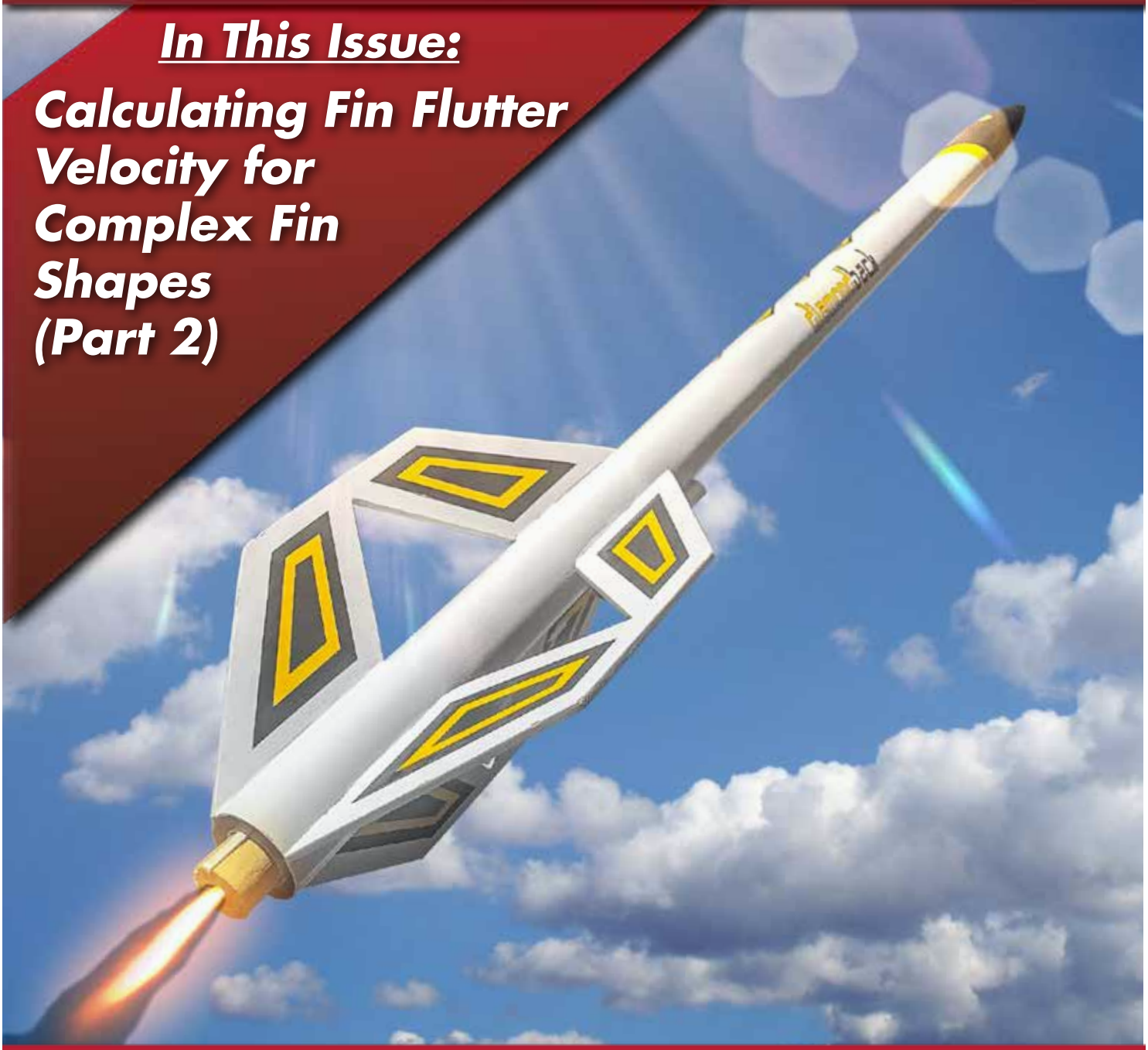
PEAK OF FLIGHT NEWSLETTER

Issue 618 / January 30th 2024

APOGEE COMPONENTS
www.ApogeeRockets.com
4960 Northpark Dr, Colorado Springs CO 80918
Ph# 719-535-9335

In This Issue:

Calculating Fin Flutter Velocity for Complex Fin Shapes (Part 2)



https://www.apogeerockets.com/Rocket-Kits/Skill-Level-2-Model-Rocket-Kits/Diamondback

PEAK OF FLIGHT NEWSLETTER

Issue 618 / January 30th, 2024

Calculating Fin Flutter Velocity for Complex Fin Shapes (Part 2)

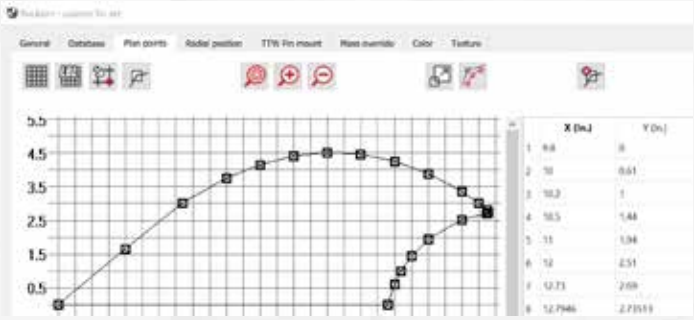
By John K. Bennett

In Part 1 of this article, we learned how to handle triangular, multi-sided, and other fin shapes based upon additive and subtractive geometry. In Part 2 we will generalize these techniques to allow us to estimate the flutter velocity of arbitrary fin shapes. Part 2 will make a lot more sense if Part 1 and *Peak of Flight* Issue #615[1] (where I described the underlying calculations) are read first. Part 2 will involve some computer programming. Even if that is not your cup of tea, I hope that you will find useful information here.

Analyzing Arbitrary Fin Shapes with Multi-Sided Polygon Triangulation

Some fins designs may be sufficiently complex that their analysis is impractical using the methods we have discussed so far. Consider, for example, the fin design on the Apogee Peregrine[2] rocket. This is a commercially available 4-inch rocket kit with an unusual fin design, as shown below:

In *Peak of Flight* #488[3],Tim Van Milligan demonstrated how complex curved fins such as these could be entered into RockSim by selecting vertices on a grid. These vertices are shown in the figure below. OpenRocket provides a similar mechanism for creating fins from a set of vertices.



This fin design may not correspond to easily manipulated geometric elements like circles, triangles, rectangles, or ellipses. So, how do we calculate the area and centroid of fins like these? It turns out that we can generalize geometric decomposition to handle almost any fin design, from simple trapezoidal fins to the complex fin shown here. The basic idea follows the manual process we used for simple polygons:

1. Define the fin outline with a set of vertices.
2. Find a set of triangles within the fin that use these vertices.
3. Calculate the area of each triangle from its vertices. The total fin area is the sum of the areas of the triangles.
4. Calculate Cx of each triangle. The Cx of the entire fin is the weighted average of the Cx's of each triangle.

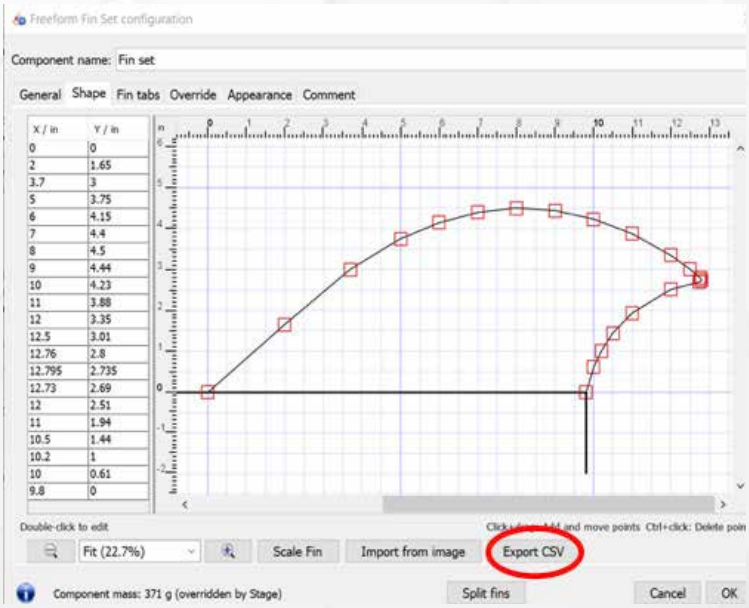
Let's take each of these steps one-by-one.

About this Newsletter
You can subscribe to recieve this e-zine FREE at the Apogee Components website www.ApogeeComponents.com, or by clicking the link here Newsletter Sign-Up

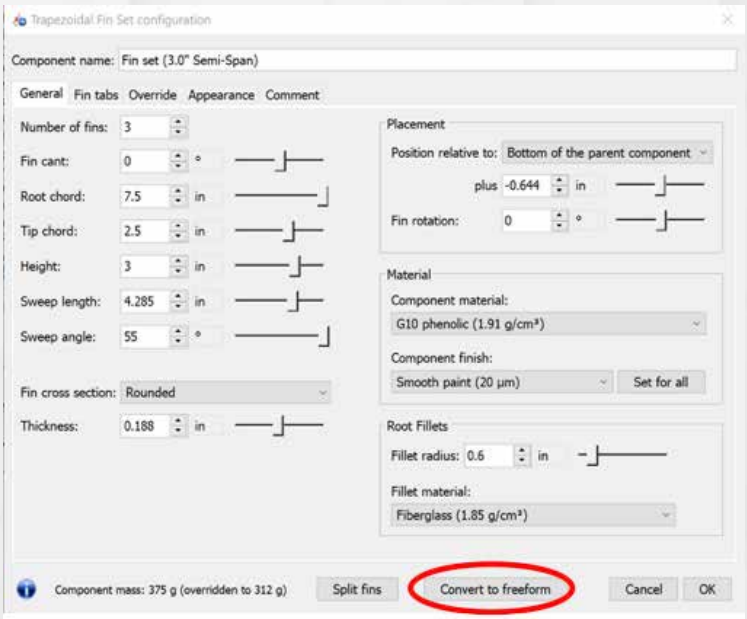
Newsletter Staff
Writer: John K. Bennett
Editor: Tim Van Milligan
Layout: Ryan Conway

Obtaining the Vertices

The vertices come straight from our simulator. For fins already defined by vertices, OpenRocket has an “Export CSV” button on its Freeform Fin Set Configuration page, as shown below.



For trapezoidal fins, OpenRocket has a “Convert to Freeform” button on its Trapezoidal Fin Set Configuration page:



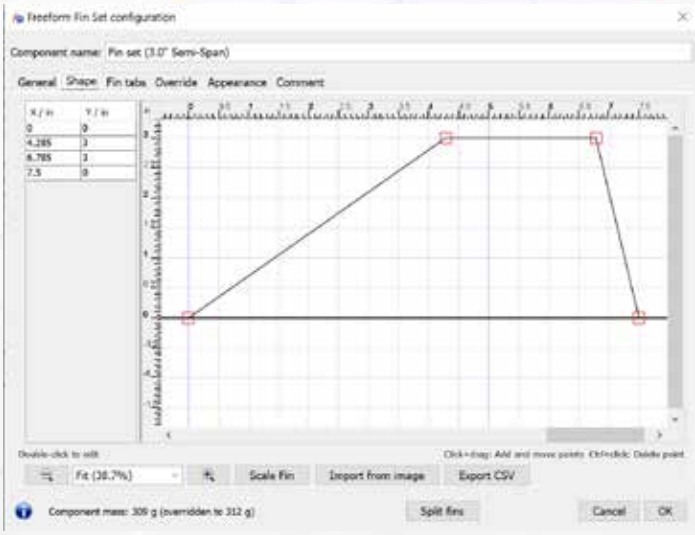
which will produce a freeform page from which the vertices can be exported as above:

PEAK OF FLIGHT NEWSLETTER

Issue 618 / January 30th, 2024

Calculating Fin Flutter Velocity for Complex Fin Shapes (Part 2)

By John K. Bennett



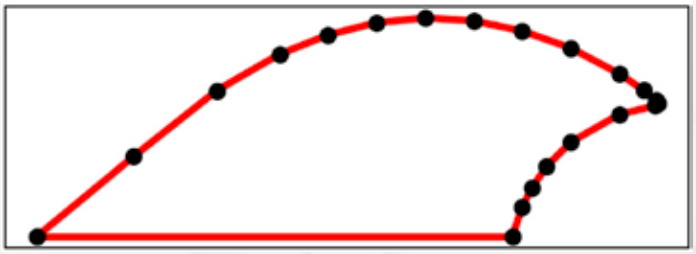
RockSim does not make things quite this easy, but since OpenRocket can open a RockSim file to perform the export, you can use either simulator to design your fins. The vertex file exported by OpenRocket looks like this (for the four-vertex simple fin shown above):

```
X / in, Y / in,
0, 0,
4.285, 3,
6.785, 3,
7.5, 0,
```

This text file has a header row that identifies the X and Y coordinate columns, as well as the unit system. Each row contains one vertex (with the x and y coordinates separated by a comma), and for some reason, an extra comma at the end of the row.

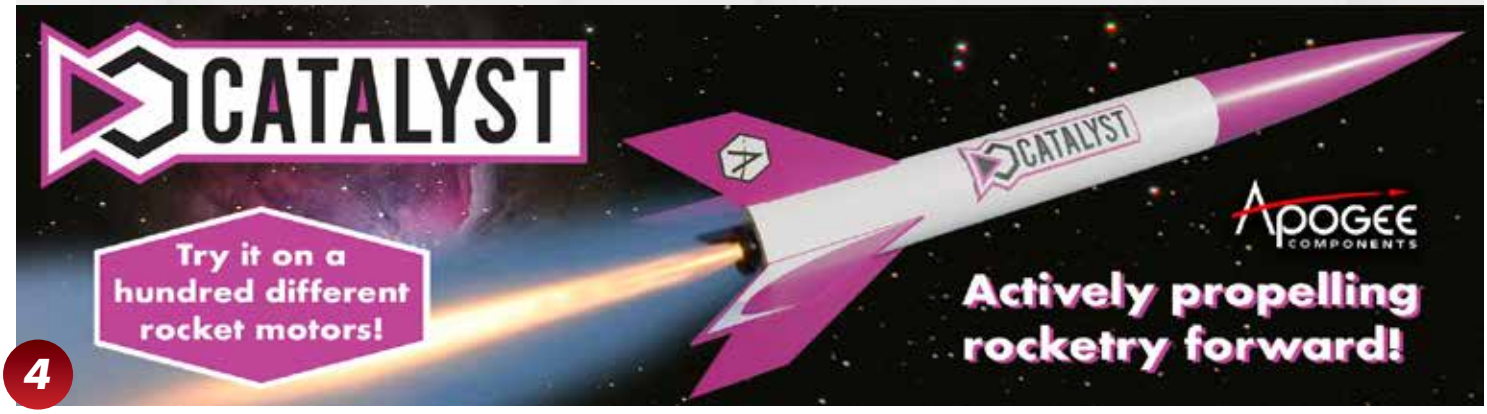
Creating the Triangles

The exported fin vertices should define a closed polygon. The more vertices there are, the smoother any fin edge curve will be. For example, the Peregrine rocket fins define twenty-one vertices:



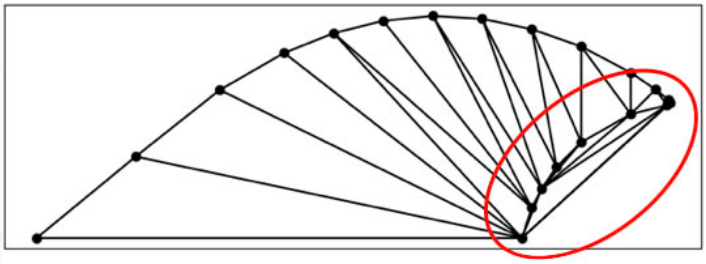
As we saw in Part 1, for small sets of vertices, it is possible to manually construct a set of triangles within the fin polygon. However, as the number of vertices grows, the manual process becomes tedious at best, and at some point, overwhelming. Fortunately, triangulation of polygons is a well-studied area of computational geometry. Much of that work revolves around the inherent complexity of the problem (how fast we can generate the triangles). As it turns out, modern computer graphics is based on creating and processing millions of triangles many times per second.

One popular algorithm for polygon triangulation was created by Boris Delaunay in 1934[4]. A desirable aspect of Delaunay's approach is that it tries to minimize the number of triangles with very small angles, so called "sliver triangles." However, the basic Delaunay algorithm treats the input vertices as just a set of vertices. This results in the algorithm generating extra triangles outside the fin boundary when it encounters a concave shape, as demonstrated in the image on the next page:

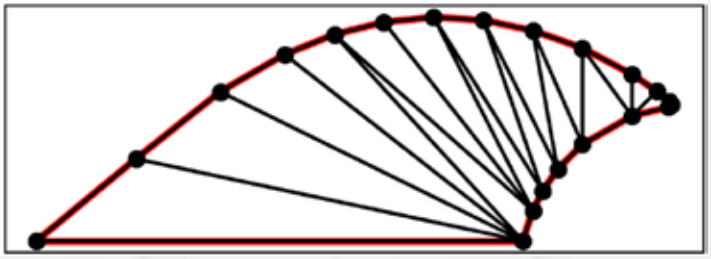


Calculating Fin Flutter Velocity for Complex Fin Shapes (Part 2)

By John K. Bennett

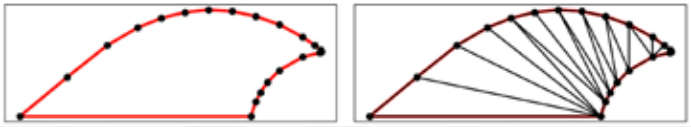


To address this problem, the "constrained" Delaunay triangulation algorithm was developed. Constrained Delaunay triangulation assumes that the input vertices form a "planar straight-line graph", or PSLG, that defines a closed boundary, and requires additional input in the form of non-crossing line segments built from the vertex set, all in one plane, that define that boundary. If we apply the constrained Delaunay triangulation algorithm[5] to our set of vertices and segments, we get the following set of triangles:



This is the result that we want, but we are going to have to do a bit of computer programming to get there. The good news is that well-developed (and free) software packages exist to perform most of what we need to accomplish. Let's work in Python, which has a rich set of software packages for scientific and engineering programming. You can download Python here[6]. A basic Python tutorial can be found here[7]. Help on installing Python packages can be found here[8]. There are also a wide selection of books and online resources for all things Python. Two of my two favorites among the many Python introductory books would be these[9]. We will initially use three Python software packages: `numpy` (a library for working with arrays), `matplotlib` (a library for creating graphs and charts), and `triangle` (the package that will perform the actual triangulation[10]). By using these packages, our (heavily commented) code to perform the triangulation is short **(code-1 in the Appendix).**

If we run this code, we get the following output:



It's always a good idea to display the triangles that are produced as a check that everything went according to plan. If we are curious, we can see how many triangles were produced, and what the list of triangles looks like, by adding two lines (the last two lines shown below) to our code **(code-2 in the Appendix).**

When we run the revised code, we get the following output (in addition to the plot above) **(code-3 in the Appendix).**

Each of the nineteen triangles is defined by three integers. These integers represent indices into the vertex array that we used as input to the triangulation process. For example, adding the **(code-4 in the Appendix)** will print the vertices of the first triangle (whose vertex indices are: [1, 0, 20]).

Running this code prints (in addition to the previous output) the vertices of our first triangle (the one in the bottom left) **(code-5 in the Appendix)**

PEAK OF FLIGHT NEWSLETTER

Issue 618 / January 30th, 2024

Calculating Fin Flutter Velocity for Complex Fin Shapes (Part 2)

By John K. Bennett

Calculating Fin Area and Cx

Now that we have our triangles, we already know how to calculate fin area and Cx, given each triangle's vertices. We just need to write the code to do this. Let's add this code to what we already have **(code-6 in the Appendix)**

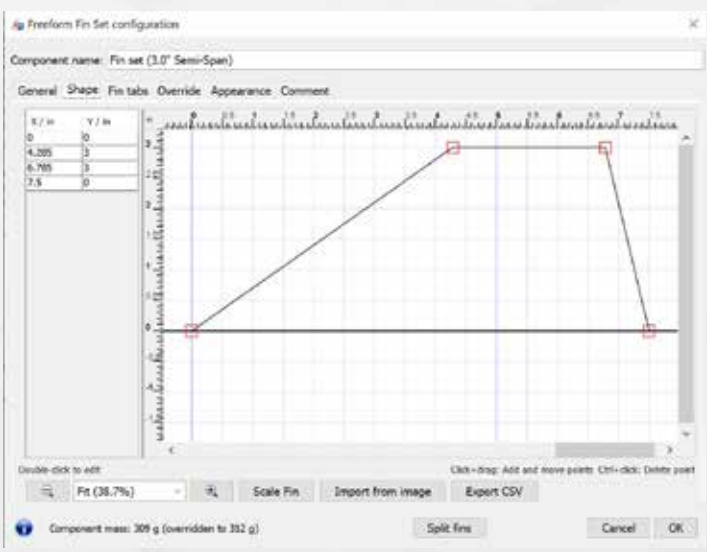
When we run this code, we get: **(code-7 in the Appendix)**

Fin Area = 35.85
Fin Cx = 6.78

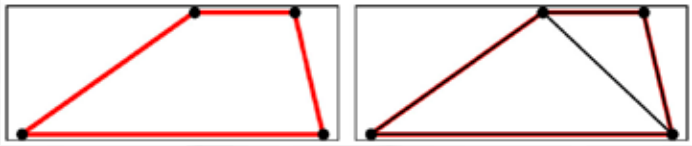
How cool is that? This program (**Fin_Area_and_Cx.py**) is available for download from GitHub[11]. It is worth noting that this program can analyze any fin set for which we can produce a set of vertices, from very simple trapezoidal fins to curved fins with holes. Let's convince ourselves that this is true, starting with the original rocket/fin "worked example" in ***Peak of Flight* #615**[1]. The vertex file produced by that fin is:

```
X / in, Y / in,
0, 0,
4.285, 3,
6.785, 3,
7.5, 0,
```

The fin configuration that produced this file is shown below:



We only need two triangles to characterize this fin, as demonstrated below:



To compute $\mathbf{V_f}$, we need to augment our program (using the same rocket and motor selections as the original PoF article). The entire program so far is shown below. Notice that we have imported another package, "`math`", which we need to use square root and the constant pi. Also, in addition to computing $\mathbf{V_f}$ and $\mathbf{C_x}$, we have added an inputs section, as well as ensuring that the program prints out the correct units for each result. **(code-8 in the Appendix)**

When we run this code, we obtain the following output, confirming that we get the same answer as before: **(code-9 in the Appendix)**

Note that printing of the intermediate calculations can be turned off by setting the constant "VERBOSE" to 0.

Now, let's go back and run this code on the Apogee Peregrine fin set. Do to this, we need to pick a motor. The motor mount tube of the Peregrine is 38mm, and 11 inches long. The Peregrine is intended for dual deployment, so we will be using a plugged motor. The highest total impulse 38mm motors that fit within the motor mount length are the Cesaroni I236-17A Blue Streak and the AeroTech I59WN-P. If we simulate a launch (using RockSim) with these motors, we get the following results:

PEAK OF FLIGHT NEWSLETTER

Issue 618 / January 30th, 2024

Calculating Fin Flutter Velocity for Complex Fin Shapes (Part 2)

By John K. Bennett

Engines loaded	Optimal delay	Max. altitude Feet	Max. velocity Feet / Sec	Max. acceleration Gees
[I236BS-0]	10.53	2544.09	463.44	11.44
[I59WN-0]	6.58	2849.46	285.61	5.68

we enter the highest velocity data (the I236BS) into our program (again turning off the "VERBOSE" switch in the code) and calculate V_f; we get: **(code-10 in the Appendix)**

These data tell us that any 38mm H or I motor that fits within the motor mount dimensions will probably fly well in the Peregrine. If we are willing for the motor case to extend past the motor mount by up to an inch, these motors have the highest total impulse (RockSim launch simulation data is also shown):

Engines loaded	Optimal delay	Max. altitude Feet	Max. velocity Feet / Sec	Max. acceleration Gees
[I24ZWH-0]	11.82	3470.01	565.26	9.77
[I303BS-0]	12.01	3466.34	587.80	14.94
[I435T-0]	12.36	3580.77	623.42	28.52
[I284W-0]	12.10	3527.40	595.57	17.17

If we enter the highest velocity data for these motors (the I435T) into our program and calculate V_f; we get: **(code-11 in the Appendix)**

This is right on the edge of a safe flight, even on a cool day. The other motors are likely fine under most flight conditions, unless it's a super-hot day. If we want to stretch to a J motor, we will need to carefully select one with a slow burn rate at the low end of the J total impulse range, e.g., the Cesaroni P38-5G Mellow (J94), which simulates as follows:

Engines loaded	Optimal delay	Max. altitude Feet	Max. velocity Feet / Sec	Max. acceleration Gees
[J94MY-0]	10.07	4066.90	469.28	5.06

(code-12 in the Appendix)

The final program code that performs these calculations (**Fin_Area_and_Cx_and_Input.py**) can be downloaded from GitHub11. This downloadable version of the code can also optionally read a configuration file (e.g., "rocket.py") that looks just like the input section of the code: **(code-13 in the Appendix)**

This change allows us to create a separate file for each rocket/fin/launch site combination, without having to significantly edit the actual program text itself. We just need to name each file appropriately, and change "rocket" in the four lines of the program code shown below to be the chosen filename prefix. Alternatively, we could leave the program code untouched, and just change each configuration's file name to "rocket.py" (only one at a time): **(code-14 in the Appendix)**

The program code must change slightly to support this optional input file, as shown below (this change has been incorporated in the downloadable version of the code[11]): **(code-15 in the Appendix)**

PEAK OF FLIGHT NEWSLETTER

Issue 618 / January 30th, 2024

Calculating Fin Flutter Velocity for Complex Fin Shapes (Part 2)

By John K. Bennett

End Notes

It is possible to edit the triangle package input to include holes in the fins, e.g.:

tr_input = dict(vertices=verts, segments=segs, holes=holes)

This technique requires us to provide vertices and segments that define each hole, as well as a point within each hole. Completing this modification (more fully described in the Python triangle package documentation[12]) is left to the intrepid reader as an exercise. Before proceeding along these lines, it is probably wise to recall that holes in fins, from an aerodynamic perspective, may be a spectacularly bad idea.

Finally, to reiterate what I said in both Part 1 and PoF #615, these methods only estimate the fin flutter velocity. The choice of input values (altitudes, temperatures, fin dimensions, and in particular, fin material shear modulus) will have a significant impact on the results. We will always strive to use the best data available, but it is important to remember that the final result is only an estimate.

[1] https://www.apogeerockets.com/education/downloads/Newsletter615.pdf

[2] https://www.apogeerockets.com/Rocket-Kits/Skill-Level-3-Model-Rocket-Kits/Peregrine

[3] https://www.apogeerockets.com/education/downloads/Newsletter488_Large.pdf

[4] https://www.mathnet.ru/links/a0aeb3483e137913db801557182e6e35/im4937.pdf
(in French)

[5] L.P. Chew, SCG '87: Proceedings of the Third Annual Symposium on Computational Geometry, October 1987, Pages 215–222

[6] https://www.python.org/downloads/

[7] https://www.w3schools.com/python/python_intro.asp

[8] https://python.land/virtual-environments/installing-packages-with-pip#Python_Install_Pip

[9] https://www.amazon.com/gp/product/B071Z2Q6TQ and https://www.amazon.com/Python-Crash-Course-Eric-Matthes/dp/1718502702

[10] The Python triangle package is actually a Python "wrapper" around Jonathan Shewchuk's two-dimensional mesh generator and Delaunay triangulator library, written in C (see https://www.cs.cmu.edu/~quake/triangle.html).

[11] https://github.com/jkb-git/Fin-Flutter-Velocity-Calculator

[12] https://rufat.be/triangle/examples.html

About the Author

John Bennett is an engineering Professor Emeritus at the University of Colorado Boulder, and a TRA and NAR member in OregonRocketry. Using Brinley's book as a guide, he built and flew many Zn/S and KN03/sugar rockets as a teenager. In retirement, he is now rediscovering his love of amateur rocketry, this time with better supervision.

APPENDIX

code-1

```
import numpy as np
import matplotlib.pyplot as plt
import triangle as tr

# Read in the vertex file exported by OpenRocket
verts = np.genfromtxt("peregrinefins.csv", delimiter=",")
verts = np.delete(verts, 0, 0)  # delete the header row of the csv file
verts = np.delete(verts, 2, 1)  # delete third column of csv file (created by extra
comma)
segs = []                       # must specify segments as well as verts to create a PSLG
for i in range (len(verts)):    # create segments of a closed PSLG from list of vertices
    seg = []
    seg.append(i)
    if (i == (len(verts)-1)):
        seg.append(0)
    else:
        seg.append(i+1)
    segs.append(seg)
tr_input = dict(vertices=verts, segments=segs)  # triangle expects Python dict as input
tr_output = tr.triangulate(tr_input,'p')        # make triangles. 'p' = input is a PSLG
tris = tr_output['triangles'].tolist()          # convert output of triangle to Python
list
tr.compare(plt, tr_input, tr_output)            # set up what we are going to plot
plt.show()
```

code-2

```
tr_input = dict(vertices=verts, segments=segs)  # triangle expects a Python dict as input
tr_output = tr.triangulate(tr_input,'p')        # make triangles. 'p' = input is a PSLG
tris = tr_output['triangles'].tolist()          # convert output of triangle to Python
list
# tr.compare(plt, tr_input, tr_output)          # set up what we are going to plot
# plt.show()
print ("Number of Triangles Produced = ", len(tris))
print (tris)
```

code-3

```
Number of Triangles Produced =  19
[[1, 0, 20], [3, 2, 20], [20, 2, 1], [4, 20, 19], [3, 20, 4], [19, 6, 5], [5, 4,
 19], [7, 6, 18], [6, 19, 18], [17, 8, 7], [15, 14, 11], [16, 15, 9], [17, 16, 8
], [15, 10, 9], [9, 8, 16], [14, 12, 11], [11, 10, 15], [14, 13, 12], [17, 7, 18
]]
```

code-4

```
print("Number of Triangles Produced = ", len(tris))
print(tris)
print("First Triangle Vertices = (", \
      verts[1].tolist(), ", ", verts[0].tolist(), ", " , verts[20].tolist(),")")
```

code-5

```
First Triangle Vertices:( [2.0, 1.65] ,  [0.0, 0.0] ,  [9.8, 0.0]
```

code-6

```
# for each of our triangles, compute an area and a Cx
tri_areas = []      # this list will hold the areas of each triangle
tri_Cxs = []        # this list will hold the Cx's of each triangle
fin_area = 0
fin_Cx = 0
for tri in tris:
    A = verts[tri[0]]       #extract the vertices of this triangle
    B = verts[tri[1]]
    C = verts[tri[2]]
    # compute triangle area; 0 is the index of the x coord; 1 is the index of the y coord
    ar = ((A[0]*(B[1]-C[1])) + (B[0]*(C[1] - A[1])) + (C[0]*(A[1] - B[1])))/2
    tri_areas.append(ar)    # add the area of this triangle to our list of areas
    fin_area += ar          # add the area of this triangle to the total fin area
    # compute triangle Cx; 0 is the index of the x coord
    tri_Cx = (A[0] + B[0] + C[0]) / 3
    tri_Cxs.append(tri_Cx)  # add the Cx of this triangle to our list of Cx's
# compute Fin Cx by taking the weighted average of all the triangle Cx's
for i in range(len(tri_areas)):             # first get the numerator
    fin_Cx += (tri_Cxs[i] * tri_areas[i])   # use fin_Cx to hold intermediate result
fin_Cx = (fin_Cx / fin_area)                # now divide by the total fin area
print ("Fin Area = ", "{:2.2f}".format(fin_area))
print ("Fin Cx = ", "{:2.2f}".format(fin_Cx))
```

code-7

```
Fin Area =  35.85
Fin Cx =  6.78
```

code-8

```
import numpy as np
import matplotlib.pyplot as plt
import triangle as tr
import math

# Start of Inputs
VERBOSE = 0    # '0' only prints final results; 1 prints intermediate results
# Inputs to be provided
Units = "Imperial" # "Imperial" = inches,feet,psi,deg.F; "SI" = cm,meters,KPa,deg.C
# Input provided below must match selection above
# Rocket and Launch Site Specs
```

code-8 (continued)

```
MaxV = 1500          # maximum predicted rocket velocity
AMaxV = 14000        # predicted altitude at predicted maximum rocket velocity (AGL)
LSA = 4500           # launch site altitude (ASL)
TLS = 65             # Temp (Fahrenheit or Centigrade, depending upon selected units)
DEF = "DST"          # Use Default Sea-Level Temp ("DST") or Launch Site Temp ("LST")
#Fin Specs

Thickness = 0.1875  # Fin Thickness
TC = 2.5            # if TC == -1, calculate TC (or a Pseudo TC)
RC = 7.5            # Root Chord length
Height = 3          # Fin Height
GE = 600000         # Shear Modulus
T2T  = "NO"         # Tip-to-Tip reinforcing present? "YES" or "NO"
Fin_Vertex_File_Name = "trap.csv" # Name of csv file output from OpenRocket
# End of Inputs

# Read in the vertex file exported by OpenRocket (see articles for explanation)
verts = np.genfromtxt(Fin_Vertex_File_Name, delimiter=",")
verts = np.delete(verts, 0, 0)  # delete the header row of the csv file
verts = np.delete(verts, 2, 1)  # delete third column of csv file (created by extra
comma)
segs = []                       # must specify segments and vertices to create a PSLG
for i in range (len(verts)):    # create segments of a closed PSLG from list of vertices
    seg = []
    seg.append(i)
    if (i == (len(verts)-1)):
        seg.append(0)
    else:
        seg.append(i+1)
    segs.append(seg)
tr_input = dict(vertices=verts, segments=segs)  # triangle expects Python dict as input
tr_output = tr.triangulate(tr_input,'p')        # make triangles. 'p' says input is a
PSLG
tris = tr_output['triangles'].tolist()          # convert output of triangle to Python
list
# print("Number of Triangles Produced = ", len(tris))
# print (tris)
# print ("First Triangle Vertices = (", \
#        verts[1].tolist(), ", " , verts[0].tolist(), ", " , verts[20].tolist(),")")
# next two lines moved to end of file so plot can stay visible until closed
# tr.compare(plt, tr_input, tr_output)            # set up what we are going to plot
# plt.show()                                      # create the plot

# for each generated triangle, compute an area and a Cx
tri_areas = []      # this list will hold the area of each triangle
tri_Cxs = []        # this list will hold the Cx of each triangle
fin_area = 0
fin_Cx = 0
for tri in tris:          # for every generated triangle
```

code-8 (continued)

```
    A = verts[tri[0]]     # extract the vertices of this triangle
    B = verts[tri[1]]
    C = verts[tri[2]]
    # compute triangle area; 0 is index of vertex x coord; 1 is index of vertex y coord
    ar = ((A[0]*(B[1]-C[1])) + (B[0]*(C[1] - A[1])) + (C[0]*(A[1] - B[1])))/2
    tri_areas.append(ar)    # add the area of this triangle to our list of areas
    fin_area += ar          # add the area of this triangle to the total fin area
    # compute triangle Cx; 0 is the index of the vertex x coord
    tri_Cx = (A[0] + B[0] + C[0]) / 3
    tri_Cxs.append(tri_Cx)  # add the Cx of this triangle to our list of Cx's
# compute Fin Cx by taking the weighted average of all the triangle Cx's
for i in range(len(tri_areas)):  # first get the numerator (weighted sum of areas)
    fin_Cx += (tri_Cxs[i] * tri_areas[i])
fin_Cx = (fin_Cx / fin_area)     # now divide by the total fin area to get Cx

# Compute Vf; first compute all intermediate values
Fin_Eps = (fin_Cx / RC) - 0.25  # compute epsilon (see article for definition)
if (TC < 0): # if TC is entered as -1, compute TC or pseudo TC
    TC = (((fin_area / Height) * 2) - RC)
ThicknessRatio = (Thickness / RC)     # compute three fin ratios
Lambda = (TC / RC) # if TC = 0, Lambda will also be zero (triangular fin)
AspectRatio = ((Height * Height) / fin_area)
if (Units == "Imperial"):# set constants and suffices to Imperial Units
    p0 = 14.696
    DST_Base_Temp = 59
    Temp_Dec_Per_Unit = 0.00356
    SoS_Mult = 49.03
    Low_Temp = 459.7
    T0 = 518.7
    velsuf = "ft/sec"
    finsuf = "in"
    altsuf = "feet"
    tempsuf = "deg F"
    GEsuf = "psi"
    areasuf = "sq in"
else: # set constants and suffices to SI Units
    p0 = 101.325
    DST_Base_Temp = 15
    Temp_Dec_Per_Unit = .0065
    SoS_Mult = 20.05
    Low_Temp = 273.16
    T0 = 288.16
    velsuf = "meters/sec"
    finsuf = "cm"
    altsuf = "meters"
    tempsuf = "deg C"
    GEsuf = "KPa"
    areasuf = "sq cm"
DN = (24 * Fin_Eps * 1.4 * p0) / math.pi     # compute "denominator constant"
```

code-8 (continued)

```
 if(DEF=="DST"):    # Use Default Sea-Level Temp ("DST") as base
     Temp = (DST_Base_Temp - (Temp_Dec_Per_Unit * (LSA + AMaxV)))
 else: # Use Launch Site Temp ("LST") as base
     Temp = (TLS-(Temp_Dec_Per_Unit * (AMaxV)))
Spd_of_Sound = SoS_Mult * math.sqrt(Low_Temp + Temp) # Compute speed of sound
airP = p0 * pow(((Temp  + Low_Temp)/T0),5.256)            # Compute air pressure
Term1 = (DN * pow(AspectRatio, 3)) / (pow(ThicknessRatio, 3) * (AspectRatio + 2))
Term2 = (Lambda + 1)/2
Term3 = (airP / p0)
if (T2T == "YES"): # if tip to tip reinforcing present, double GE
       GE = 2 * GE
Vf = Spd_of_Sound * math.sqrt(GE/(Term1 * Term2 * Term3))
Margin = Vf - MaxV # compute safety margin (see article)
Margin_Pct = 100 * ((Vf-MaxV)/MaxV)

# print the input for verification
print ("Unit System = ", Units)
# Rocket and Launch Site Specs, as provided
print ("****Rocket and Launch Site Specs****")
print ("MaxV = ", "{:2.1f}".format(MaxV), velsuf)
print ("AMaxV = ", "{:2.1f}".format(AMaxV), altsuf)
print ("LSA = ", "{:2.1f}".format(LSA), altsuf)
print ("TLS = ", "{:2.1f}".format(TLS), tempsuf)
print ("DEF = ", DEF)
print ("****Fin Specs****")
print ("RC = ", "{:2.3f}".format(RC), finsuf)
print ("Height = ", "{:2.3f}".format(Height), finsuf)
print ("Thickness = ", "{:2.4f}".format(Thickness), finsuf)
print ("TC = ", "{:2.3f}".format(TC), finsuf)
if (T2T == "YES"):
    print ("Tip-to-tip reinforcing present; GE doubled to: ","{:2.1f}".format(GE),GEsuf)
else:
    print ("GE = ","{:2.1f}".format(GE),GEsuf)

print ("T2T = ", T2T)
print ("Fin_Vertex_File_Name = ", Fin_Vertex_File_Name)
print()
if (VERBOSE): # print all the intermediate values, if VERBOSE is true
    print ("Fin Eps = ", "{:2.3f}".format(Fin_Eps))
    print ("TC or Pseudo TC = ", "{:2.2f}".format(TC), finsuf)
    print ("Thickness Ratio = ", "{:2.3f}".format(ThicknessRatio))
    print ("Lambda = ", "{:2.3f}".format(Lambda))
    print ("Aspect Ratio = ", "{:2.3f}".format(AspectRatio))
    print ("DN = ", "{:2.2f}".format(DN), GEsuf)
    print ("Temp at MaxV = ", "{:2.2f}".format(Temp), tempsuf)
    print ("Spd_of_Sound = ", "{:2.2f}".format(Spd_of_Sound), velsuf)
    print ("airP = ", "{:2.2f}".format(airP), GEsuf)
    print ("Term1 = ", "{:2.2f}".format(Term1), GEsuf)
    print ("Term2 = ", "{:2.2f}".format(Term2))
    print ("Term3 = ", "{:2.2f}".format(Term3))
```

PEAK OF FLIGHT NEWSLETTER

Issue 618 / January 30th, 2024

Calculating Fin Flutter Velocity for Complex Fin Shapes (Part 2)

By John K. Bennett

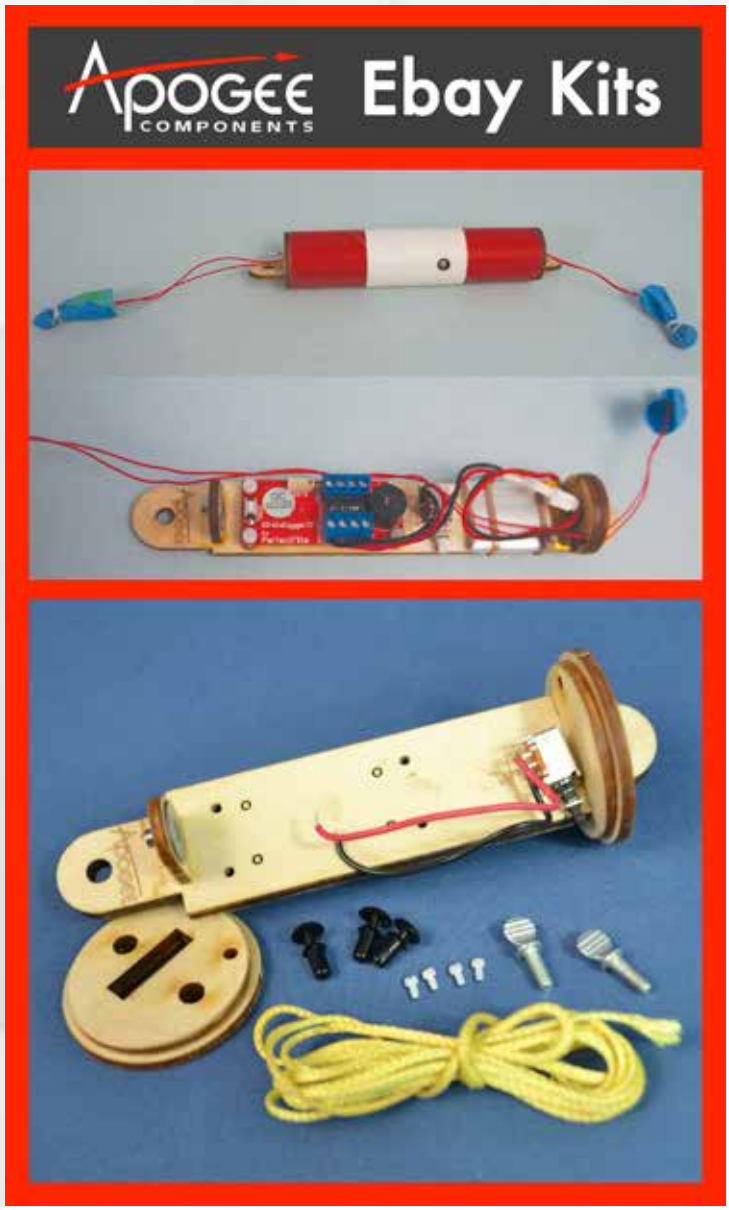
code-8 (continued)

```
# always print the important stuff
print ("Fin Area = ", "{:2.2f}".format(fin_area), areasuf)
print ("Fin Cx = ", "{:2.2f}".format(fin_Cx), finsuf)
print ("Vf = ", "{:2.1f}".format(Vf), velsuf)
print ("Margin = ", "{:2.1f}".format(Margin), velsuf)
print ("Margin% = ", "{:2.1f}%".format(Margin_Pct))
# plot moved to here so we wouldn't have to close plot to see results
tr.compare(plt, tr_input, tr_output)            # set up what we are going to plot
plt.show()                                      # create the plot
```

code-9

```
Number of Triangles Produced =  2
[[1, 0, 3], [3, 2, 1]]
Unit System =  Imperial
****Rocket and Launch Site Specs****
MaxV =  1500.0 ft/sec
AMaxV =  14000.0 feet
LSA =  4500.0 feet
TLS =  65.0 deg F
DEF =  DST
****Fin Specs****
RC =  7.500 in
Height =  3.000 in
Thickness =  0.1875 in
TC =  2.500 in
GE =  600000.0 psi
T2T =  NO
Fin_Vertex_File_Name =  trap.csv

Fin Area =  15.00 sq ft
Fin Cx =  4.49 in
Vf =  2618.1 ft/sec
Margin =  1118.1 ft/sec
Margin% =  74.5%
```



PEAK OF FLIGHT NEWSLETTER

Issue 618 / January 30th, 2024

Calculating Fin Flutter Velocity for Complex Fin Shapes (Part 2)

By John K. Bennett

code-10

```
Unit System =  Imperial
****Rocket and Launch Site Specs****
MaxV =  464.0 ft/sec
AMaxV =  2544.0 feet
LSA =  4500.0 feet
TLS =  65.0 deg F
DEF =  DST
****Fin Specs****
RC =  9.800 in
Height =  4.500 in
Thickness =  0.2500 in
TC =  6.133 in
GE =  89000.0 psi
T2T =  NO
Fin_Vertex_File_Name =  peregrinefins.csv

Fin Area =  35.85 sq ft
Fin Cx =  6.78 in
Vf =  757.6 ft/sec
Margin =  293.6 ft/sec
Margin% =  63.3%
```

code-11

```
Unit System =  Imperial
****Rocket and Launch Site Specs****
MaxV =  623.5 ft/sec
AMaxV =  3581.0 feet
LSA =  4500.0 feet
TLS =  65.0 deg F
DEF =  DST
****Fin Specs****
RC =  9.800 in
Height =  4.500 in
Thickness =  0.2500 in
TC =  6.133 in
GE =  89000.0 psi
T2T =  NO
Fin_Vertex_File_Name =  peregrinefins.csv

Fin Area =  35.85 sq ft
Fin Cx =  6.78 in
Vf =  769.8 ft/sec
Margin =  146.3 ft/sec
Margin% =  23.5%
```

PEAK OF FLIGHT NEWSLETTER

Issue 618 / January 30th, 2024

Calculating Fin Flutter Velocity for Complex Fin Shapes (Part 2)

By John K. Bennett

code-12

```
Unit System =  Imperial
****Rocket and Launch Site Specs****
MaxV =  469.3 ft/sec
AMaxV =  4067.0 feet
LSA =  4500.0 feet
TLS =  65.0 deg F
DEF =  DST
****Fin Specs****
RC =  9.800 in
Height =  4.500 in
Thickness =  0.2500 in
TC =  6.133 in
GE =  89000.0 psi
T2T =  NO
Fin_Vertex_File_Name =  peregrinefins.csv

Fin Area =  35.85 sq ft
Fin Cx =  6.78 in
Vf =  775.6 ft/sec
Margin =  306.3 ft/sec
Margin% =  65.3%
```

code-13

rocket.py

```
VERBOSE = 0     # Set to 0 to only print final results; set to 1 to see intermediate results
# Inputs to be provided
Units = "Imperial"  # "Imperial" = inches,feet,psi,deg. F; "SI" = cm,meters,KPa,deg. C
# Input provided below must match selection above
# Rocket and Launch Site Specs
MaxV = 1500          # maximum predicted rocket velocity
AMaxV = 14000        # predicted altitude at predicted maximum rocket velocity (AGL)
LSA = 4500           # launch site altitude (ASL)
TLS = 65             # Temp (Fahrenheit or Centigrade, depending upon unit selection)
DEF = "DST"          # Use Default Sea-Level Temp ("DST") or Launch Site Temp ("LST")
#Fin Specs
Thickness = 0.1875   # Fin Thickness
TC = -1              # if TC == -1, calculate TC (or a Pseudo TC)
RC = 7.5             # Root Chord length
Height = 3           # Fin Height
GE = 600000          # Shear Modulus
T2T  = "NO"          # Tip-to-Tip reinforcing present? "YES" or "NO"
Fin_Vertex_File_Name = "trap.csv" # Name of csv file output from OpenRocket
# End of Inputs
```

code-14

```
rocket = Path("rocket.py")
if rocket.is_file():      # if input file exists for initialization, read it
    # Read in the config file
    from rocket import *
```

code-15

```
# start of change
from pathlib import Path # we need the pathlib package to manage filename paths
# *** you must use the same name in the next four lines. i.e.,
    # peregrine_rocket = Path("peregrine_rocket.py")
    # if peregrine_rocket.is_file():  # if input file exists for initialization, read it
    # # Read in the config file
    #  from peregrine_rocket import *
rocket = Path("rocket.py")
if rocket.is_file():      # if input file exists for initialization, read it
    # Read in the config file
    from rocket import *
else:                          # otherwise, initialize variables as follows
      VERBOSE = 0 # Set to 0 to only see final results; set to 1 to see full results
      # Inputs to be provided
      Units = "Imperial"  # "Imperial" = inches,feet,psi,deg.F; "SI" =
cm,meters,KPa,deg.C
      # Input provided below must match selection above
      # Rocket and Launch Site Specs
      MaxV = 1500         # maximum predicted rocket velocity
      AMaxV = 14000       # predicted altitude at predicted maximum rocket velocity (AGL)
      LSA = 4500          # launch site altitude (ASL)
      TLS = 65            # Temp (Fahrenheit or Centigrade, depending upon unit
selection)
      DEF = "DST"         # Use Default Sea-Level Temp ("DST") or Launch Site Temp
("LST")
      #Fin Specs
      Thickness = 0.1875  # Fin Thickness
      TC = -1             # if TC == -1, calculate TC or Pseudo TC
      RC = 7.5            # Root chord length
      Height = 3          # Fin Height
      GE = 600000         # Shear Modulus
      T2T = "NO"          # Tip-to-Tip reinforcing present? "YES" or "NO"
      Fin_Vertex_File_Name = "trap.csv" # Name of csv file output from OpenRocket
      # End of Inputs
# end of change
```