log file looking for the last iteration at
    ' which the forces on the individual segments were written.  In other words, it
```

```
' looks for the last iteration for which (2 * NumOfFlyingSeg) + 1 "forces output:"
' statements appear in the file.  It does not look for any indication of
' convergence.  Therefore, this subroutine can extract forces from an OpenFoam
' job which was not carried through to completion.  This is handy as it saves time
' when the membrane is still in its early, rough shape.

' Note that OpenFoam writes (2 * NumOfFlyingSeg) + 1 forces, where the
' 2 * NumOfFlyingSeg forces are those for the top and bottom surfaces.  But, OpenFoam
' also writes the total force acting on the membrane.  It happens to write the total
' force first.  Therefore, this subroutine ignores the first force in the list.  To
' be precise, it actually does save it, in the 0-index position of the vectors, where
' it is available if some other procedure needed to use it.

'////////////////
'// Data entry //
'////////////////
Public OpenFoamLogFileName As String = "ofLog.txt"

Public Sub ExtractOpenFoamForces( _
    ByVal NumNylonSeg As Int32, _
    ByVal NumSegOnLECircle As Int32, ByVal NumOfFlyingSeg As Int32, _
    ByRef FxPtop() As Double, ByRef FyPtop() As Double, ByRef MzPtop() As Double, _
    ByRef FxVtop() As Double, ByRef FyVtop() As Double, ByRef MzVtop() As Double, _
    ByRef FxPbot() As Double, ByRef FyPbot() As Double, ByRef MzPbot() As Double, _
    ByRef FxVbot() As Double, ByRef FyVbot() As Double, ByRef MzVbot() As Double, _
    ByRef FxTop() As Double, ByRef FyTop() As Double, ByRef MzTop() As Double, _
    ByRef FxBot() As Double, ByRef FyBot() As Double, ByRef MzBot() As Double, _
    ByRef LastTimeStep As Int32)
    '
    Dim FirstLine As String        ' The text of the first line read
    Dim IterationString As String  ' The text of one complete Time Step
    Dim LineString As String       ' The text of one line in the file
    Dim TimeStep As Int32          ' The number of the current Time Step
    Dim Locator As Int32           ' The position of certain text in a string
    Dim TempString As String       ' A temporary string for parsing
    '
    ' Step #1: Open the input file
    Dim Filereader As System.IO.StreamReader
    Filereader = New System.IO.StreamReader(OpenFoamLogFileName)
    '
    ' Step #2: Find the start of the first Time Step in the file
    Do
        If (Filereader.EndOfStream = True) Then
            MsgBox("Error: There are no Time Steps in the OpenFoam log file.")
            Return
            Exit Sub
        Else
            FirstLine = Filereader.ReadLine
            If (Strings.Left(FirstLine, 7) = "Time = ") Then
                Exit Do
            End If
        End If
    Loop
    '
    ' Step #3: Main loop to parse the file
    Do
        ' We are here with FirstLine being the start of a new Time Step, or empty
        If (Len(FirstLine) = 0) Then
```

```
    Exit Do
End If
TempString = Strings.Right(FirstLine, Len(FirstLine) - 7)
TimeStep = CInt(Val(TempString))
IterationString = ""
'Display the Time Step on the terminal screen
Form1.TextArea.Text = _
"Reading file ..." & vbCrLf & _
"    Time Step = " & Trim(Str(TimeStep))
Form1.TextArea.Refresh()
' Look for the start of the next Time Step, or the end of the file.
' Add lines to IterationString until that event occurs.  When that
' event occurs, save the first line of the next Time Step in
' variable FirstLine, so it is available for the next cycle.
Do
    If (Filereader.EndOfStream = True) Then
        FirstLine = ""
        Exit Do
    Else
        LineString = Filereader.ReadLine
        If (Strings.Left(LineString, 7) = "Time = ") Then
            FirstLine = LineString
            Exit Do
        Else
            IterationString = IterationString & " " & LineString
        End If
    End If
Loop
' We are here with the complete IterationString for Time Step.  If the
' IterationString is empty, then we have completed processing of the
' file.  If IterationString is not empty, then parse it.
If (Len(IterationString) = 0) Then
    Exit Do
End If
' Search for the first force report in the IterationString
Dim NumForceReports As Int32
Locator = InStr(IterationString, "forces output:")
If (Locator <> 0) Then
    ' There is at least one force report.  How many are there?
    NumForceReports = 1
    TempString = Strings.Right(IterationString, _
        Len(IterationString) - Locator - 13)
    Do
        Locator = InStr(TempString, "forces output:")
        If (Locator = 0) Then
            Exit Do
        Else
            NumForceReports = NumForceReports + 1
            TempString = Strings.Right(TempString, _
                Len(TempString) - Locator - 13)
        End If
    Loop
    ' If there are too many forces, then there is definitely an error.
    ' We will ignore the case when there is more than one force, but
    ' fewer than ((2 * NumOfFlyingSeg) + 3) forces, which could arise if
    ' the OpenFoam run was interrupted but later resumed.
    If (NumForceReports > ((2 * NumOfFlyingSeg) + 3)) Then
        MsgBox("Error: Too many forces in one iteration.")
```

```
        Return
        Exit Sub
    End If
    ' If there are just the right number of forces, then continue
    If (NumForceReports = ((2 * NumOfFlyingSeg) + 3)) Then
        ' Parse out the total forces acting on the membrane
        Locator = InStr(IterationString, "forces output:", _
            CompareMethod.Text)
        If (Locator = 0) Then
            MsgBox("Error: ... when parsing total forces on membrane.")
            Return
            Exit Sub
        End If
        IterationString = Strings.Right(IterationString, _
            Len(IterationString) - Locator - 13)
        Locator = InStr(IterationString, "viscous)((")
        If (Locator = 0) Then
            MsgBox("Error: Could not find FxP in the total forces.")
            Return
            Exit Sub
        End If
        IterationString = Strings.Right(IterationString, _
            Len(IterationString) - Locator - 9)
        Locator = InStr(IterationString, " ")
        If (Locator = 0) Then
            MsgBox("Error: Could not find FyP in the total forces.")
        End If
        FxPtop(0) = Val(Strings.Left(IterationString, Locator - 1))
        IterationString = Trim(Strings.Right(IterationString, _
            Len(IterationString) - Locator))
        Locator = InStr(IterationString, " ")
        If (Locator = 0) Then
            MsgBox("Error: Could not find FzP in the total forces.")
            Return
            Exit Sub
        End If
        FyPtop(0) = Val(Strings.Left(IterationString, Locator - 1))
        IterationString = Strings.Right(IterationString, _
            Len(IterationString) - Locator)
        Locator = InStr(IterationString, ") (")
        If (Locator = 0) Then
            MsgBox("Error: Could not find FxV in the total forces.")
            Return
            Exit Sub
        End If
        IterationString = Strings.Right(IterationString, _
            Len(IterationString) - Locator - 2)
        Locator = InStr(IterationString, " ")
        If (Locator = 0) Then
            MsgBox("Error: Could not find FyV in the total forces.")
            Return
            Exit Sub
        End If
        FxVtop(0) = Val(Strings.Left(IterationString, Locator - 1))
        IterationString = Strings.Right(IterationString, _
            Len(IterationString) - Locator)
        Locator = InStr(IterationString, " ")
        If (Locator = 0) Then
```

```
    MsgBox("Error: Could not find FzV in the total forces.")
    Return
    Exit Sub
End If
FyVtop(0) = Val(Strings.Left(IterationString, Locator - 1))
IterationString = Strings.Right(IterationString, _
    Len(IterationString) - Locator)
Locator = InStr(IterationString, "viscous)((")
If (Locator = 0) Then
    MsgBox("Error: Could not find MxP in the total moment.")
    Return
    Exit Sub
End If
IterationString = Strings.Right(IterationString, _
    Len(IterationString) - Locator - 9)
Locator = InStr(IterationString, " ")
If (Locator = 0) Then
    MsgBox("Error: Could not find MyP in the total moment.")
    Return
    Exit Sub
End If
IterationString = Strings.Right(IterationString, _
    Len(IterationString) - Locator)
Locator = InStr(IterationString, " ")
If (Locator = 0) Then
    MsgBox("Error: Could not find MzP in the total moment.")
    Return
    Exit Sub
End If
IterationString = Strings.Right(IterationString, _
    Len(IterationString) - Locator)
Locator = InStr(IterationString, ") (")
If (Locator = 0) Then
    MsgBox("Error: Could not find MxV in the total moment.")
    Return
    Exit Sub
End If
MzPtop(0) = Val(Strings.Left(IterationString, Locator - 3))
IterationString = Strings.Right(IterationString, _
    Len(IterationString) - Locator - 2)
Locator = InStr(IterationString, " ")
If (Locator = 0) Then
    MsgBox("Error: Could not find MyV in the total moment.")
    Return
    Exit Sub
End If
IterationString = Strings.Right(IterationString, _
    Len(IterationString) - Locator)
Locator = InStr(IterationString, " ")
If (Locator = 0) Then
    MsgBox("Error: Could not find MzV in the total moment.")
    Return
    Exit Sub
End If
IterationString = Strings.Right(IterationString, _
    Len(IterationString) - Locator)
Locator = InStr(IterationString, "))")
If (Locator = 0) Then
```

```
        MsgBox("Error: Could not find the end of the total moment.")
        Return
        Exit Sub
    End If
    MzVtop(0) = Val(Strings.Left(IterationString, Locator - 2))
    ' Parse out the forces on the top surface of the membrane
    For I As Int32 = 1 To NumOfFlyingSeg Step 1
        Locator = InStr(IterationString, "forces output:")
        If (Locator = 0) Then
            MsgBox("Error: ... when parsing forces on top surface.")
            Return
            Exit Sub
        End If
        IterationString = Strings.Right(IterationString, _
            Len(IterationString) - Locator - 13)
        Locator = InStr(IterationString, "viscous)((")
        If (Locator = 0) Then
            MsgBox("Error: Could not find FxP in a top segment.")
            Return
            Exit Sub
        End If
        IterationString = Strings.Right(IterationString, _
            Len(IterationString) - Locator - 9)
        Locator = InStr(IterationString, " ")
        If (Locator = 0) Then
            MsgBox("Error: Could not find FyP in a top segment.")
        End If
        FxPtop(I) = Val(Strings.Left(IterationString, Locator - 1))
        IterationString = Trim(Strings.Right(IterationString, _
            Len(IterationString) - Locator))
        Locator = InStr(IterationString, " ")
        If (Locator = 0) Then
            MsgBox("Error: Could not find FzP in a top segment.")
            Return
            Exit Sub
        End If
        FyPtop(I) = Val(Strings.Left(IterationString, Locator - 1))
        IterationString = Strings.Right(IterationString, _
            Len(IterationString) - Locator)
        Locator = InStr(IterationString, ") (")
        If (Locator = 0) Then
            MsgBox("Error: Could not find FxV in a top segment.")
            Return
            Exit Sub
        End If
        IterationString = Strings.Right(IterationString, _
            Len(IterationString) - Locator - 2)
        Locator = InStr(IterationString, " ")
        If (Locator = 0) Then
            MsgBox("Error: Could not find FyV in a top segment.")
            Return
            Exit Sub
        End If
        FxVtop(I) = Val(Strings.Left(IterationString, Locator - 1))
        IterationString = Strings.Right(IterationString, _
            Len(IterationString) - Locator)
        Locator = InStr(IterationString, " ")
        If (Locator = 0) Then
```

```
    MsgBox("Error: Could not find FzV in a top segment.")
    Return
    Exit Sub
End If
FyVtop(I) = Val(Strings.Left(IterationString, Locator - 1))
IterationString = Strings.Right(IterationString, _
    Len(IterationString) - Locator)
Locator = InStr(IterationString, "viscous)((")
If (Locator = 0) Then
    MsgBox("Error: Could not find MxP in a top segment.")
    Return
    Exit Sub
End If
IterationString = Strings.Right(IterationString, _
    Len(IterationString) - Locator - 9)
Locator = InStr(IterationString, " ")
If (Locator = 0) Then
    MsgBox("Error: Could not find MyP in a top segment.")
    Return
    Exit Sub
End If
IterationString = Strings.Right(IterationString, _
    Len(IterationString) - Locator)
Locator = InStr(IterationString, " ")
If (Locator = 0) Then
    MsgBox("Error: Could not find MzP in a top segment.")
    Return
    Exit Sub
End If
IterationString = Strings.Right(IterationString, _
    Len(IterationString) - Locator)
Locator = InStr(IterationString, ") (")
If (Locator = 0) Then
    MsgBox("Error: Could not find MxV in a top segment.")
    Return
    Exit Sub
End If
MzPtop(I) = Val(Strings.Left(IterationString, Locator - 1))
IterationString = Strings.Right(IterationString, _
    Len(IterationString) - Locator - 2)
Locator = InStr(IterationString, " ")
If (Locator = 0) Then
    MsgBox("Error: Could not find MyV in a top segment.")
    Return
    Exit Sub
End If
IterationString = Strings.Right(IterationString, _
    Len(IterationString) - Locator)
Locator = InStr(IterationString, " ")
If (Locator = 0) Then
    MsgBox("Error: Could not find MzV in a top segment.")
    Return
    Exit Sub
End If
IterationString = Strings.Right(IterationString, _
    Len(IterationString) - Locator)
Locator = InStr(IterationString, "))")
If (Locator = 0) Then
```

```
            MsgBox("Error: Could not find the end of a top segment.")
            Return
            Exit Sub
        End If
        MzVtop(I) = Val(Strings.Left(IterationString, Locator - 1))
    Next I
    ' Parse out the forces on the bottom surface of the membrane
    For I As Int32 = 1 To NumOfFlyingSeg Step 1
        Locator = InStr(IterationString, "forces output:")
        If (Locator = 0) Then
            MsgBox("Error: ... when parsing forces on bottom surface.")
            Return
            Exit Sub
        End If
        IterationString = Strings.Right(IterationString, _
            Len(IterationString) - Locator - 13)
        Locator = InStr(IterationString, "viscous)((")
        If (Locator = 0) Then
            MsgBox("Error: Could not find FxP in a bottom segment.")
            Return
            Exit Sub
        End If
        IterationString = Strings.Right(IterationString, _
            Len(IterationString) - Locator - 9)
        Locator = InStr(IterationString, " ")
        If (Locator = 0) Then
            MsgBox("Error: Could not find FyP in a bottom segment.")
        End If
        FxPbot(I) = Val(Strings.Left(IterationString, Locator - 1))
        IterationString = Trim(Strings.Right(IterationString, _
            Len(IterationString) - Locator))
        Locator = InStr(IterationString, " ")
        If (Locator = 0) Then
            MsgBox("Error: Could not find FzP in a bottom segment.")
            Return
            Exit Sub
        End If
        FyPbot(I) = Val(Strings.Left(IterationString, Locator - 1))
        IterationString = Strings.Right(IterationString, _
            Len(IterationString) - Locator)
        Locator = InStr(IterationString, ") (")
        If (Locator = 0) Then
            MsgBox("Error: Could not find FxV in a bottom segment.")
            Return
            Exit Sub
        End If
        IterationString = Strings.Right(IterationString, _
            Len(IterationString) - Locator - 2)
        Locator = InStr(IterationString, " ")
        If (Locator = 0) Then
            MsgBox("Error: Could not find FyV in a bottom segment.")
            Return
            Exit Sub
        End If
        FxVbot(I) = Val(Strings.Left(IterationString, Locator - 1))
        IterationString = Strings.Right(IterationString, _
            Len(IterationString) - Locator)
        Locator = InStr(IterationString, " ")
```

```
If (Locator = 0) Then
    MsgBox("Error: Could not find FzV in a bottom segment.")
    Return
    Exit Sub
End If
FyVbot(I) = Val(Strings.Left(IterationString, Locator - 1))
IterationString = Strings.Right(IterationString, _
    Len(IterationString) - Locator)
Locator = InStr(IterationString, "viscous)((")
If (Locator = 0) Then
    MsgBox("Error: Could not find MxP in a bottom segment.")
    Return
    Exit Sub
End If
IterationString = Strings.Right(IterationString, _
    Len(IterationString) - Locator - 9)
Locator = InStr(IterationString, " ")
If (Locator = 0) Then
    MsgBox("Error: Could not find MyP in a bottom segment.")
    Return
    Exit Sub
End If
IterationString = Strings.Right(IterationString, _
    Len(IterationString) - Locator)
Locator = InStr(IterationString, " ")
If (Locator = 0) Then
    MsgBox("Error: Could not find MzP in a bottom segment.")
    Return
    Exit Sub
End If
IterationString = Strings.Right(IterationString, _
    Len(IterationString) - Locator)
Locator = InStr(IterationString, ") (")
If (Locator = 0) Then
    MsgBox("Error: Could not find MxV in a bottom segment.")
    Return
    Exit Sub
End If
MzPbot(I) = Val(Strings.Left(IterationString, Locator - 1))
IterationString = Strings.Right(IterationString, _
    Len(IterationString) - Locator - 2)
Locator = InStr(IterationString, " ")
If (Locator = 0) Then
    MsgBox("Error: Could not find MyV in a bottom segment.")
    Return
    Exit Sub
End If
IterationString = Strings.Right(IterationString, _
    Len(IterationString) - Locator)
Locator = InStr(IterationString, " ")
If (Locator = 0) Then
    MsgBox("Error: Could not find MzV in a bottom segment.")
    Return
    Exit Sub
End If
IterationString = Strings.Right(IterationString, _
    Len(IterationString) - Locator)
Locator = InStr(IterationString, "))")
```

```
                    If (Locator = 0) Then
                        MsgBox("Error: Could not find the end of a bottom segment.")
                        Return
                        Exit Sub
                    End If
                    MzVbot(I) = Val(Strings.Left(IterationString, Locator - 1))
                Next I
                LastTimeStep = TimeStep
            End If
        End If
    Loop
    '
    ' Step #4: Add up the pressure and viscous components of the forces and moments
    For I As Int32 = 1 To NumOfFlyingSeg Step 1
        FxTop(I) = FxPtop(I) + FxVtop(I)
        FyTop(I) = FyPtop(I) + FyVtop(I)
        MzTop(I) = MzPtop(I) + MzVtop(I)
        FxBot(I) = FxPbot(I) + FxVbot(I)
        FyBot(I) = FyPbot(I) + FyVbot(I)
        MzBot(I) = MzPbot(I) + MzVbot(I)
    Next I
End Sub

End Module
```

Listing of Module *RenderForces*

```
Option Strict On
Option Explicit On

Public Module RenderForces

    ' The subroutine in this module graphs the forces on the membrane.  It is invoked
    ' after the subroutine ExtractOpenFoamForces() has read the forces from the ofLog.txt
    ' file written by OpenFoam.  The forces are displayed in a plot area located on the
    ' terminal screen below the membrane's plot area.

    ' The subroutine assumes that NumOfFlyingSeg is the number of forces to be plotted
    ' and that these forces are the first NumOfFlyingSeg forces in the vectors FxTop(),
    ' FyTop(), FxBot() and FyBot().

    ' The subroutine assumes that the forces are references to OpenFoam's co-ordinate
    ' frame of reference.  It therefore un-rotates the forces so they can be plotted with
    ' the reference chord lying horizontal.

    ' The net force on each flying segment is plotted.  The net force combines the
    ' pressure and viscous forces on both the top and bottom surfaces.  Net forces which
    ' act upwards in the +Y-direction are shown in green.  Net forces which act downwards
    ' in the -Y-direction are shown in red.

    ' The horizontal axis for this plot is the reference chord, not the length along
    ' the surface of the membrane.  The forces on the top and bottom surfaces are plotted
    ' as vectors emanating from points equally spaced along the horizontal axis.  These
    ' starting points are not the mid-points of the membrane's segments, nor are they
    ' the mid-points of the membrane's segments projected onto the reference chord.
    ' Either of those alternatives would be better, but neither is possible.  The shape
```

```
' of the membrane will likely not be available when this plotting routine is called.
' The shape which was used for the OpenFoam run was calculated in the last iteration,
' but the computer may have been used for other things since then.

' The input variables are:
'    NumOfFlyingSeg = number of segments on the free-flying nylon membrane
'    FxTop(NumOfFlyingSeg) = X-direction force on top surface
'    FyTop(NumOfFlyingSeg) = Y-direction force on top surface
'    FxBot(NumOfFlyingSeg) = X-direction force on bottom surface
'    FyBot(NumOfFlyingSeg) = Y-direction force on bottom surface
'    ChordLength    } These are used to scale the forces and to rotate them
'    AngleAttackRad } from the wind tunnel's X-Y axes to the reference chord line.

' The output variable is:
'    ForceBitmap

' The subroutine uses an array to hold the Cartesian co-ordinates of the
' beginning and end of all force vectors w.r.t. the reference chord.  The array is
' F(NumOfFlyingSeg, 4), where the 4 numbers are Xstart, Ystart, Xend, Yend of each
' force vector.  After the end-points of the force vectors have been calculated,
' they are rotated into the horizontal position using the angle of attack.

' The force vectors are scaled so that the magnitude of the greatest force is set
' equal to one-quarter of the length of the reference chord.

Public Sub RenderForces( _
    ByVal sender As System.Object, _
    ByVal e As System.EventArgs, _
    ByRef ForceBitmap As Bitmap, _
    ByVal NumOfFlyingSeg As Int32, _
    ByVal ChordLength As Double, ByVal AngleAttackRad As Double, _
    ByVal FxTop() As Double, ByVal FyTop() As Double, _
    ByVal FxBot() As Double, ByVal FyBot() As Double)
    '
    ' Add the forces on the top and bottom surfaces
    Dim FxTot(NumOfFlyingSeg) As Double
    Dim FyTot(NumOfFlyingSeg) As Double
    For I As Int32 = 1 To NumOfFlyingSeg Step 1
        FxTot(I) = FxTop(I) + FxBot(I)
        FyTot(I) = FyTop(I) + FyBot(I)
    Next I
    '
    ' Search for the magnitude of the greatest force
    Dim MaxForce As Double = -1.0E+20
    For I As Int32 = 1 To NumOfFlyingSeg Step 1
        Dim Temp As Double
        Temp = Math.Sqrt((FxTot(I) * FxTot(I)) + (FyTot(I) * FyTot(I)))
        If (Temp > MaxForce) Then MaxForce = Temp
    Next I
    '
    ' Calculate the scaling factor to be applied to the forces
    Dim SFForcePerLength As Double
    SFForcePerLength = 0.25 * ChordLength / MaxForce
    '
    ' Un-rotate the forces by the angle of attack
    Dim SinAlpha As Double = Math.Sin(AngleAttackRad)
    Dim CosAlpha As Double = Math.Cos(AngleAttackRad)
    For I As Int32 = 1 To NumOfFlyingSeg Step 1
```

```
    Dim Temp As Double
    Temp = (FxTot(I) * CosAlpha) - (FyTot(I) * SinAlpha)
    FyTot(I) = (FxTot(I) * SinAlpha) + (FyTot(I) * CosAlpha)
    FxTot(I) = Temp
Next I
'
' Calculate and store the mid-points of the NumOfFlyingSeg segments
Dim F(NumOfFlyingSeg, 4) As Double
For I As Int32 = 1 To NumOfFlyingSeg Step 1
    F(I, 1) = ChordLength * (I - 0.5) / NumOfFlyingSeg
    F(I, 2) = 0
Next I
'
' Calculate and store the end-points of the NumNylonSeg forces.  The color
' code is assigned: green if the rotated Fy component is positive, red if the
' rotated Fy component is negative.
Dim ColorCodeTop(NumOfFlyingSeg) As String
Dim ColorCodeBot(NumOfFlyingSeg) As String
For I As Int32 = 1 To NumOfFlyingSeg Step 1
    If (FyTot(I) >= 0) Then
        F(I, 3) = F(I, 1) + (FxTot(I) * SFForcePerLength)
        F(I, 4) = F(I, 2) + (FyTot(I) * SFForcePerLength)
        ColorCodeTop(I) = "G"
    Else
        F(I, 3) = F(I, 1) - (FxTot(I) * SFForcePerLength)
        F(I, 4) = F(I, 2) - (FyTot(I) * SFForcePerLength)
        ColorCodeTop(I) = "R"
    End If
Next I
'
' Search the forces for the maximum and minimum X- and Y-values
Dim MaxX As Double = -1.0E+20
Dim MinX As Double = +1.0E+20
Dim MaxY As Double = -1.0E+20
Dim MinY As Double = +1.0E+20
For I As Int32 = 1 To NumOfFlyingSeg Step 1
    If (F(I, 1) > MaxX) Then MaxX = F(I, 1)
    If (F(I, 3) > MaxX) Then MaxX = F(I, 3)
    If (F(I, 1) < MinX) Then MinX = F(I, 1)
    If (F(I, 3) < MinX) Then MinX = F(I, 3)
    If (F(I, 2) > MaxY) Then MaxY = F(I, 2)
    If (F(I, 4) > MaxY) Then MaxY = F(I, 4)
    If (F(I, 2) < MinY) Then MinY = F(I, 2)
    If (F(I, 4) < MinY) Then MinY = F(I, 4)
Next I
'
' Include the leading and trailing edges in the search for extrema
If (0 > MaxX) Then MaxX = 0
If (0 < MinX) Then MinX = 0
If (ChordLength > MaxX) Then MaxX = ChordLength
If (ChordLength < MinX) Then MinX = ChordLength
If (0 > MaxY) Then MaxY = 0
If (0 < MinY) Then MinY = 0
'
' Calculate the appropriate scaling factor, in pixels per meter.
' Leave a 1% margin all around the display.
Dim HorAvailPxls As Double = ForceBitmap.Width
Dim VerAvailPxls As Double = ForceBitmap.Height
```

```
        Dim DeltaXMeters As Double = (MaxX - MinX) * 1.02
        Dim DeltaYMeters As Double = (MaxY - MinY) * 1.02
        Dim SFPixelsPerMeter As Double
        If ((HorAvailPxls / DeltaXMeters) < (VerAvailPxls / DeltaYMeters)) Then
            SFPixelsPerMeter = HorAvailPxls / DeltaXMeters
        Else
            SFPixelsPerMeter = VerAvailPxls / DeltaYMeters
        End If
        '
        ' Express the location and dimensions of the bitmap in meters
        Dim bmLeftMeters As Double = MinX - (0.01 * (MaxX - MinX))
        Dim bmTopMeters As Double = MaxY + (0.01 * (MaxY - MinY))
        Dim bmWidthMeters As Double = DeltaXMeters
        Dim bmHeightMeters As Double = DeltaYMeters
        '
        ' Draw the forces one-by-one starting from the leading edge
        Dim g As Graphics = Graphics.FromImage(ForceBitmap)
        Dim RedPen As New Drawing.Pen(Color.Red, 2)
        Dim GreenPen As New Drawing.Pen(Color.Green, 2)
        Dim StartX As Single
        Dim StartY As Single
        Dim StopX As Single
        Dim StopY As Single
        For I As Int32 = 1 To NumOfFlyingSeg Step 1
            StartX = CSng((F(I, 1) - bmLeftMeters) * SFPixelsPerMeter)
            StartY = CSng((bmTopMeters - F(I, 2)) * SFPixelsPerMeter)
            StopX = CSng((F(I, 3) - bmLeftMeters) * SFPixelsPerMeter)
            StopY = CSng((bmTopMeters - F(I, 4)) * SFPixelsPerMeter)
            If (ColorCodeTop(I) = "R") Then
                g.DrawLine(RedPen, StartX, StartY, StopX, StopY)
            Else
                g.DrawLine(GreenPen, StartX, StartY, StopX, StopY)
            End If
        Next I
        '
        ' Draw the reference chord line
        Dim BlackPen As New Drawing.Pen(Color.Black, 3)
        StartX = CSng((0 - bmLeftMeters) * SFPixelsPerMeter)
        StartY = CSng((bmTopMeters - 0) * SFPixelsPerMeter)
        StopX = CSng((ChordLength - bmLeftMeters) * SFPixelsPerMeter)
        StopY = CSng((bmTopMeters - 0) * SFPixelsPerMeter)
        g.DrawLine(BlackPen, StartX, StartY, StopX, StopY)
        g.Dispose()
    End Sub

End Module
```

Listing of Module *OneMarchAlongMembrane*

```
Option Strict On
Option Explicit On

Public Module OneMarchAlongMembrane

    ' The subroutine in this module makes one march along the membrane, from the
    ' departure point to the trailing edge of the airfoil section (as distinct from the
    ' aft edge of the free-flying membrane).  The membrane is subjected to forces which
    ' are supplied in vectors Fx() and Fy().  These are the components of the force, in
    ' Newtons per span-wise meter, at the mid-points of the segments.  The membrane is
    ' also subjected to moments which are supplied in vector Mz().  The subroutine
    ' ignores the angle of attack and carries out all calculations with the X-axis being
    ' the reference chord line.  Therefore, the supplied forces Fx() and Fy() must be the
    ' components parallel to and perpendicular to the reference chord line, respectively.

    ' A guess for the number of segments on the leading edge circle is supplied, in the
    ' variable GuessNumSegOnLECircle.  Using this guess, this subroutine calculates the
    ' co-ordinates of the points on the leading edge circle, from Hinge #1 at the "start
    ' of membrane" point to Hinge #GuessNumSegOnLECircle + 1 at the departure point.  The
    ' remainder of the nylon membrane is divided equally into segments having the desired
    ' length.  The number of segments in this free-flying part of the surface is returned
    ' in the variable NumOfFlyingSeg.

    ' Note that this subroutine re-calculates the co-ordinates of the points on the
    ' leading edge circle.  The calling routine does not need to make this calculation.

    ' Note also that this subroutine does not alter the forces and moments in the vectors
    ' Fx(), Fy() and Mz().  It uses whatever numbers are there and leaves the vectors
    ' intact for subsequent iterations.

    ' As the iterations progress, it is possible that the numbers of segments on the
    ' leading edge circle and on the trailing edge string will change.  One or more
    ' segments may be added to the free-flying part of the surface.  To improve the
    ' accuracy of the procedure, an assumption is made about the forces which will likely
    ' act on these new segments.  The force on segments added near the departure point
    ' is set equal to the force which OpenFoam calculated for the first free-flying
    ' segment.  The force on segments added near the trailing edge is set equal to the
    ' force which OpenFoam calculated for the last free-flying segment.

    ' The co-ordinates of the hinges are returned in the vectors X() and Y().  These
    ' vectors contain enough information to construct the entire airfoil section,
    ' including those segments on the leading edge circle and the end-points of the
    ' trailing edge string.  These details are explained in the following note.

    '////////////////////////////////////////////////////////////////////////////////
    ' Special handling of the LE circle and the TE string
    ' At all times,
    '    NumNylonSeg = the number of segments into which the nylon membrane is divided
    '    GuessNumSegOnLECircle = the number of nylon segments on the leading edge circle
    '    Fx() = forces which are parallel to the reference chord line, in N/m
    '    Fy() = forces which are perpendicular to the reference chord line, in N/m
    '    Mz() = moments in the spanwise direction, N
    '    X(NumNylonSeg + 2) = X-co-ordinates of the hinges, meters
    '    Y(NumNylonSeg + 2) = Y-co-ordinates of the hinges, meters
```

```
'    Note that X() and Y() include the hinges which lie on the leading edge circle
'
' The part of the nylon membrane which lies on the LE circle is comprised of:
'    Index of first segment = 1
'    Index of last segment = NumSegOnLECircle
'    Index of "left-most" hinge (the "start of membrane" point) = 1
'    Index of "right-most" hinge (the departure point) = NumSegOnLECircle + 1
'    The subroutine ignores the forces Fx() and Fy() and moment Mz() acting on these
'    segments.  The subroutine assumes that the co-ordinates X() and Y() of these
'    hinges are their physical co-ordinates (unrotated to the angle of attack) around
'    the leading edge circle.
'
' The free-flying part of the nylon membrane is comprised of:
'    Index of first segment = NumSegOnLECircle + 1
'    Index of last segment = NumNylonSeg
'    Index of left-most hinge (the departure point) = NumSegOnLECircle + 1
'    Index of right-most hinge (aft edge of nylon) = NumNylonSeg + 1
'    The forces Fx() and Fy() and moment Mz() acting on these segments are used in
'    the calculation.  The co-ordinates X() and Y() of the hinges are stated with
'    respect to the (unrotated to the angle of attack) reference chord line.
'
' The TE string is comprised of:
'    Index of TEstring = NumNylonSeg +21
'    Index of left-most hinge (aft edge of nylon) = NumNylonSeg + 1
'    Index of right-most hinge (trailing edge of section) = NumNylonSeg + 2
'    Fx() = Fy() = Mz() = 0 for these segments.  The co-ordinates X() and Y() of
'    these two hinges are stated with respect to the (unrotated to the angle of
'    attack) reference chord line.
'////////////////////////////////////////////////////////////////////////////////

' The input variables are:
'    MembraneBitmap = the bitmap to be displayed
'    NumNylonSeg = the number of segments into which the membrane is divided
'    GuessNumSegOnLECircle
'    ChordLength = length of chord, meters
'    NylonLength = length of membrane, meters
'    LERadius = radius of leading edge tube, meters
'    LenTEString = length of trailing edge string, meters
'    Fx(NumNylonSeg) = force on segment parallel to the reference chord line, N
'    Fy(NumNylonSeg) = force on segment perpendicular to the reference chord line, N

' The calculated quantities which are returned to the calling procedure are:
'    NumOfFlyingSeg
'    X(NumNylonSeg + 2) = X-co-ordinates of all hinges, meters
'    Y(NumNylonSeg + 2) = Y-co-ordinates of all hinges, meters
'    T(NumSegOnLECircle + 1 to NumNylonSeg) = Tension at the hinges, Newtons
'    ThetaRad(NumSegOnLECircle to NumNylonSeg + 1) = Angles at the hinges, radians
'    PsiRad(NumSegOnLECircle + 1 to NumNylonSeg) = Slopes of the segments, radians

Public Sub OneMarchAlongMembrane( _
    ByVal NumNylonSeg As Int32, ByVal GuessNumSegOnLECircle As Int32, _
    ByVal ChordLength As Double, ByVal NylonLength As Double, _
    ByVal LERadius As Double, ByVal LenTEString As Double, _
    ByVal Fx() As Double, ByVal Fy() As Double, ByVal Mz() As Double, _
    ByVal GuessTension As Double, ByVal GuessTheta0Rad As Double, _
    ByRef X() As Double, ByRef Y() As Double, _
    ByRef T() As Double, _
    ByRef ThetaRad() As Double, ByRef PsiRad() As Double, _
```

```
ByRef NumOfFlyingSeg As Int32)
'
' Calculate the lengths of the segments
Dim SegmentLength As Double = NylonLength / NumNylonSeg
'
' Calculate the co-ordinates of hinges on the leading edge circle
Dim DeltaAngle As Double = SegmentLength / LERadius
For I As Int32 = 1 To (GuessNumSegOnLECircle + 1) Step 1
    Dim Argument As Double = (-0.75 * Math.PI) + ((I - 1) * DeltaAngle)
    X(I) = LERadius * (1 - Math.Cos(Argument))
    Y(I) = LERadius * Math.Sin(Argument)
Next I
'
' Calculate the number of segments on the free-flying membrane
NumOfFlyingSeg = NumNylonSeg - GuessNumSegOnLECircle
'
' Define the first and last segments on the free-flying membrane
Dim FirstFlyingSeg As Int32 = GuessNumSegOnLECircle + 1
Dim LastFlyingSeg As Int32 = GuessNumSegOnLECircle + NumOfFlyingSeg
Dim SinThetaI As Double
Dim CosThetaI As Double
Dim SinThetaIplus1 As Double
Dim CosThetaIplus1 As Double
'
' Initialize the first segment
ThetaRad(FirstFlyingSeg) = GuessTheta0Rad
T(FirstFlyingSeg) = GuessTension
'
' Main loop to run through the free-flying segments, left to right
For I As Int32 = FirstFlyingSeg To LastFlyingSeg Step 1
    SinThetaI = Math.Sin(ThetaRad(I))
    CosThetaI = Math.Cos(ThetaRad(I))
    '
    ' Calculate the angle of the hinge on the right side - Equation (21A)
    ThetaRad(I + 1) = Math.Atan2( _
        Fy(I) - (T(I) * SinThetaI), _
        Fx(I) - (T(I) * CosThetaI))
    '
    ' Validate the Quadrant in which ThetaRad(I + 1) lies.
    Dim Temp As Double
    Temp = -Fx(I) + (T(I) * CosThetaI)
    If (Temp >= 0) Then
        ' ThetaRad(I + 1) is in Quadrants 1 or 4
        If (ThetaRad(I + 1) > (Math.PI / 2)) Then
            ThetaRad(I + 1) = ThetaRad(I + 1) - Math.PI
        End If
        If (ThetaRad(I + 1) < (-Math.PI / 2)) Then
            ThetaRad(I + 1) = ThetaRad(I + 1) + Math.PI
        End If
    Else
        ' ThetaRad(I + 1) is in Quadrants 2 or 3
        If ((ThetaRad(I + 1) >= 0) And (ThetaRad(I + 1) < (Math.PI / 2))) Then
            ThetaRad(I + 1) = ThetaRad(I + 1) - Math.PI
        End If
        If ((ThetaRad(I + 1) < 0) And (ThetaRad(I + 1) > (-Math.PI / 2))) Then
            ThetaRad(I + 1) = ThetaRad(I + 1) + Math.PI
        End If
    End If
```

```
            '
            ' Calculate the tension at the hinge on the right side - Equation (21B)
            SinThetaIplus1 = Math.Sin(ThetaRad(I + 1))
            CosThetaIplus1 = Math.Cos(ThetaRad(I + 1))
            T(I + 1) = (T(I) * Math.Cos(ThetaRad(I + 1) - ThetaRad(I))) - _
                ((Fx(I) * CosThetaIplus1) + (Fy(I) * SinThetaIplus1))
            '
            ' Calculate the slope of the segment - Equations (27), (31) and (32).
            ' This routine translates the aerodynamic moment to the segment's L.E.
            Dim lTranslatedMoment As Double
            Dim lAcoefficient, lBcoefficient As Double
            Dim lSinPsi1, lSinPsi2 As Double
            Dim lCosPsi1, lCosPsi2 As Double
            Dim lPsi1, lPsi2 As Double
            Dim lTestValue1, lTestValue2 As Double
            lTranslatedMoment = Mz(I) + (-X(I) * Fy(I)) + (Y(I) * Fx(I))
            lAcoefficient = _
                CosThetaIplus1 / SinThetaIplus1
            lBcoefficient = _
                -lTranslatedMoment / (SegmentLength * T(I + 1) * SinThetaIplus1)
            lSinPsi1 = _
                ((-lAcoefficient * lBcoefficient) + _
                Math.Sqrt(1 + (lAcoefficient ^ 2) + (lBcoefficient ^ 2))) / _
                (1 + (lAcoefficient ^ 2))
            lSinPsi2 = _
                ((-lAcoefficient * lBcoefficient) - _
                Math.Sqrt(1 + (lAcoefficient ^ 2) + (lBcoefficient ^ 2))) / _
                (1 + (lAcoefficient ^ 2))
            lPsi1 = Math.Asin(lSinPsi1)
            lPsi2 = Math.Asin(lSinPsi2)
            lCosPsi1 = Math.Cos(lPsi1)
            lCosPsi2 = Math.Cos(lPsi2)
            '
            ' Pick the value of psi which satisfies the original equation
            lTestValue1 = lCosPsi1 - (lAcoefficient * lSinPsi1) - lBcoefficient
            lTestValue2 = lCosPsi2 - (lAcoefficient * lSinPsi2) - lBcoefficient
            If (Math.Abs(lTestValue1) < Math.Abs(lTestValue2)) Then
                PsiRad(I) = lPsi1
            Else
                PsiRad(I) = lPsi2
            End If
            '
            ' Calculate the local co-ordinates of the right-side hinge
            X(I + 1) = X(I) + (SegmentLength * Math.Cos(PsiRad(I)))
            Y(I + 1) = Y(I) + (SegmentLength * Math.Sin(PsiRad(I)))
        Next I
        '
        ' Extend the final tension to the trailing edge of the airfoil section
        X(LastFlyingSeg + 2) = X(LastFlyingSeg + 1) + _
            (LenTEString * Math.Cos(ThetaRad(LastFlyingSeg + 1)))
        Y(LastFlyingSeg + 2) = Y(LastFlyingSeg + 1) + _
            (LenTEString * Math.Sin(ThetaRad(LastFlyingSeg + 1)))
    End Sub

End Module
```

Listing of Module *ConvergeToShape*

```
Option Strict On
Option Explicit On

Public Module ConvergeToShape

    ' The subroutine in this module makes multiple marches along the membrane, from left
    ' to right, until it homes in on the shape which is consistent with the given force
    ' distribution.  The individual marches are done by the subroutine
    ' OneMarchAlongMembrane in the module with the same name.  The subroutine in this
    ' module only manages the process of converging.

    ' The input variables are:
    '    MembraneBitmap = the bitmap to be displayed
    '    NumNylonSeg = the number of segments into which the nylon sheet is divided
    '    NumSegOnLECircle -- a guess is supplied
    '    GuessTension = guess for tension at the leading edge, N/m
    '    GuessTheta0Deg = guess for angle at leading edge, degrees
    '    ChordLength = length of chord, meters
    '    MembraneLength = length of membrane, meters
    '    LERadius = radius of leading edge tube, meters
    '    LenTEString = length of trailing edge string, meters
    '    XLE, YLE = co-ordinates of the leading edge
    '    Fx(NumOfFlyingSeg) = force on segment parallel to the chord, N/m
    '    Fy(NumOfFlyingSeg) = force on segment perpendicular to the chord, N/m

    ' The calculated quantities which are returned to the calling procedure are:
    '    NumSegOnLECircle } final values ...
    '    NumOfFlyingSeg   } ... are returned
    '    X(NumNylonSeg + 2) = X-co-ordinates of all hinges, meters
    '    Y(NumNylonSeg + 2) = Y-co-ordinates of all hinges, meters
    '    T(NumNylonSeg + 2) = Tension at the hinges, Newtons
    '    ThetaRad(NumNylonSeg + 2) = Tension angles at the hinges, radians
    '    PsiRad(NumNylonSeg + 1) = Slopes of the segments, radians

    ' Miscellaneous parameters needed for this procedure
    Public MaxNumIterations As Int32 = 10000     ' Limit for ThetaJ+1 bisection routine
    Public MaxAbsError As Double = 0.00000001    ' Limit for ThetaJ+1 bisection routine
    Public MaxTEError As Double = 0.000001       ' Limit for automatic convergence

    Public Sub ConvergeToShape( _
        ByVal NumNylonSeg As Int32, ByRef NumSegOnLECircle As Int32, _
        ByVal ChordLength As Double, ByVal NylonLength As Double, _
        ByVal LERadius As Double, ByVal LenTEString As Double, _
        ByVal Fx() As Double, ByVal Fy() As Double, ByVal Mz() As Double, _
        ByVal GuessTension As Double, ByVal GuessTheta0Deg As Double, _
        ByRef X() As Double, ByRef Y() As Double, _
        ByRef T() As Double, _
        ByRef ThetaRad() As Double, ByRef PsiRad() As Double, _
        ByRef NumOfFlyingSeg As Int32, _
        ByVal BasicTextAreaContents As String)
        '
        Dim OldTension As Double
        Dim NewTension As Double
        Dim OldTheta0Rad As Double
```

```
Dim NewTheta0Rad As Double
Dim OldTEXerror As Double
Dim OldTEYerror As Double
Dim OldTERadius As Double   ' Distance from membrane's L.E. to T.E.
Dim OldTEAngle As Double    ' Angle from membrane's L.E. to T.E.
Dim NewTEXerror As Double
Dim NewTEYerror As Double
Dim NewTERadius As Double
Dim NewTEAngle As Double
Dim TensionVariation As Double = 0.001 ' Start variations at 0.1%
Dim TVReduction As Double = 0.999      ' Reduce variation by 0.1% per iteration
'
' Execute the first run
OneMarchAlongMembrane.OneMarchAlongMembrane( _
    NumNylonSeg, NumSegOnLECircle, _
    ChordLength, NylonLength, _
    LERadius, LenTEString, _
    Fx, Fy, Mz, _
    GuessTension, GuessTheta0Deg * Math.PI / 180, _
    X, Y, T, _
    ThetaRad, PsiRad, _
    NumOfFlyingSeg)
'
' Identify the first hinge on the flexible material
Dim DepartPoint As Int32 = NumSegOnLECircle + 1
'
' Record the parameters from the first run
OldTension = T(DepartPoint)
OldTheta0Rad = ThetaRad(DepartPoint)
'
' Calculate the errors from the first run
OldTEXerror = X(NumNylonSeg + 2) - ChordLength
OldTEYerror = Y(NumNylonSeg + 2) - 0
OldTERadius = Math.Sqrt( _
    (X(NumNylonSeg + 2) * X(NumNylonSeg + 2)) + _
    (Y(NumNylonSeg + 2) * Y(NumNylonSeg + 2)))
OldTEAngle = Math.Atan2(Y(NumNylonSeg + 2), X(NumNylonSeg + 2))
'
' Increase the parameters by 1% in preparation for a second run
NewTension = OldTension * 1.01
' Joggle the angle away from zero so the multiplicative factor works
If (OldTheta0Rad = 0) Then
    NewTheta0Rad = 0.001
Else
    NewTheta0Rad = OldTheta0Rad * 1.01
End If
'
' Execute the second run
OneMarchAlongMembrane.OneMarchAlongMembrane( _
    NumNylonSeg, NumSegOnLECircle, _
    ChordLength, NylonLength, _
    LERadius, LenTEString, _
    Fx, Fy, Mz, _
    NewTension, NewTheta0Rad, _
    X, Y, T, _
    ThetaRad, PsiRad, _
    NumOfFlyingSeg)
'
```

```
' Calculate the errors from the second run
NewTEXerror = X(NumNylonSeg + 2) - ChordLength
NewTEYerror = Y(NumNylonSeg + 2) - 0
NewTERadius = Math.Sqrt( _
    (X(NumNylonSeg + 2) * X(NumNylonSeg + 2)) + _
    (Y(NumNylonSeg + 2) * Y(NumNylonSeg + 2)))
NewTEAngle = Math.Atan2(Y(NumNylonSeg + 2), X(NumNylonSeg + 2))
'
' Main loop
Do
    Dim TempNewTension As Double
    Dim TempNewLEAngle As Double
    Dim WhichVariable As Boolean = True
    '
    '////////////////////////////////////////////////////////////////////////
    '// CONVERGENCE PROCEDURE
    '// The convergence procedure alternates between changes to the leading edge
    '// Tension and changes to the leading edge Tension angle.  The Boolean
    '// variable controls which change is made during the current iteration.
    '////////////////////////////////////////////////////////////////////////
    '
    ' Part #A of convergence algorithm: change the assumed Tension at the LE
    If (WhichVariable = True) Then
        If (((OldTERadius - ChordLength) * (NewTERadius - ChordLength)) < 0) Then
            ' If both OldTERadius and NewTERadius bound the chord length, then
            ' use the average Tension for the next iteration.
            TempNewTension = (OldTension + NewTension) / 2
        Else
            If (NewTERadius > ChordLength) Then
                ' If both OldTERadius and NewTERadius are greater than the chord
                ' length, then decrease the Tension
                TempNewTension = NewTension / (1 + TensionVariation)
            Else
                ' If both OldTERadius and NewTERadius are less than the chord
                ' length, then increase the Tension.  The increase will be by
                ' only half as much as a corresponding decrease, in order to
                ' avoid bouncing back and forth.
                TempNewTension = NewTension * (1 + (0.5 * TensionVariation))
            End If
        End If
        ' Reduce the Tension Variation factor, even if it was not used during
        ' this iteration.
        TensionVariation = TensionVariation * TVReduction
    End If
    'Part #B of convergence algorithm: change the assumed angle at the LE
    If ((OldTEAngle * NewTEAngle) < 0) Then
        ' If both OldTEAngle and NewTEAngle bound zero, then use the average
        ' leading edge angle for the next iteration.
        TempNewLEAngle = (OldTheta0Rad + NewTheta0Rad) / 2
    Else
        ' Else, change Theta(1) by 1% of the error at the trailing edge
        TempNewLEAngle = NewTheta0Rad - (0.01 * NewTEAngle)
    End If
    '
    ' Switch variables for the next iteration
    WhichVariable = Not (WhichVariable)
    '
    ' Set up for the next iteration
```

```
OldTERadius = NewTERadius
OldTEAngle = NewTEAngle
OldTension = NewTension
NewTension = TempNewTension
OldTheta0Rad = NewTheta0Rad
NewTheta0Rad = TempNewLEAngle
'
' Execute the next iteration
OneMarchAlongMembrane.OneMarchAlongMembrane( _
    NumNylonSeg, NumSegOnLECircle, _
    ChordLength, NylonLength, _
    LERadius, LenTEString, _
    Fx, Fy, Mz, _
    NewTension, NewTheta0Rad, _
    X, Y, T, _
    ThetaRad, PsiRad, _
    NumOfFlyingSeg)
'
' Calculate errors from the new run
NewTEXerror = X(NumNylonSeg + 2) - ChordLength
NewTEYerror = Y(NumNylonSeg + 2) - 0
NewTERadius = Math.Sqrt( _
    (X(NumNylonSeg + 2) * X(NumNylonSeg + 2)) + _
    (Y(NumNylonSeg + 2) * Y(NumNylonSeg + 2)))
NewTEAngle = Math.Atan2(Y(NumNylonSeg + 2), X(NumNylonSeg + 2))
'
' Display the details of the current iteration
Form1.TextArea.Text = BasicTextAreaContents & vbCrLf & vbCrLf & _
    "Current status:" & vbCrLf & _
    "  T.E. X-error = " & FormatNumber(NewTEXerror, 9) & " m" & vbCrLf & _
    "  T.E. Y-error = " & FormatNumber(NewTEYerror, 9) & " m" & vbCrLf & _
    "  T.E. Angle = " & FormatNumber( _
        NewTEAngle * 180 / Math.PI, 9) & " deg" & vbCrLf & _
    "  Tension = " & FormatNumber(NewTension, 9) & " N/m" & vbCrLf & _
    "  L.E. angle = " & FormatNumber( _
        NewTheta0Rad * 180 / Math.PI, 9) & " deg"
Form1.TextArea.Refresh()
'
' Display the current shape
' Part A: Clear the graphics
Dim g As Graphics = Graphics.FromImage(Form1.MembraneBitmap)
g.Clear(Control.DefaultBackColor)
g.Dispose()
Form1.MembranePlotArea.BackgroundImage = Form1.MembraneBitmap
Form1.MembranePlotArea.Refresh()
' Part C: Paint the Bitmap
Dim e As System.EventArgs
RenderMembrane.RenderMembrane( _
    Form1.MembranePlotArea, e, Form1.MembraneBitmap, _
    NumNylonSeg, ChordLength, LERadius, _
    NumSegOnLECircle, NumOfFlyingSeg, _
    0, X, Y)
' Part D: Display the Bitmap
Form1.MembranePlotArea.BackgroundImage = Form1.MembraneBitmap
Form1.Refresh()
'
' Check if convergence has been reached
Dim TEError As Double
```

```
            TEError = Math.Sqrt( _
                (NewTEXerror * NewTEXerror) + (NewTEYerror * NewTEYerror))
            If (TEError < MaxTEError) Then
                ' Replace guess for tension with the value found
                GuessTension = NewTension
                Return
                Exit Sub
            End If
            '
            ' Wait 50ms between iterations
            Threading.Thread.Sleep(50)
            ' Give other processes a chance
            Application.DoEvents()
            '
            ' Terminate if the Halt button has been clicked
            If (Form1.AutoOn = False) Then
                Exit Do
            End If
        Loop
    End Sub

End Module
```

Appendix "D"

Listing of "Membrane.geo.txt" generated by Module *WriteGMeshFile*

```
// Shape of kite membrane in a 2D airflow
//   Chord length (m) = 1.0254
//   Nylon length (m) = .98
//   L.E. radius (m) = .0127
Mesh.RandomFactor = 1e-11;
Geometry.AutoCoherence = 1;
Geometry.HighlightOrphans = 1;
Geometry.MatchGeomAndMesh = 1;
Geometry.SnapX = 0;
Geometry.SnapY = 0;
Geometry.SnapZ = 0;
Geometry.Tolerance = 1e-15;
//
// Parameters
WTDistanceAhead = 3.0762;
WTDistanceAstern = 4.1016;
WTDistanceAbove = 3.0762;
WTDistanceBelow = 3.5889;
lcWT = .1;
WTWidth = .001;
WTHalfWidth = .0005;
MembraneThickness = .001;
Membrane_NPS = 2;
lcMembrane = .00098;
//
// Nylon membrane's upper surface, departure point to aft edge
Point(1) = { 0.0100215242, 0.0119064004, -WTHalfWidth, lcMembrane };
Point(2) = { 0.0119477791, 0.0122751820, -WTHalfWidth, lcMembrane };
Point(3) = { 0.0138744893, 0.0126415769, -WTHalfWidth, lcMembrane };
Point(4) = { 0.0158016521, 0.0130055844, -WTHalfWidth, lcMembrane };
Point(5) = { 0.0177292643, 0.0133672041, -WTHalfWidth, lcMembrane };
```

For the sake of brevity, I have excluded the lines which define Points #6 through 472.

```
Point(473) = { 0.9177932852, -0.0815645835, -WTHalfWidth, lcMembrane };
Point(474) = { 0.9196030398, -0.0823203917, -WTHalfWidth, lcMembrane };
Point(475) = { 0.9214118566, -0.0830784414, -WTHalfWidth, lcMembrane };
Point(476) = { 0.9232197328, -0.0838387316, -WTHalfWidth, lcMembrane };
Point(477) = { 0.9250266657, -0.0846012610, -WTHalfWidth, lcMembrane };
//
// Nylon membrane's lower surface, departure point to aft edge
Point(478) = { 0.0102095597, 0.0109242381, -WTHalfWidth, lcMembrane };
Point(479) = { 0.0121345976, 0.0112927876, -WTHalfWidth, lcMembrane };
Point(480) = { 0.0140600906, 0.0116589517, -WTHalfWidth, lcMembrane };
Point(481) = { 0.0159860357, 0.0120227301, -WTHalfWidth, lcMembrane };
Point(482) = { 0.0179124301, 0.0123841221, -WTHalfWidth, lcMembrane };
```

For the sake of brevity, I have excluded the lines which define Points #482 through 950.

```
Point(951) = { 0.9192165244, -0.0832426746, -WTHalfWidth, lcMembrane };
Point(952) = { 0.9210241988, -0.0840002448, -WTHalfWidth, lcMembrane };
Point(953) = { 0.9228309333, -0.0847600540, -WTHalfWidth, lcMembrane };
Point(954) = { 0.9246367251, -0.0855221010, -WTHalfWidth, lcMembrane };
Point(955) = { 0.9264415713, -0.0862863846, -WTHalfWidth, lcMembrane };
//
// Exposed points on the L.E. circle, clockwise
Point(956) = { 0.0121884564, 0.0111454154, -WTHalfWidth, lcMembrane };
Point(957) = { 0.0141776012, 0.0110545600, -WTHalfWidth, lcMembrane };
Point(958) = { 0.0161280955, 0.0106539054, -WTHalfWidth, lcMembrane };
Point(959) = { 0.0179919905, 0.0099533008, -WTHalfWidth, lcMembrane };
Point(960) = { 0.0197234669, 0.0089699689, -WTHalfWidth, lcMembrane };
Point(961) = { 0.0212799600, 0.0077280828, -WTHalfWidth, lcMembrane };
Point(962) = { 0.0226232069, 0.0062581715, -WTHalfWidth, lcMembrane };
Point(963) = { 0.0237201871, 0.0045963694, -WTHalfWidth, lcMembrane };
Point(964) = { 0.0245439336, 0.0027835281, -WTHalfWidth, lcMembrane };
Point(965) = { 0.0250741966, 0.0008642124, -WTHalfWidth, lcMembrane };
Point(966) = { 0.0252979408, -0.0011143958, -WTHalfWidth, lcMembrane };
Point(967) = { 0.0252096657, -0.0031036568, -WTHalfWidth, lcMembrane };
Point(968) = { 0.0248115416, -0.0050546691, -WTHalfWidth, lcMembrane };
Point(969) = { 0.0241133554, -0.0069194714, -WTHalfWidth, lcMembrane };
Point(970) = { 0.0231322705, -0.0086522219, -WTHalfWidth, lcMembrane };
Point(971) = { 0.0218924045, -0.0102103246, -WTHalfWidth, lcMembrane };
Point(972) = { 0.0204242369, -0.0115554772, -WTHalfWidth, lcMembrane };
//
// Covered points on the L.E. circle, continued clockwise
Point(973) = { 0.0209761339, -0.0123893895, -WTHalfWidth, lcMembrane };
Point(974) = { 0.0191759156, -0.0135660221, -WTHalfWidth, lcMembrane };
Point(975) = { 0.0172137108, -0.0144463641, -WTHalfWidth, lcMembrane };
Point(976) = { 0.0151378941, -0.0150087122, -WTHalfWidth, lcMembrane };
Point(977) = { 0.0129996415, -0.0152392025, -WTHalfWidth, lcMembrane };
Point(978) = { 0.0108516680, -0.0151321528, -WTHalfWidth, lcMembrane };
Point(979) = { 0.0087469282, -0.0146902021, -WTHalfWidth, lcMembrane };
Point(980) = { 0.0067373110, -0.0139242461, -WTHalfWidth, lcMembrane };
Point(981) = { 0.0048723601, -0.0128531680, -WTHalfWidth, lcMembrane };
Point(982) = { 0.0031980528, -0.0115033735, -WTHalfWidth, lcMembrane };
Point(983) = { 0.0017556664, -0.0099081396, -WTHalfWidth, lcMembrane };
Point(984) = { 0.0005807603, -0.0081067941, -WTHalfWidth, lcMembrane };
Point(985) = { -0.0002976999, -0.0061437460, -WTHalfWidth, lcMembrane };
Point(986) = { -0.0008580574, -0.0040673912, -WTHalfWidth, lcMembrane };
Point(987) = { -0.0010864975, -0.0019289186, -WTHalfWidth, lcMembrane };
Point(988) = { -0.0009773883, 0.0002189514, -WTHalfWidth, lcMembrane };
Point(989) = { -0.0005334199, 0.0023232664, -WTHalfWidth, lcMembrane };
Point(990) = { 0.0002344627, 0.0043321483, -WTHalfWidth, lcMembrane };
Point(991) = { 0.0013073284, 0.0061960714, -WTHalfWidth, lcMembrane };
Point(992) = { 0.0026587275, 0.0078690837, -WTHalfWidth, lcMembrane };
Point(993) = { 0.0042553437, 0.0093099400, -WTHalfWidth, lcMembrane };
Point(994) = { 0.0060578149, 0.0104831184, -WTHalfWidth, lcMembrane };
Point(995) = { 0.0080217043, 0.0113596961, -WTHalfWidth, lcMembrane };
//
// Lines along nylon membrane's upper surface, from LE to TE
Line(996) = {1, 2};
Line(997) = {2, 3};
Line(998) = {3, 4};
```

```
Line(999) = {4, 5};
Line(1000) = {5, 6};
```

For the sake of brevity, I have excluded the lines which define Lines #1001 through 1467.

```
Line(1468) = {473, 474};
Line(1469) = {474, 475};
Line(1470) = {475, 476};
Line(1471) = {476, 477};
Line(1472) = {477, 955};
//
// Lines along membrane's bottom surface, from LE to TE
Line(1473) = {478, 479};
Line(1474) = {479, 480};
Line(1475) = {480, 481};
Line(1476) = {481, 482};
Line(1477) = {482, 483};
```

For the sake of brevity, I have excluded the lines which define Lines #1478 through 1944.

```
Line(1945) = {950, 951};
Line(1946) = {951, 952};
Line(1947) = {952, 953};
Line(1948) = {953, 954};
Line(1949) = {954, 955};
//
// Lines around the leading edge circle, clockwise
Line(1950) = {478, 956};
Line(1951) = {956, 957};
Line(1952) = {957, 958};
Line(1953) = {958, 959};
Line(1954) = {959, 960};
Line(1955) = {960, 961};
Line(1956) = {961, 962};
Line(1957) = {962, 963};
Line(1958) = {963, 964};
Line(1959) = {964, 965};
Line(1960) = {965, 966};
Line(1961) = {966, 967};
Line(1962) = {967, 968};
Line(1963) = {968, 969};
Line(1964) = {969, 970};
Line(1965) = {970, 971};
Line(1966) = {971, 972};
Line(1967) = {972, 973};
Line(1968) = {973, 974};
Line(1969) = {974, 975};
Line(1970) = {975, 976};
Line(1971) = {976, 977};
Line(1972) = {977, 978};
Line(1973) = {978, 979};
Line(1974) = {979, 980};
```

```
Line(1975) = {980, 981};
Line(1976) = {981, 982};
Line(1977) = {982, 983};
Line(1978) = {983, 984};
Line(1979) = {984, 985};
Line(1980) = {985, 986};
Line(1981) = {986, 987};
Line(1982) = {987, 988};
Line(1983) = {988, 989};
Line(1984) = {989, 990};
Line(1985) = {990, 991};
Line(1986) = {991, 992};
Line(1987) = {992, 993};
Line(1988) = {993, 994};
Line(1989) = {994, 995};
Line(1990) = {995, 1};
//
// Line Loop around the entire section, clockwise
Line Loop(1991) = {
     996, 997, 998, 999, 1000, 1001, 1002, 1003, 1004,
     1005, 1006, 1007, 1008, 1009, 1010, 1011, 1012, 1013,
     1014, 1015, 1016, 1017, 1018, 1019, 1020, 1021, 1022,
     1023, 1024, 1025, 1026, 1027, 1028, 1029, 1030, 1031,
     1032, 1033, 1034, 1035, 1036, 1037, 1038, 1039, 1040,
     1041, 1042, 1043, 1044, 1045, 1046, 1047, 1048, 1049,
     1050, 1051, 1052, 1053, 1054, 1055, 1056, 1057, 1058,
     1059, 1060, 1061, 1062, 1063, 1064, 1065, 1066, 1067,
     1068, 1069, 1070, 1071, 1072, 1073, 1074, 1075, 1076,
     1077, 1078, 1079, 1080, 1081, 1082, 1083, 1084, 1085,
     1086, 1087, 1088, 1089, 1090, 1091, 1092, 1093, 1094,
     1095, 1096, 1097, 1098, 1099, 1100, 1101, 1102, 1103,
     1104, 1105, 1106, 1107, 1108, 1109, 1110, 1111, 1112,
     1113, 1114, 1115, 1116, 1117, 1118, 1119, 1120, 1121,
     1122, 1123, 1124, 1125, 1126, 1127, 1128, 1129, 1130,
     1131, 1132, 1133, 1134, 1135, 1136, 1137, 1138, 1139,
     1140, 1141, 1142, 1143, 1144, 1145, 1146, 1147, 1148,
     1149, 1150, 1151, 1152, 1153, 1154, 1155, 1156, 1157,
     1158, 1159, 1160, 1161, 1162, 1163, 1164, 1165, 1166,
     1167, 1168, 1169, 1170, 1171, 1172, 1173, 1174, 1175,
     1176, 1177, 1178, 1179, 1180, 1181, 1182, 1183, 1184,
     1185, 1186, 1187, 1188, 1189, 1190, 1191, 1192, 1193,
     1194, 1195, 1196, 1197, 1198, 1199, 1200, 1201, 1202,
     1203, 1204, 1205, 1206, 1207, 1208, 1209, 1210, 1211,
     1212, 1213, 1214, 1215, 1216, 1217, 1218, 1219, 1220,
     1221, 1222, 1223, 1224, 1225, 1226, 1227, 1228, 1229,
     1230, 1231, 1232, 1233, 1234, 1235, 1236, 1237, 1238,
     1239, 1240, 1241, 1242, 1243, 1244, 1245, 1246, 1247,
     1248, 1249, 1250, 1251, 1252, 1253, 1254, 1255, 1256,
     1257, 1258, 1259, 1260, 1261, 1262, 1263, 1264, 1265,
     1266, 1267, 1268, 1269, 1270, 1271, 1272, 1273, 1274,
     1275, 1276, 1277, 1278, 1279, 1280, 1281, 1282, 1283,
     1284, 1285, 1286, 1287, 1288, 1289, 1290, 1291, 1292,
     1293, 1294, 1295, 1296, 1297, 1298, 1299, 1300, 1301,
     1302, 1303, 1304, 1305, 1306, 1307, 1308, 1309, 1310,
```

1311, 1312, 1313, 1314, 1315, 1316, 1317, 1318, 1319,
1320, 1321, 1322, 1323, 1324, 1325, 1326, 1327, 1328,
1329, 1330, 1331, 1332, 1333, 1334, 1335, 1336, 1337,
1338, 1339, 1340, 1341, 1342, 1343, 1344, 1345, 1346,
1347, 1348, 1349, 1350, 1351, 1352, 1353, 1354, 1355,
1356, 1357, 1358, 1359, 1360, 1361, 1362, 1363, 1364,
1365, 1366, 1367, 1368, 1369, 1370, 1371, 1372, 1373,
1374, 1375, 1376, 1377, 1378, 1379, 1380, 1381, 1382,
1383, 1384, 1385, 1386, 1387, 1388, 1389, 1390, 1391,
1392, 1393, 1394, 1395, 1396, 1397, 1398, 1399, 1400,
1401, 1402, 1403, 1404, 1405, 1406, 1407, 1408, 1409,
1410, 1411, 1412, 1413, 1414, 1415, 1416, 1417, 1418,
1419, 1420, 1421, 1422, 1423, 1424, 1425, 1426, 1427,
1428, 1429, 1430, 1431, 1432, 1433, 1434, 1435, 1436,
1437, 1438, 1439, 1440, 1441, 1442, 1443, 1444, 1445,
1446, 1447, 1448, 1449, 1450, 1451, 1452, 1453, 1454,
1455, 1456, 1457, 1458, 1459, 1460, 1461, 1462, 1463,
1464, 1465, 1466, 1467, 1468, 1469, 1470, 1471, 1472,
-1949, -1948, -1947, -1946, -1945, -1944, -1943, -1942, -1941,
-1940, -1939, -1938, -1937, -1936, -1935, -1934, -1933, -1932,
-1931, -1930, -1929, -1928, -1927, -1926, -1925, -1924, -1923,
-1922, -1921, -1920, -1919, -1918, -1917, -1916, -1915, -1914,
-1913, -1912, -1911, -1910, -1909, -1908, -1907, -1906, -1905,
-1904, -1903, -1902, -1901, -1900, -1899, -1898, -1897, -1896,
-1895, -1894, -1893, -1892, -1891, -1890, -1889, -1888, -1887,
-1886, -1885, -1884, -1883, -1882, -1881, -1880, -1879, -1878,
-1877, -1876, -1875, -1874, -1873, -1872, -1871, -1870, -1869,
-1868, -1867, -1866, -1865, -1864, -1863, -1862, -1861, -1860,
-1859, -1858, -1857, -1856, -1855, -1854, -1853, -1852, -1851,
-1850, -1849, -1848, -1847, -1846, -1845, -1844, -1843, -1842,
-1841, -1840, -1839, -1838, -1837, -1836, -1835, -1834, -1833,
-1832, -1831, -1830, -1829, -1828, -1827, -1826, -1825, -1824,
-1823, -1822, -1821, -1820, -1819, -1818, -1817, -1816, -1815,
-1814, -1813, -1812, -1811, -1810, -1809, -1808, -1807, -1806,
-1805, -1804, -1803, -1802, -1801, -1800, -1799, -1798, -1797,
-1796, -1795, -1794, -1793, -1792, -1791, -1790, -1789, -1788,
-1787, -1786, -1785, -1784, -1783, -1782, -1781, -1780, -1779,
-1778, -1777, -1776, -1775, -1774, -1773, -1772, -1771, -1770,
-1769, -1768, -1767, -1766, -1765, -1764, -1763, -1762, -1761,
-1760, -1759, -1758, -1757, -1756, -1755, -1754, -1753, -1752,
-1751, -1750, -1749, -1748, -1747, -1746, -1745, -1744, -1743,
-1742, -1741, -1740, -1739, -1738, -1737, -1736, -1735, -1734,
-1733, -1732, -1731, -1730, -1729, -1728, -1727, -1726, -1725,
-1724, -1723, -1722, -1721, -1720, -1719, -1718, -1717, -1716,
-1715, -1714, -1713, -1712, -1711, -1710, -1709, -1708, -1707,
-1706, -1705, -1704, -1703, -1702, -1701, -1700, -1699, -1698,
-1697, -1696, -1695, -1694, -1693, -1692, -1691, -1690, -1689,
-1688, -1687, -1686, -1685, -1684, -1683, -1682, -1681, -1680,
-1679, -1678, -1677, -1676, -1675, -1674, -1673, -1672, -1671,
-1670, -1669, -1668, -1667, -1666, -1665, -1664, -1663, -1662,
-1661, -1660, -1659, -1658, -1657, -1656, -1655, -1654, -1653,
-1652, -1651, -1650, -1649, -1648, -1647, -1646, -1645, -1644,
-1643, -1642, -1641, -1640, -1639, -1638, -1637, -1636, -1635,
-1634, -1633, -1632, -1631, -1630, -1629, -1628, -1627, -1626,

```
     -1625, -1624, -1623, -1622, -1621, -1620, -1619, -1618, -1617,
     -1616, -1615, -1614, -1613, -1612, -1611, -1610, -1609, -1608,
     -1607, -1606, -1605, -1604, -1603, -1602, -1601, -1600, -1599,
     -1598, -1597, -1596, -1595, -1594, -1593, -1592, -1591, -1590,
     -1589, -1588, -1587, -1586, -1585, -1584, -1583, -1582, -1581,
     -1580, -1579, -1578, -1577, -1576, -1575, -1574, -1573, -1572,
     -1571, -1570, -1569, -1568, -1567, -1566, -1565, -1564, -1563,
     -1562, -1561, -1560, -1559, -1558, -1557, -1556, -1555, -1554,
     -1553, -1552, -1551, -1550, -1549, -1548, -1547, -1546, -1545,
     -1544, -1543, -1542, -1541, -1540, -1539, -1538, -1537, -1536,
     -1535, -1534, -1533, -1532, -1531, -1530, -1529, -1528, -1527,
     -1526, -1525, -1524, -1523, -1522, -1521, -1520, -1519, -1518,
     -1517, -1516, -1515, -1514, -1513, -1512, -1511, -1510, -1509,
     -1508, -1507, -1506, -1505, -1504, -1503, -1502, -1501, -1500,
     -1499, -1498, -1497, -1496, -1495, -1494, -1493, -1492, -1491,
     -1490, -1489, -1488, -1487, -1486, -1485, -1484, -1483, -1482,
     -1481, -1480, -1479, -1478, -1477, -1476, -1475, -1474, -1473,
     1950, 1951, 1952, 1953, 1954, 1955, 1956, 1957, 1958,
     1959, 1960, 1961, 1962, 1963, 1964, 1965, 1966, 1967,
     1968, 1969, 1970, 1971, 1972, 1973, 1974, 1975, 1976,
     1977, 1978, 1979, 1980, 1981, 1982, 1983, 1984, 1985,
     1986, 1987, 1988, 1989, 1990};
//
// Points at the corners of the wind tunnel
Point(1992) = {-WTDistanceAhead, WTDistanceAbove, -WTHalfWidth, lcWT};
Point(1993) = {WTDistanceAstern, WTDistanceAbove, -WTHalfWidth, lcWT};
Point(1994) = {WTDistanceAstern, -WTDistanceBelow, -WTHalfWidth, lcWT};
Point(1995) = {-WTDistanceAhead, -WTDistanceBelow, -WTHalfWidth, lcWT};
//
// Lines along the edges of the wind tunnel, clockwise
Line(1992) = {1992, 1993};
Line(1993) = {1993, 1994};
Line(1994) = {1994, 1995};
Line(1995) = {1995, 1992};
//
// Line Loop around the wind tunnel, directed outwards
Line Loop(1996) = {1992, 1993, 1994, 1995};
//
// Plane Surface on the wind tunnel, right side,
// excluding the hole left by the membrane.
Plane Surface(1997) = {1996, 1991};
//
// Extrude Plane Surface of the wind tunnel in the Z-direction
NewWT[] = Extrude { 0 , 0 , WTWidth } {
    Surface{1997};
    Layers{1};
    Recombine; };
//
// Physical Surfaces on the membrane (for OpenFoam's use)
//
// Part #A -- All segments on LE circle
Physical Surface("LETube.1") = { NewWT[6] };
Physical Surface("LETube.2") = { NewWT[7] };
Physical Surface("LETube.3") = { NewWT[8] };
```

```
Physical Surface("LETube.4") = { NewWT[9] };
Physical Surface("LETube.5") = { NewWT[10] };
Physical Surface("LETube.6") = { NewWT[11] };
Physical Surface("LETube.7") = { NewWT[12] };
Physical Surface("LETube.8") = { NewWT[13] };
Physical Surface("LETube.9") = { NewWT[14] };
Physical Surface("LETube.10") = { NewWT[15] };
Physical Surface("LETube.11") = { NewWT[16] };
Physical Surface("LETube.12") = { NewWT[17] };
Physical Surface("LETube.13") = { NewWT[18] };
Physical Surface("LETube.14") = { NewWT[19] };
Physical Surface("LETube.15") = { NewWT[20] };
Physical Surface("LETube.16") = { NewWT[21] };
Physical Surface("LETube.17") = { NewWT[22] };
Physical Surface("LETube.18") = { NewWT[23] };
Physical Surface("LETube.19") = { NewWT[24] };
Physical Surface("LETube.20") = { NewWT[25] };
Physical Surface("LETube.21") = { NewWT[26] };
Physical Surface("LETube.22") = { NewWT[27] };
Physical Surface("LETube.23") = { NewWT[28] };
Physical Surface("LETube.24") = { NewWT[29] };
Physical Surface("LETube.25") = { NewWT[30] };
Physical Surface("LETube.26") = { NewWT[31] };
Physical Surface("LETube.27") = { NewWT[32] };
Physical Surface("LETube.28") = { NewWT[33] };
Physical Surface("LETube.29") = { NewWT[34] };
Physical Surface("LETube.30") = { NewWT[35] };
Physical Surface("LETube.31") = { NewWT[36] };
Physical Surface("LETube.32") = { NewWT[37] };
Physical Surface("LETube.33") = { NewWT[38] };
Physical Surface("LETube.34") = { NewWT[39] };
Physical Surface("LETube.35") = { NewWT[40] };
Physical Surface("LETube.36") = { NewWT[41] };
Physical Surface("LETube.37") = { NewWT[42] };
Physical Surface("LETube.38") = { NewWT[43] };
Physical Surface("LETube.39") = { NewWT[44] };
Physical Surface("LETube.40") = { NewWT[45] };
Physical Surface("LETube.41") = { NewWT[46] };
//
// Part #B -- Lower/inner surface of membrane
Physical Surface("SegmentOnBot.1") = { NewWT[47] };
Physical Surface("SegmentOnBot.2") = { NewWT[48] };
Physical Surface("SegmentOnBot.3") = { NewWT[49] };
Physical Surface("SegmentOnBot.4") = { NewWT[50] };
Physical Surface("SegmentOnBot.5") = { NewWT[51] };
```

For the sake of brevity, I have excluded the lines which define physical segments on the bottom #6 through #472.

```
Physical Surface("SegmentOnBot.473") = { NewWT[519] };
Physical Surface("SegmentOnBot.474") = { NewWT[520] };
Physical Surface("SegmentOnBot.475") = { NewWT[521] };
Physical Surface("SegmentOnBot.476") = { NewWT[522] };
Physical Surface("SegmentOnBot.477") = { NewWT[523] };
```

```
//
// Part #C -- Upper/outer surface of membrane
Physical Surface("SegmentOnTop.1") = { NewWT[1000] };
Physical Surface("SegmentOnTop.2") = { NewWT[999] };
Physical Surface("SegmentOnTop.3") = { NewWT[998] };
Physical Surface("SegmentOnTop.4") = { NewWT[997] };
Physical Surface("SegmentOnTop.5") = { NewWT[996] };
```

For the sake of brevity, I have excluded the lines which define physical segments on the top #6 through #472.

```
Physical Surface("SegmentOnTop.473") = { NewWT[528] };
Physical Surface("SegmentOnTop.474") = { NewWT[527] };
Physical Surface("SegmentOnTop.475") = { NewWT[526] };
Physical Surface("SegmentOnTop.476") = { NewWT[525] };
Physical Surface("SegmentOnTop.477") = { NewWT[524] };
//
// Physical Surfaces on the wind tunnel (for OpenFoam's use)
Physical Surface("LeftWall") = { NewWT[0] };
Physical Surface("Top") = { NewWT[2] };
Physical Surface("Outlet") = { NewWT[3] };
Physical Surface("Bottom") = { NewWT[4] };
Physical Surface("Inlet") = { NewWT[5] };
Physical Surface("RightWall") = { 1997 };
//
// Define the Physical Volume for OpenFoam's use
Physical Volume("Internal") = { NewWT[1] };
```

Appendix "E"

Listing of "OpenFoamFunction.txt" generated by Module *WriteOpenFoamFunction*

```
//
// Function to print forces exerted on a flexible membrane.
// The total force is printed every iteration.
// The force on each segment is printed every 250 iterations.
//
functions
{
  TotalForceOnMembrane
  {
    type                 forces;
    functionObjectLibs   ( "libforces.so" );
    patches              ( "SegmentOnTop.*" "SegmentOnBot.*" "LETube.*" );
    rhoName              rhoInf;
    pName                p;
    UName                U;
    log                  true;
    rhoInf               1.225;
    CofR                 ( 0 0 0 );
    outputControl        timeStep;
    outputInterval       1;
  }
  ForceOnTopSegment#1
  {
    type                 forces;
    functionObjectLibs   ( "libforces.so" );
    patches              ( "SegmentOnTop.1" );
    rhoName              rhoInf;
    pName                p;
    UName                U;
    log                  true;
    rhoInf               1.225;
    CofR                 ( 0 0 0 );
    outputControl        timeStep;
    outputInterval       250;
  }
  ForceOnTopSegment#2
  {
    type                 forces;
    functionObjectLibs   ( "libforces.so" );
    patches              ( "SegmentOnTop.2" );
    rhoName              rhoInf;
    pName                p;
    UName                U;
    log                  true;
    rhoInf               1.225;
    CofR                 ( 0 0 0 );
    outputControl        timeStep;
    outputInterval       250;
  }
  ForceOnTopSegment#3
```

```
{
  type                 forces;
  functionObjectLibs   ( "libforces.so" );
  patches              ( "SegmentOnTop.3" );
  rhoName              rhoInf;
  pName                p;
  UName                U;
  log                  true;
  rhoInf               1.225;
  CofR                 ( 0 0 0 );
  outputControl        timeStep;
  outputInterval       250;
}
```

For the sake of brevity, I have not listed the functions which print forces on the top segments #4 through #474.

```
ForceOnTopSegment#475
{
  type                 forces;
  functionObjectLibs   ( "libforces.so" );
  patches              ( "SegmentOnTop.475" );
  rhoName              rhoInf;
  pName                p;
  UName                U;
  log                  true;
  rhoInf               1.225;
  CofR                 ( 0 0 0 );
  outputControl        timeStep;
  outputInterval       250;
}
ForceOnTopSegment#476
{
  type                 forces;
  functionObjectLibs   ( "libforces.so" );
  patches              ( "SegmentOnTop.476" );
  rhoName              rhoInf;
  pName                p;
  UName                U;
  log                  true;
  rhoInf               1.225;
  CofR                 ( 0 0 0 );
  outputControl        timeStep;
  outputInterval       250;
}
ForceOnTopSegment#477
{
  type                 forces;
  functionObjectLibs   ( "libforces.so" );
  patches              ( "SegmentOnTop.477" );
  rhoName              rhoInf;
  pName                p;
  UName                U;
  log                  true;
```

```
  rhoInf               1.225;
  CofR                 ( 0 0 0 );
  outputControl        timeStep;
  outputInterval       250;
}
ForceOnBottomSegment#1
{
  type                 forces;
  functionObjectLibs   ( "libforces.so" );
  patches              ( "SegmentOnBot.1" );
  rhoName              rhoInf;
  pName                p;
  UName                U;
  log                  true;
  rhoInf               1.225;
  CofR                 ( 0 0 0 );
  outputControl        timeStep;
  outputInterval       250;
}
ForceOnBottomSegment#2
{
  type                 forces;
  functionObjectLibs   ( "libforces.so" );
  patches              ( "SegmentOnBot.2" );
  rhoName              rhoInf;
  pName                p;
  UName                U;
  log                  true;
  rhoInf               1.225;
  CofR                 ( 0 0 0 );
  outputControl        timeStep;
  outputInterval       250;
}
ForceOnBottomSegment#3
{
  type                 forces;
  functionObjectLibs   ( "libforces.so" );
  patches              ( "SegmentOnBot.3" );
  rhoName              rhoInf;
  pName                p;
  UName                U;
  log                  true;
  rhoInf               1.225;
  CofR                 ( 0 0 0 );
  outputControl        timeStep;
  outputInterval       250;
}
```

For the sake of brevity, I have not listed the functions which print forces on the bottom segments #4 through #474.

```
ForceOnBottomSegment#475
{
  type                 forces;
```

```
  functionObjectLibs   ( "libforces.so" );
  patches              ( "SegmentOnBot.475" );
  rhoName              rhoInf;
  pName                p;
  UName                U;
  log                  true;
  rhoInf               1.225;
  CofR                 ( 0 0 0 );
  outputControl        timeStep;
  outputInterval       250;
}
ForceOnBottomSegment#476
{
  type                 forces;
  functionObjectLibs   ( "libforces.so" );
  patches              ( "SegmentOnBot.476" );
  rhoName              rhoInf;
  pName                p;
  UName                U;
  log                  true;
  rhoInf               1.225;
  CofR                 ( 0 0 0 );
  outputControl        timeStep;
  outputInterval       250;
}
ForceOnBottomSegment#477
{
  type                 forces;
  functionObjectLibs   ( "libforces.so" );
  patches              ( "SegmentOnBot.477" );
  rhoName              rhoInf;
  pName                p;
  UName                U;
  log                  true;
  rhoInf               1.225;
  CofR                 ( 0 0 0 );
  outputControl        timeStep;
  outputInterval       250;
}
TotalForceOnFreeFlyingSegments
{
  type                 forces;
  functionObjectLibs   ( "libforces.so" );
  patches              ( "SegmentOnTop.*" "SegmentOnBot.*" );
  rhoName              rhoInf;
  pName                p;
  UName                U;
  log                  true;
  rhoInf               1.225;
  CofR                 ( 0 0 0 );
  outputControl        timeStep;
  outputInterval       250;
}
TotalForceOnLETube
```

```
  {
    type                 forces;
    functionObjectLibs   ( "libforces.so" );
    patches              ( "LETube.*" );
    rhoName              rhoInf;
    pName                p;
    UName                U;
    log                  true;
    rhoInf               1.225;
    CofR                 ( 0 0 0 );
    outputControl        timeStep;
    outputInterval       250;
  }
};
```

Appendix “F”

Listing of 11 files from the case directory of the first base case OpenFoam run

Listing of file `constant/polyMesh/boundary`

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  2.1.1                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       polyBoundaryMesh;
    location    "constant/polyMesh";
    object      boundary;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

1001
(
    RightWall
    {
        type            empty;
        nFaces          202439;
        startFace       302504;
    }
    LeftWall
    {
        type            empty;
        nFaces          202439;
        startFace       504943;
    }
    Top
    {
        type            symmetryPlane;
        nFaces          72;
        startFace       707382;
    }
    Outlet
    {
        type            patch;
        nFaces          67;
        startFace       707454;
    }
    Bottom
    {
        type            symmetryPlane;
        nFaces          72;
        startFace       707521;
    }
    Inlet
    {
```

```
    type            patch;
    nFaces          67;
    startFace       707593;
}
LETube.1
{
    type            wall;
    nFaces          3;
    startFace       707660;
}
LETube.2
{
    type            wall;
    nFaces          3;
    startFace       707663;
}
LETube.3
{
    type            wall;
    nFaces          3;
    startFace       707666;
}
```

For the sake of brevity, I have not listed the lines which describe surfaces #4 through #38 on the leading edge tube.

```
LETube.39
{
    type            wall;
    nFaces          3;
    startFace       707773;
}
LETube.40
{
    type            wall;
    nFaces          3;
    startFace       707776;
}
LETube.41
{
    type            wall;
    nFaces          3;
    startFace       707779;
}
SegmentOnBot.1
{
    type            wall;
    nFaces          2;
    startFace       707782;
}
SegmentOnBot.2
{
    type            wall;
    nFaces          2;
    startFace       707784;
}
SegmentOnBot.3
{
```

```
    type            wall;
    nFaces          2;
    startFace       707786;
}
```

For the sake of brevity, I have not listed the lines which describe free-flying surfaces #4 through #474 on the bottom of the nylon membane.

```
SegmentOnBot.475
{
    type            wall;
    nFaces          2;
    startFace       708730;
}
SegmentOnBot.476
{
    type            wall;
    nFaces          2;
    startFace       708732;
}
SegmentOnBot.477
{
    type            wall;
    nFaces          2;
    startFace       708734;
}
SegmentOnTop.477
{
    type            wall;
    nFaces          3;
    startFace       708736;
}
SegmentOnTop.476
{
    type            wall;
    nFaces          2;
    startFace       708739;
}
SegmentOnTop.475
{
    type            wall;
    nFaces          2;
    startFace       708741;
}
```

For the sake of brevity, I have not listed the lines which describe free-flying surfaces #474 through #4 on the top of the nylon membane.

```
SegmentOnTop.3
{
    type            wall;
    nFaces          2;
    startFace       709685;
}
SegmentOnTop.2
{
    type            wall;
    nFaces          2;
```

```
        startFace       709687;
    }
    SegmentOnTop.1
    {
        type            wall;
        nFaces          2;
        startFace       709689;
    }
)
// ************************************************************************* //
```

Listing of file `constant/RASProperties`

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  2.1.1                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "constant";
    object      RASProperties;
}
RASModel        SpalartAllmaras;
turbulence      on;
printCoeffs     on;
```

Listing of file `constant/transportProperties`

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  2.1.1                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "constant";
    object      transportProperties;
}

// U.S. Standard Atmosphere
// Altitude---   Density-----  Dynamic visc---  Kinematic visc-
//      0 feet   1.225 kg/m^3  1.789E-5 Ns/m^2  1.4604E-5 m^2/s
//  5,000        0.7364        1.628E-5         2.2108E-5
// 10,000        0.4135        1.458E-5         3.5260E-5
```

```
// 15,000        0.1948        1.422E-5         7.2998E-5

transportModel  Newtonian;
nu              nu [0 2 -1 0 0 0 0] 1.4604E-5;
rho             rho [ 1 -3 0 0 0 0 0 ] 1.225;
```

Listing of file `system/controlDict`

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  2.1.1                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      controlDict;
}
application     simpleFoam;
startFrom       latestTime;
startTime       0;
stopAt          endTime;
endTime         50000;
deltaT          1;
writeControl    timeStep;
writeInterval   250;
purgeWrite      3;
writeFormat     ascii;
writePrecision  8;
writeCompression off;
timeFormat      general;
timePrecision   6;
runTimeModifiable true;
//
// Function to print forces exerted on a flexible membrane.
// The total force is printed every iteration.
// The force on each segment is printed every 250 iterations.
//
functions
{
  TotalForceOnMembrane
  {
    type                forces;
    functionObjectLibs  ( "libforces.so" );
    patches             ( "SegmentOnTop.*" "SegmentOnBot.*" "LETube.*" );
    rhoName             rhoInf;
    pName               p;
    UName               U;
    log                 true;
    rhoInf              1.225;
    CofR                ( 0 0 0 );
    outputControl       timeStep;
    outputInterval      1;
```

```
}
ForceOnTopSegment#1
{
  type                 forces;
  functionObjectLibs   ( "libforces.so" );
  patches              ( "SegmentOnTop.1" );
  rhoName              rhoInf;
  pName                p;
  UName                U;
  log                  true;
  rhoInf               1.225;
  CofR                 ( 0 0 0 );
  outputControl        timeStep;
  outputInterval       250;
}
```

For the sake of brevity, I have omitted most of the force functions. I have listed only the force functions for the top of the first free-flying segment (above) and the bottom of the last free-flying segment (below).

```
ForceOnBottomSegment#477
{
  type                 forces;
  functionObjectLibs   ( "libforces.so" );
  patches              ( "SegmentOnBot.477" );
  rhoName              rhoInf;
  pName                p;
  UName                U;
  log                  true;
  rhoInf               1.225;
  CofR                 ( 0 0 0 );
  outputControl        timeStep;
  outputInterval       250;
}
TotalForceOnFreeFlyingSegments
{
  type                 forces;
  functionObjectLibs   ( "libforces.so" );
  patches              ( "SegmentOnTop.*" "SegmentOnBot.*" );
  rhoName              rhoInf;
  pName                p;
  UName                U;
  log                  true;
  rhoInf               1.225;
  CofR                 ( 0 0 0 );
  outputControl        timeStep;
  outputInterval       250;
}
TotalForceOnLETube
{
  type                 forces;
  functionObjectLibs   ( "libforces.so" );
  patches              ( "LETube.*" );
  rhoName              rhoInf;
  pName                p;
  UName                U;
  log                  true;
  rhoInf               1.225;
```

```
    CofR                ( 0 0 0 );
    outputControl       timeStep;
    outputInterval      250;
  }
};
```

Listing of file `system/decomposeParDict`

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  2.1.1                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      decomposeParDict;
}
numberOfSubdomains   8;
method               scotch;
scotchCoeffs
{}
distributed          no;
roots
    ();

// To run a case in parallel, do this:
// 1.  <prompt> decomposePar
// 2.  <prompt> mpirun -np 8 simpleFoam -parallel | tee -a ofLog.txt
// 3.  When done, <prompt> reconstructPar
```

Listing of file `system/fvSchemes`

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  2.1.1                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "system";
    object      fvSchemes;
}

ddtSchemes
{
```

```
    default         steadyState;
}

gradSchemes
{
    default         Gauss linear;
    grad(p)         Gauss linear;
    grad(U)         Gauss linear;
}

divSchemes
{
    default         none;
    div(phi,U)      Gauss linearUpwind grad(U);
    div(phi,nuTilda) Gauss linearUpwind grad(nuTilda);
    div((nuEff*dev(T(grad(U))))) Gauss linear;
}

laplacianSchemes
{
    default         none;
    laplacian(nuEff,U) Gauss linear corrected;
    laplacian((1|A(U)),p) Gauss linear corrected;
    laplacian(DnuTildaEff,nuTilda) Gauss linear corrected;
    laplacian(1,p)  Gauss linear corrected;
}

interpolationSchemes
{
    default         linear;
    interpolate(U)  linear;
}

snGradSchemes
{
    default         corrected;
}

fluxRequired
{
    default         no;
    p               ;
}
```

Listing of file `system/fvSolution`

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  2.1.1                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
```

```
    class       dictionary;
    location    "system";
    object      fvSolution;
}
solvers
{
    p
    {
        solver          GAMG;
        tolerance       1e-06;
        relTol          0.05;
        smoother        GaussSeidel;
        nPreSweeps      0;
        nPostSweeps     2;
        cacheAgglomeration true;
        nCellsInCoarsestLevel 10;
        agglomerator    faceAreaPair;
        mergeLevels     1;
    }
    U
    {
        solver          smoothSolver;
        smoother        GaussSeidel;
        nSweeps         2;
        tolerance       1e-08;
        relTol          0.1;
    }
    nuTilda
    {
        solver          smoothSolver;
        smoother        GaussSeidel;
        nSweeps         2;
        tolerance       1e-08;
        relTol          0.1;
    }
}
SIMPLE
{
    nNonOrthogonalCorrectors 0;
    pRefCell        0;
    pRefValue       0;
    residualControl
    {
        p               1e-5;
        U               1e-5;
        nuTilda         1e-5;
    }
}
relaxationFactors
// Start with pRelax=0.35, URelax=0.7 and nuRelax=0.8
{
    fields
    {
        p               0.35;
    }
    equations
    {
        U               0.7;
```

```
        nuTilda         0.8;
    }
}
```

Listing of file `0/p`

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  2.1.1                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volScalarField;
    object      p;
}
dimensions      [0 2 -2 0 0 0 0];
internalField   uniform 0;
boundaryField
{
    Inlet
    {
        type            zeroGradient;
    }
    Outlet
    {
        type         fixedValue;
        value        uniform 0;
    }
    RightWall
    {
        type            empty;
    }
    LeftWall
    {
        type            empty;
    }
    Top
    {
        type            symmetryPlane;
    }
    Bottom
    {
        type            symmetryPlane;
    }
    "SegmentOnTop.*"
    {
        type            zeroGradient;

    }
    "SegmentOnBot.*"
    {
        type            zeroGradient;
```

```
    }
    "LETube.*"
    {
        type            zeroGradient;

    }
}
```

Listing of file `0/U`

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  2.1.1                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volVectorField;
    object      U;
}

// 10mph = 4.4704 m/s
// 20mph = 8.9408 m/s
// 30mph = 13.4112 m/s

dimensions      [0 1 -1 0 0 0 0];
internalField   uniform (8.9408 0 0);

boundaryField
{
    Inlet
    {
       type          fixedValue;
       value         uniform (8.9408 0 0);
    }
    Outlet
    {
        type            zeroGradient;
    }
    RightWall
    {
        type            empty;
    }
    LeftWall
    {
        type            empty;
    }
    Top
    {
        type            symmetryPlane;
    }
    Bottom
```

```
    {
        type            symmetryPlane;
    }
    "SegmentOnTop.*"
    {
        type            fixedValue;
        value           uniform (0 0 0);
    }
    "SegmentOnBot.*"
    {
        type            fixedValue;
        value           uniform (0 0 0);
    }
    "LETube.*"
    {
        type         fixedValue;
        value           uniform (0 0 0);
    }
}
```

Listing of file `0/nuTilda`

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  2.1.1                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volScalarField;
    object      nuTilda;
}

// Calculate nuTilda = sqrt(1.5) * UIl, where
//   U = 8.9408 m/s
//   I = 0.025 is the estimated turbulent intensity
//   l = 25 centimeters is the estimated length scale
// Then, nuTilda = 0.068
// Set the freestream value of nuTilda to five times this.

dimensions      [0 2 -1 0 0 0 0];
internalField   uniform 0.34;
boundaryField
{
    Inlet
    {
        type            freestream;
        freestreamValue uniform 0.34;
    }
    Outlet
    {
        type            freestream;
        freestreamValue uniform 0.34;
```

```
    }
    RightWall
    {
        type            empty;
    }
    LeftWall
    {
        type            empty;
    }
    Top
    {
        type            symmetryPlane;
    }
    Bottom
    {
        type            symmetryPlane;
    }
    "SegmentOnTop.*"
    {
        type            fixedValue;
        value           uniform 0;
    }
    "SegmentOnBot.*"
    {
        type            fixedValue;
        value           uniform 0;
    }
    "LETube.*"
    {
        type            fixedValue;
        value           uniform 0;
    }
}
```

Listing of file `0/nut`

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  2.1.1                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volScalarField;
    object      nut;
}

// Set the freestream value of nut to one-tenth of nuTilda.
// If nuTilda = 0.068, then nut = 0.0068.
dimensions      [0 2 -1 0 0 0 0];
internalField   uniform 0.0068;
boundaryField
{
```

```
    Inlet
    {
        type            freestream;
        freestreamValue uniform 0.0068;
    }
    Outlet
    {
        type            freestream;
        freestreamValue uniform 0.0068;
    }
    RightWall
    {
        type            empty;
    }
    LeftWall
    {
        type            empty;
    }
    Top
    {
        type            symmetryPlane;
    }
    Bottom
    {
        type            symmetryPlane;
    }
    "SegmentOnTop.*"
    {
        type            nutUSpaldingWallFunction;
        value           uniform 0;
    }
    "SegmentOnBot.*"
    {
        type            nutUSpaldingWallFunction;
        value           uniform 0;
    }
    "LETube.*"
    {
        type            nutUSpaldingWallFunction;
        value           uniform 0;
    }
}
```

Appendix "G"

Listing of the Visual Basic program used to automate the equations of static equilibrium

The following program was developed in the Visual Basic 2010 Express version of Visual Basic. It consists solely of a main form.

```
Option Strict On
Option Explicit On

' Calculates equilibrium conditions for a 2D kite setion.

Public Class Form1
    Inherits System.Windows.Forms.Form

    ' Parameters for Case #1 - 10cm TE strings, 20 mph wind speed at sea level
    Public CaseNumber As Int32 = 1
    Public AngleOfAttackDeg As Double = 7
    Public TMfore As Double = 98.456396938              ' Tension at departure point
    Public ThetaMforeDeg As Double = 19.625850678
    Public TMrear As Double = 98.070243234              ' Tension at trailing edge string
    Public ThetaMrearDeg As Double = 12.921531933
    Public FaeroLEPressure_OFx As Double = -0.31996312  ' Forces on LE tube (per meter)
    Public FaeroLEPressure_OFy As Double = 1.52188
    Public FaeroLEViscous_OFx As Double = 0.02830463
    Public FaeroLEViscous_OFy As Double = 0.015333341
    Public MaeroLEPressure_OFz As Double = 0.018512874
    Public MaeroLEViscous_OFz As Double = 0.000070863637

    '' Parameters for Case #3 - 10cm TE strings, 20 mph wind speed at 15,000 feet
    'Public CaseNumber As Int32 = 3
    'Public AngleOfAttackDeg As Double = 7
    'Public TMfore As Double = 15.225690696              ' Tension at departure point
    'Public ThetaMforeDeg As Double = 19.56176824
    'Public TMrear As Double = 15.12565795               ' Tension at trailing edge
string
    'Public ThetaMrearDeg As Double = 12.978171354
    'Public FaeroLEPressure_OFx As Double = -0.0113301   ' Forces on LE tube (per meter)
    'Public FaeroLEPressure_OFy As Double = 0.2183428
    'Public FaeroLEViscous_OFx As Double = 0.0108459
    'Public FaeroLEViscous_OFy As Double = 0.006183
    'Public MaeroLEPressure_OFz As Double = 0.0027174926
    'Public MaeroLEViscous_OFz As Double = 0.000020377324

    ' Force of gravity on LE tube - Equations (B2) and (B3)
    Public RLE As Double = 0.5 * 2.54 / 100         ' Radius of LE tube, meters
    Public tLE As Double = (1 / 32) * 2.54 / 100    ' Thickness of LE tube, meters
    Public RhoLE As Double = 2700                   ' Mass density of aluminum, kg/m^3
    Public g As Double = 9.80665                    ' Gravitational acceleration, m/s^2
    Public FgLE_OFx As Double = 0
    Public FgLE_OFy As Double = -RhoLE * Math.PI * ((RLE ^ 2) - ((RLE - tLE) ^ 2)) * g
    Public FgLE_REFx As Double
    Public FgLE_REFy As Double
    Public POAgLE_OFx As Double
    Public POAgLE_OFy As Double
    Public POAgLE_REFx As Double = RLE
    Public POAgLE_REFy As Double = 0
```

```
' Force of gravity on ribs - Equations (B4) and (B5)
Public Lrib As Double = 1                       ' Length of ribs, meters
Public Mrib As Double = 0.014                   ' Mass of one rib, kg
Public Srib As Double = 2 * 12 * 2.54 / 100     ' Rib spacing, meters
Public FgRib_OFx As Double = 0
Public FgRib_OFy As Double = -(Mrib / Srib) * g
Public FgRib_REFx As Double
Public FgRib_REFy As Double
Public POAgRib_OFx As Double
Public POAgRib_OFy As Double
Public POAgRib_REFx As Double = ((2 * RLE) + (0.5 * Lrib))
Public POAgRib_REFy As Double = 0

' Force of gravity on nylon wrapped around LE tube - Equations (B6A) and (B7A)
Public WNylon As Double = 1                     ' Width of nylon sheet, meters
Public RhoNylon As Double = 0.07                ' Density of nylon, kg/m^2
Public FgNylon1_OFx As Double = 0
Public FgNylon1_OFy As Double = -RhoNylon * 2 * Math.PI * RLE * g
Public FgNylon1_REFx As Double
Public FgNylon1_REFy As Double
Public POAgNylon1_OFx As Double
Public POAgNylon1_OFy As Double
Public POAgNylon1_REFx As Double = RLE
Public POAgNylon1_REFy As Double = 0

' Force of gravity of flat part of nylon - Equations (B6B) and (B7B)
Public FgNylon2_OFx As Double = 0
Public FgNylon2_OFy As Double = -RhoNylon * (WNylon - (2 * Math.PI * RLE)) * g
Public FgNylon2_REFx As Double
Public FgNylon2_REFy As Double
Public POAgNylon2_OFx As Double
Public POAgNylon2_OFy As Double
Public POAgNylon2_REFx As Double = (2 * RLE) + (0.5 * Lrib)
Public POAgNylon2_REFy As Double = 0

' Force of gravity on hardware at front of ribs - Equations (B8A) and (B9A)
Public MgforeHW As Double = 0.03                ' Mass of forward hardware, kg/rib
Public FgforeHW_OFx As Double = 0
Public FgforeHW_OFy As Double = -(MgforeHW / Srib) * g
Public FgforeHW_REFx As Double
Public FgforeHW_REFy As Double
Public POAgforeHW_OFx As Double
Public POAgforeHW_OFy As Double
Public POAgforeHW_REFx As Double = 2 * RLE
Public POAgforeHW_REFy As Double = 0

' Force of gravity on hardware at rear of ribs - Equations (B8B) and (B9B)
Public MgrearHW As Double = 0.015               ' Mass of rear hardware, kg/rib
Public FgrearHW_OFx As Double = 0
Public FgrearHW_OFy As Double = -(MgrearHW / Srib) * g
Public FgrearHW_REFx As Double
Public FgrearHW_REFy As Double
Public POAgrearHW_OFx As Double
Public POAgrearHW_OFy As Double
Public POAgrearHW_REFx As Double = (2 * RLE) + Lrib
Public POAgrearHW_REFy As Double = 0
```

```
' Tension at the departure point - results taken from VB_ShapeFinder
Public ThetaMforeRad As Double = ThetaMforeDeg * Math.PI / 180
Public TMfore_OFx As Double
Public TMfore_OFy As Double
Public TMfore_REFx As Double = TMfore * Math.Cos(ThetaMforeRad)
Public TMfore_REFy As Double = TMfore * Math.Sin(ThetaMforeRad)
Public POAMfore_OFx As Double
Public POAMfore_OFy As Double
Public POAMfore_REFx As Double = RLE * (1 - Math.Cos(ThetaMforeRad))
Public POAMfore_REFy As Double = RLE * Math.Sin(ThetaMforeRad)

' Tension at the trailing edge strings - results taken from VB_ShapeFinder
Public ThetaMrearRad As Double = ThetaMrearDeg * Math.PI / 180
Public TMrear_OFx As Double
Public TMrear_OFy As Double
Public TMrear_REFx As Double = -TMrear * Math.Cos(ThetaMrearRad)
Public TMrear_REFy As Double = TMrear * Math.Sin(ThetaMrearRad)
Public POAMrear_OFx As Double
Public POAMrear_OFy As Double
Public POAMrear_REFx As Double = (2 * RLE) + Lrib
Public POAMrear_REFy As Double = 0

' Aerodynamic forces on LE tube - results taken directly from OpenFoam runs
' POA is assumed to be zero.
Public FaeroLE_OFx As Double = FaeroLEPressure_OFx + FaeroLEViscous_OFx
Public FaeroLE_OFy As Double = FaeroLEPressure_OFy + FaeroLEViscous_OFy
Public MaeroLE_OFz As Double = MaeroLEPressure_OFz + MaeroLEViscous_OFz
Public FaeroLE_REFx As Double
Public FaeroLE_REFy As Double

' Parameters of the bridle lines
Public Lfore As Double
Public Lrear As Double
Public GAMMAforeDeg As Double
Public GAMMArearDeg As Double
Public GAMMAforeRad As Double
Public GAMMArearRad As Double
Public TforeBL As Double
Public TrearBL As Double

' Tension force in tether
Public Ftether As Double
Public BETADeg As Double
Public BETARad As Double
Public TangentBETA As Double
Public Ftether_OFx As Double
Public Ftether_OFy As Double
Public Ftether_REFx As Double
Public Ftether_REFy As Double
Public Mtether_OFz As Double

' Moments around the leading edge due to gravitational forces
Public MgLE_OFz As Double
Public MgRib_OFz As Double
Public MgNylon1_OFz As Double
Public MgNylon2_OFz As Double
Public MgforeHW_OFz As Double
Public MgrearHW_OFz As Double
```

```
' Moments around the leading edge due to membrane tension forces
Public MaeroTfore_OFz As Double
Public MaeroTrear_OFz As Double

' Totals of forces and moments at the leading edge
Public FgTotal_OFx As Double
Public FgTotal_OFy As Double
Public MgTotal_OFz As Double
Public FaeroTotal_OFx As Double
Public FaeroTotal_OFy As Double
Public MaeroTotal_OFz As Double

' Miscelaneous variables
Public DisplayString1 As String = ""
Public DisplayString2 As String = ""

Public Sub New()
    InitializeComponent()
    With Me
        Name = ""
        Text = "Calculate equilibrium for a 2D kite section"
        FormBorderStyle = Windows.Forms.FormBorderStyle.FixedSingle
        Size = New Drawing.Size(1024, 740)
        CenterToScreen()
        Visible = True
        Controls.Add(buttonCalculateTrim) : buttonCalculateTrim.BringToFront()
        Controls.Add(buttonExit) : buttonExit.BringToFront()
        Controls.Add(labelText) : labelText.BringToFront()
        PerformLayout()
    End With
    Initialization()
End Sub

'////////////////////
'// Initialization //
'////////////////////
Public Sub Initialization()
    '
    ' Step #1: Rotate forces and points-of-action vectors wherever possible
    ' 1. Tension at the departure point
    RotateFromREFframetoOFframe(AngleOfAttackDeg, _
        TMfore_REFx, TMfore_REFy, TMfore_OFx, TMfore_OFy)
    RotateFromREFframetoOFframe(AngleOfAttackDeg, _
        POAMfore_REFx, POAMfore_REFy, POAMfore_OFx, POAMfore_OFy)
    ' 2. Tension at the trailing edge strings
    RotateFromREFframetoOFframe(AngleOfAttackDeg, _
        TMrear_REFx, TMrear_REFy, TMrear_OFx, TMrear_OFy)
    RotateFromREFframetoOFframe(AngleOfAttackDeg, _
        POAMrear_REFx, POAMrear_REFy, POAMrear_OFx, POAMrear_OFy)
    ' 3. Aerodynamic forces on LE tube
    RotateFromOFframetoREFframe(AngleOfAttackDeg, _
        FaeroLE_OFx, FaeroLE_OFy, FaeroLE_REFx, FaeroLE_REFy)
    ' 4. Force of gravity on LE tube
    RotateFromOFframetoREFframe(AngleOfAttackDeg, _
        FgLE_OFx, FgLE_OFy, FgLE_REFx, FgLE_REFy)
    RotateFromREFframetoOFframe(AngleOfAttackDeg, _
        POAgLE_REFx, POAgLE_REFy, POAgLE_OFx, POAgLE_OFy)
```

```
' 5. Force of gravity on ribs
RotateFromOFframetoREFframe(AngleOfAttackDeg, _
    FgRib_OFx, FgRib_OFy, FgRib_REFx, FgRib_REFy)
RotateFromREFframetoOFframe(AngleOfAttackDeg, _
    POAgRib_REFx, POAgRib_REFy, POAgRib_OFx, POAgRib_OFy)
' 6. Force of gravity on nylon wrapped around LE tube
RotateFromOFframetoREFframe(AngleOfAttackDeg, _
    FgNylon1_OFx, FgNylon1_OFy, FgNylon1_REFx, FgNylon1_REFy)
RotateFromREFframetoOFframe(AngleOfAttackDeg, _
    POAgNylon1_REFx, POAgNylon1_REFy, POAgNylon1_OFx, POAgNylon1_OFy)
' 7. Force of gravity on flat part of nylon
RotateFromOFframetoREFframe(AngleOfAttackDeg, _
    FgNylon2_OFx, FgNylon2_OFy, FgNylon2_REFx, FgNylon2_REFy)
RotateFromREFframetoOFframe(AngleOfAttackDeg, _
    POAgNylon2_REFx, POAgNylon2_REFy, POAgNylon2_OFx, POAgNylon2_OFy)
' 8. Force of gravity on hardware at front of ribs
RotateFromOFframetoREFframe(AngleOfAttackDeg, _
    FgforeHW_OFx, FgforeHW_OFy, FgforeHW_REFx, FgforeHW_REFy)
RotateFromREFframetoOFframe(AngleOfAttackDeg, _
    POAgforeHW_REFx, POAgforeHW_REFy, POAgforeHW_OFx, POAgforeHW_OFy)
' 9. Force of gravity on hardware at rear of ribs
RotateFromOFframetoREFframe(AngleOfAttackDeg, _
    FgrearHW_OFx, FgrearHW_OFy, FgrearHW_REFx, FgrearHW_REFy)
RotateFromREFframetoOFframe(AngleOfAttackDeg, _
    POAgrearHW_REFx, POAgrearHW_REFy, POAgrearHW_OFx, POAgrearHW_OFy)
'
' Step #2: Calculate the moments around the leading edge - Equation (B13)
' 1. Moment exerted by tension at the departure point
MaeroTfore_OFz = (POAMfore_OFx * TMfore_OFy) - (POAMfore_OFy * TMfore_OFx)
' 2. Moment exerted by tension at the trailing edge strings
MaeroTrear_OFz = (POAMrear_OFx * TMrear_OFy) - (POAMrear_OFy * TMrear_OFx)
' 3. Moment exerted by aerodynamic forces on the LE tube - GIVEN BY OPENFOAM
' 4. Moment exerted by force of gravity on LE tube
MgLE_OFz = (POAgLE_OFx * FgLE_OFy) - (POAgLE_OFy * FgLE_OFx)
' 5. Moment exerted by force of gravity on ribs
MgRib_OFz = (POAgRib_OFx * FgRib_OFy) - (POAgRib_OFy * FgRib_OFx)
' 6. Moment exerted by force of gravity on nylon wrapped around LE tube
MgNylon1_OFz = (POAgNylon1_OFx * FgNylon1_OFy) - (POAgNylon1_OFy * FgNylon1_OFx)
' 7. Moment exerted by force of gravity on flat part of nylon
MgNylon2_OFz = (POAgNylon2_OFx * FgNylon2_OFy) - (POAgNylon2_OFy * FgNylon2_OFx)
' 8. Moment exerted by force of gravity on hardware at front of ribs
MgforeHW_OFz = (POAgforeHW_OFx * FgforeHW_OFy) - (POAgforeHW_OFy * FgforeHW_OFx)
' 9. Moment exerted by force of gravity on hardware at rear of ribs
MgrearHW_OFz = (POAgrearHW_OFx * FgrearHW_OFy) - (POAgrearHW_OFy * FgrearHW_OFx)
'
' Step #3: Add up the gravitational forces and moments
FgTotal_OFx = FgLE_OFx + FgRib_OFx + _
    FgNylon1_OFx + FgNylon2_OFx + FgforeHW_OFx + FgrearHW_OFx
FgTotal_OFy = FgLE_OFy + FgRib_OFy + _
    FgNylon1_OFy + FgNylon2_OFy + FgforeHW_OFy + FgrearHW_OFy
MgTotal_OFz = MgLE_OFz + MgRib_OFz + _
   MgNylon1_OFz + MgNylon2_OFz + MgforeHW_OFz + MgrearHW_OFz
'
' Step #4: Add up the aerodynamic forces and moments
FaeroTotal_OFx = TMfore_OFx + TMrear_OFx + FaeroLE_OFx
FaeroTotal_OFy = TMfore_OFy + TMrear_OFy + FaeroLE_OFy
MaeroTotal_OFz = MaeroTfore_OFz + MaeroTrear_OFz + MaeroLE_OFz
'
```

```
    ' Step #5: Prepare a string to display the data on the screen
    DisplayString1 = _
        "Gravitational forces (N/spanwise meter):" & vbCrLf & _
        "  On LE tube = " & FormatNumber(FgLE_OFy, 6) & vbCrLf & _
        "  On ribs = " & FormatNumber(FgRib_OFy, 6) & vbCrLf & _
        "  On nylon = " & FormatNumber(FgNylon1_OFy + FgNylon2_OFy, 6) & vbCrLf & _
        "  On hardware = " & FormatNumber(FgforeHW_OFy + FgrearHW_OFy, 6) & _
        vbCrLf & vbCrLf & _
        "Total forces acting at the leading edge (N/spanwise meter):" & vbCrLf & _
        "  Fgravity_OFx = " & FormatNumber(FgTotal_OFx, 6) & vbCrLf & _
        "  Fgravity_OFy = " & FormatNumber(FgTotal_OFy, 6) & vbCrLf & _
        "  Faero_OFx = " & FormatNumber(FaeroTotal_OFx, 6) & vbCrLf & _
        "  Faero_OFy = " & FormatNumber(FaeroTotal_OFy, 6) & _
        vbCrLf & vbCrLf & _
        "Total moments acting around the leading edge (N):" & vbCrLf & _
        "  Mgravity = " & FormatNumber(MgTotal_OFz, 6) & vbCrLf & _
        "  Maero = " & FormatNumber(MaeroTotal_OFz, 6)
    Me.Refresh()
End Sub

'////////////////////////////////////////////////////////////////////////////////
'// Controls
'////////////////////////////////////////////////////////////////////////////////

Public WithEvents buttonCalculateTrim As New Windows.Forms.Button With _
    {.Size = New Drawing.Size(100, 30), _
     .Location = New Drawing.Point(5, 5), _
     .Text = "Calculate trim", _
     .TextAlign = ContentAlignment.MiddleCenter}

Public WithEvents buttonExit As New Windows.Forms.Button With _
    {.Size = New Drawing.Size(100, 30), _
     .Location = New Drawing.Point(5, 40), _
     .Text = "Exit", .TextAlign = ContentAlignment.MiddleCenter}

Public labelText As New Windows.Forms.Label With _
    {.Size = New Drawing.Size(1000, 600), _
     .Location = New Drawing.Point(5, 75), _
     .Text = "", .TextAlign = ContentAlignment.TopLeft}

'////////////////////////////////////////////////////////////////////////////////
'// Handlers
'////////////////////////////////////////////////////////////////////////////////

Public Sub buttonCalculateTrim_Click() Handles buttonCalculateTrim.MouseClick
    CalculateGroundTrimEquilibrium()
End Sub

Public Sub buttonExit_Click() Handles buttonExit.MouseClick
    Application.Exit()
End Sub

'////////////////////////////////////////////////////////////////////////////////
'// Subroutine to calculate the ground-trim equilibria
'////////////////////////////////////////////////////////////////////////////////
Public Sub CalculateGroundTrimEquilibrium()
    ' Range of parameters for calculating trim
    Dim LforeBegin As Double = 1
```

```
    Dim LforeEnd As Double = 4
    Dim LforeDelta As Double = 0.5
    Dim lNumLfore As Int32 = 1 + CInt((LforeEnd - LforeBegin) / LforeDelta)
    Dim AngleOfAttackRad As Double = AngleOfAttackDeg * Math.PI / 180
    ' Calculate components of tether tension vector
    Ftether_OFx = -(FgTotal_OFx + FaeroTotal_OFx)
    Ftether_OFy = -(FgTotal_OFy + FaeroTotal_OFy)
    Ftether = Math.Sqrt((Ftether_OFx ^ 2) + (Ftether_OFy ^ 2))
    ' Calculate the tether angle
    TangentBETA = Ftether_OFx / Ftether_OFy
    BETARad = Math.Atan(TangentBETA)
    BETADeg = BETARad * 180 / Math.PI
    ' Calculate the tether moment
    Mtether_OFz = -(MgTotal_OFz + MaeroTotal_OFz)
    ' Prepare a string to display the results on the screen
    DisplayString2 = _
        "Tether tension:" & vbCrLf & _
        "  Magnitude = " & FormatNumber(Ftether, 6) & " N/m" & vbCrLf & _
        "  Angle Beta = " & FormatNumber(BETADeg, 6) & " deg" & _
        vbCrLf & vbCrLf
    ' Step through the range of bridle lengths
    For I As Int32 = 1 To lNumLfore Step 1
        Lfore = LforeBegin + ((I - 1) * LforeDelta)
        Dim lTemp As Double
        lTemp = (((MgTotal_OFz + MaeroTotal_OFz) / Ftether) - _
            (2 * RLE * Math.Cos(BETARad - AngleOfAttackRad))) / Lfore
        GAMMAforeRad = Math.Acos(lTemp) + BETARad - AngleOfAttackRad
        GAMMAforeDeg = GAMMAforeRad * 180 / Math.PI
        Lrear = Math.Sqrt( _
            (Lfore ^ 2) + (Lrib ^ 2) + (-2 * Lfore * Lrib * Math.Cos(GAMMAforeRad)))
        GAMMArearRad = Math.Asin(Lfore * Math.Sin(GAMMAforeRad) / Lrear)
        GAMMArearDeg = GAMMArearRad * 180 / Math.PI
        TrearBL = _
            Ftether * Math.Cos(BETARad - GAMMAforeRad - AngleOfAttackRad) / _
            Math.Sin(GAMMAforeRad + GAMMArearRad)
        TforeBL = _
            Ftether * Math.Cos(BETARad + GAMMArearRad - AngleOfAttackRad) / _
            Math.Sin(GAMMAforeRad + GAMMArearRad)
        DisplayString2 = DisplayString2 & _
            "If LforeBL = " & FormatNumber(Lfore, 6) & " m, then " & _
            "LrearBL = " & FormatNumber(Lrear, 6) & " m" & vbCrLf & _
            "  GammaforeBL = " & FormatNumber(GAMMAforeDeg, 6) & " deg and " & _
            "GammarearBL = " & FormatNumber(GAMMArearDeg, 6) & " deg" & vbCrLf & _
            "  TforeBL = " & FormatNumber(TforeBL, 6) & " N/m and " & _
            "TrearBL = " & FormatNumber(TrearBL, 6) & " N/m" & vbCrLf & vbCrLf
    Next I
    MsgBox(DisplayString1 & vbCrLf & vbCrLf & DisplayString2)
End Sub

'////////////////////////////////////////////////////////////////////////////////
'// Subroutines to rotate vectors - Equations (B1A) and (B1B)
'////////////////////////////////////////////////////////////////////////////////

Public Sub RotateFromOFframetoREFframe( _
    ByVal lRotationAngleDeg As Double, _
    ByVal lXOF As Double, ByVal lYOF As Double, _
    ByRef lXref As Double, ByRef lYref As Double)
    Dim lRotationAngleRad As Double = lRotationAngleDeg * Math.PI / 180
```

```
        Dim lCosRotationAngle As Double = Math.Cos(lRotationAngleRad)
        Dim lSinRotationAngle As Double = Math.Sin(lRotationAngleRad)
        lXref = (lXOF * lCosRotationAngle) - (lYOF * lSinRotationAngle)
        lYref = (lXOF * lSinRotationAngle) + (lYOF * lCosRotationAngle)
    End Sub

    Public Sub RotateFromREFframetoOFframe( _
        ByVal lRotationAngleDeg As Double, _
        ByVal lXref As Double, ByVal lYref As Double, _
        ByRef lXOF As Double, ByRef lYOF As Double)
        Dim lRotationAngleRad As Double = lRotationAngleDeg * Math.PI / 180
        Dim lCosRotationAngle As Double = Math.Cos(lRotationAngleRad)
        Dim lSinRotationAngle As Double = Math.Sin(lRotationAngleRad)
        lXOF = (lXref * lCosRotationAngle) + (lYref * lSinRotationAngle)
        lYOF = (-lXref * lSinRotationAngle) + (lYref * lCosRotationAngle)
    End Sub

End Class
```