Walt Erbach's Calculator

Walter Erbach was a respected Professor of Engineering Mechanics at the University of Nebraska who retired from teaching in 1984. He authored several papers on indoor model design that were published in the NFFS Symposiums. At the time when he was most active, computers were huge machines that were only available in big organizations like universities.

In the 1980 edition of the Symposium, Walt presented a program that used simple theory to calculate the power needed for an indoor model to maintain level flight. This program was written in *Basic* for the Commodore 64, one of the first machines available for home use. In this note, we will recreate code he wrote in Python.

To make this more interesting, we will use **matplotlib** to generate plots showing our calculations. To make this happen, we need to initialize the plotting system:

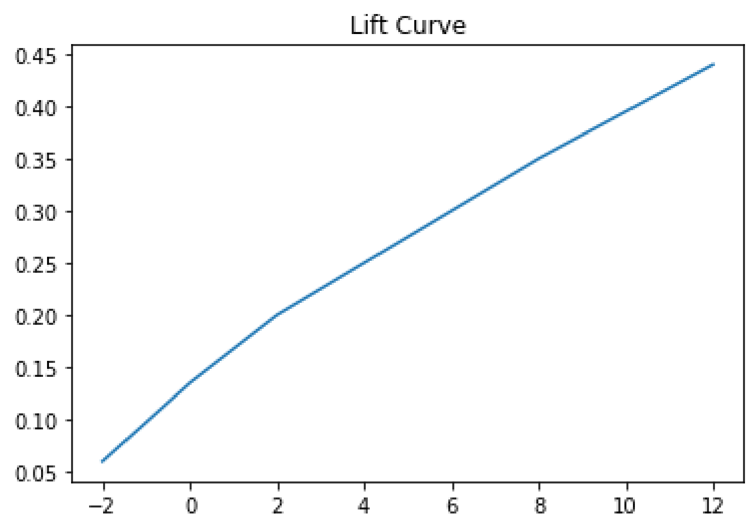
In [2]:
```
%matplotlib inline

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

Walt decided to use the McBride B7 airfoil on his model and starte dhi program with lift and drag coefficient data found in a *Frank Zaic Yearbook*

In [3]:
```
C_l = [0.06, 0.135, 0.2, 0.25, 0.3, 0.35, 0.395, 0.44]
C_d = [0.008, 0.009, 0.01, 0.012, 0.014, 0.019, 0.024, 0.0335]
alpha = [-2.0, 0.0, 2.0, 4.0, 6.0, 8.0, 10.0, 12.0]
```

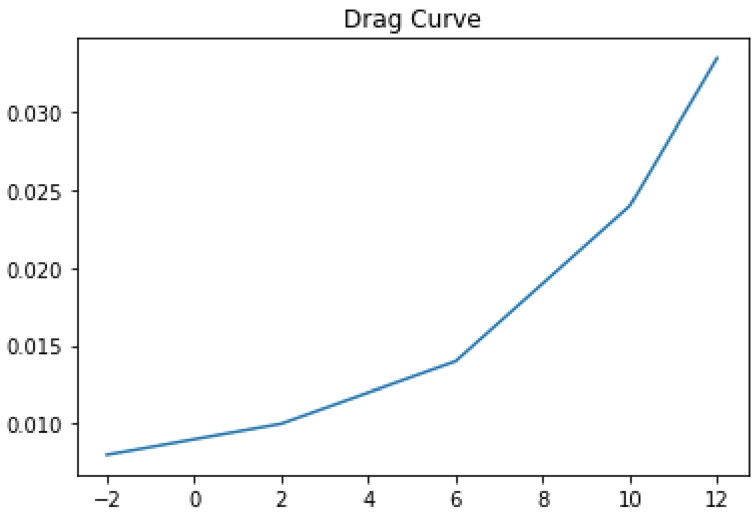
We can see this data using **matplotlib**:

In [4]:
```
plt.plot(alpha, C_l)
plt.title('Lift Curve')
plt.show()
```



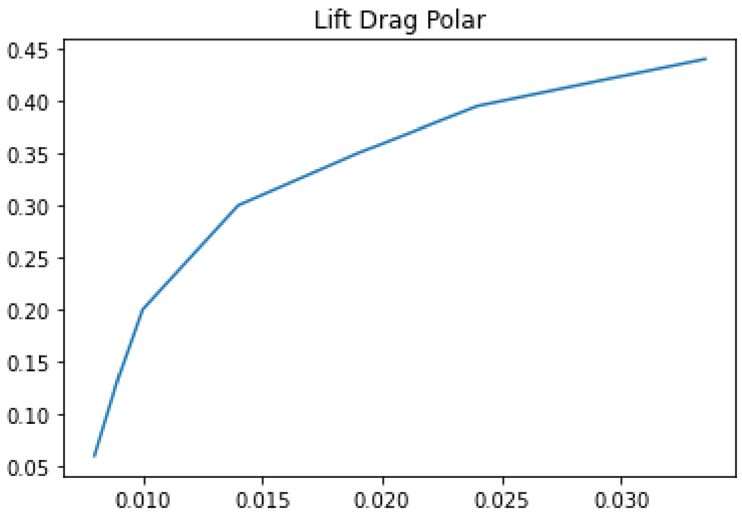
In [5]:

```
plt.plot(alpha, C_d)
plt.title('Drag Curve')
plt.show()
```



In [6]:

```
plt.plot(C_d, C_l)
plt.title('Lift Drag Polar')
plt.show()
```



These curves are a bit choppy. We will make them more useful later.

Next, we need to define a few parameters for the model we are going to study:

In [ ]:

```
SW = 150.0      # model wing area in square inches
WC = 5.5        # wing chord in inches
WI = 4          # wing incidence in degrees
WH = 3.0        # wing height in inches
W = 0.070       # model weight in ounces
PCW = 0.40      # ratio of stab area to wing area
TA = 17.0       # distance from 25% wing chord to 25% stab chord
```

The **TA** parameter is the distance from the *Mean Aerodynamic Chord* of the wing to the *MAC* of the tail.

import numpy as np wing_incidence = np.linspace(0,12,13) wing_height = np.linspace(0,3,7) center_of_gravity = np.linspace(0.3,1.0,8) body_angle = np.linspace(-2,12,8) stab_size = np.linspace(20,50,7)

In [ ]:

```
air_density = 0.00119
```

Using Curve Data

In this study, we will be using digitized data points obtained from images of data in references. What we need is a way to use that data to produce a function that can give us values for any point in the range of the data, and possible a bit beyond both ends.

This is a common problem in the *data science* research world, and there many *Python* tools available for dealing with this. The **scipy** library has a useful function that uses *spline interpolation* that we can use here.

In [7]:

```
from scipy.interpolate import InterpolatedUnivariateSpline

def fit_curve(xp, yp):
    xi = np.array(xp)
    yi = np.array(yp)
    order = 1
    s = InterpolatedUnivariateSpline(xi, yi, k=order)
    return s
```

This function takes two lists of data points, and returns a function we can use to get values both inside and outside of the range of those points. The extrapolation will simply extend the curve found at the ends for some distance past those ends. Obviously, this should be used with caution.

With this new function, we can get two new functions that will let us easily access lift and drag coefficients for our use in later code.

In [8]:

```
CL = fit_curve(alpha, C_l)
CD = fit_curve(alpha, C_d)
```

Now, **CL** and **CD** are the names of functions we can use to get values for C_l and C_d. Let's try them out:

In [10]:

```
angle = 6
cl_w = float(CL(angle))
cd_w = float(CD(angle))
print(cl_w, cd_w)
```

```
0.3 0.014
```

Erbach's Power Function

Walt's program used very short variable names, based on this diagram of a simple indoor model:

Erbach Model

The first part of Walt's code calculates the flight velocity needed for level flight. This is based on the definition of the *lift coefficient*:

$$C_l = \frac{2L}{\rho u^2 S}$$

Where:

- **L** is the lift force
- **S** is a reference surface area
- **V** is the flight velocity
- ρ is the air density.

We can rearrange this equation to find the flight velocity:

$$V = \sqrt{\frac{2L}{\rho S C_l}}$$

This equation is the basis for the first part of Walt's code, which calculates the contributions to lift from both the wing and stab to determine the needed flight velocity. Time for some code!

Dimensional Analysis

When using these equations, we need to be sure our calculations use the right units! If we simply plug in numbers, we might get answers that make no sense. We have presented some values from Walt's program, and the units he was using were shown in comments. The *Python* **pint** library makes it easy to add dimensional units to our variables and make sure our calculated values are correct!

To demonstrate this, we need to initialize **pint** and create a *Unit Registry* gadget we can use to attach units to variables:

In [12]:
```
import pint
u = pint.UnitRegistry()
```

Now, **u** is a powerful thing that knows a huge amount about units! Let's try it out on the surface area:

In [14]:
```
# S_w is the surface area of the wing
S_w = 150.0 * u.inches ** 2
S_w.to('meters**2')
```

Out[14]: 0.096774 meter^2

Now that is pretty cool! I defined the area in square inches and asked for the value in square meters. All without doing anything to figure out that conversion.

Weight vs Mass

One of the biggest headaches in generating proper code here is making sure we distinguish between the mass of something and the weight of something.

From *Newton's Second Law*, we know that;

$$F = ma$$

Where **F** is the force (weight) of an object, **m** is the *mass* of that object, and **g** is the acceleration due to gravity. To get the mass of something, we divide its weight by **g**.

In [194…
```
(1 * u.ounce).to_base_units()
```

Out[194…
```
0.028349523125000005 kilogram
```

Now we can define a few dimensional model values for our work:

In [251…
```
# WT = model weight in ounces
WT = 0.070 * u.ounces
SW = 150.0 * u.inches**2 # wing surface area
# PCW is the stab size as a percentage of wing area
PCW = 0.4
SS = SW * PCW #  stab surface area
WH = 3.0 * u.inch
```

We can calculate the mass of the model using Newton's law:

In [252…
```
WT.to_base_units()
```

Out[252…
```
0.0019844666187500007 kilogram
```

In [253…
```
MT = WT / u.gravity
MT.to_base_units()
```

Out[253…

0.00020235927852528648 kilogram second^2/meter

The density of something is the mass per unit volume. From Walt's code, we see that he used this value for the density of air:

Flight Velocity

Here is the line from Walt's code that calculates the flight velocity:

VE = (WT/(0.00119 *SW* (C_LW) + PCW * CL_s)) ** 0.5

From this line, it looks like the number in the code is actually $\rho/2$.

In [254…
```
density = 0.00119 * 2 * u.slugs/u.ft**3 # air density from program
```

```
density.to_base_units()
```

Out [254… 1.2266015877758076 kilogram/meter^3

In [255…

```
alpha_wing = 2.0
alpha_stab = -2.0
```

In [256…

```
f1 = (WT * 2 * u.gravity) / (density * SW)
VE = (f1 / (CL(alpha_wing) + PCW * CL(alpha_stab)) ) ** 0.5
VE.to('ft/sec')
```

Out [256… 3.9694209301872245 foot/second

This matches the value generated from Walt's code, as shown by *Figure 1* from his paper!

Component Lift and Drag

Now that we know the flight velocity, we can calculate the lift and drag generated by both the wing and stab. We will use Walt's variable names to match the model figure.

First, lets calculate the *dynamic pressure* $(\rho u^2)/2$ for the flight velocity:

In [319…

```
DP = density / u.gravity /2 * VE **2
DP.to_base_units()
```

Out [319… 0.09154551818218225 kilogram/meter^2

Pressure acting on an area produces a force. This will be used later.

In [320…

```
AWL = SW * DP * CL(alpha_wing)
AWL.to_base_units()
```

Out [320… 0.0017718451953125007 kilogram

In [332…

```
CTL = SS * DP * CL(alpha_stab)
CTL.to_base_units()
```

Out [332… 0.00021262142343750007 kilogram

In [333…

```
Lift = (AWL + CTL)
Lift.to_base_units()
```

Out [333… 0.0019844666187500007 kilogram

In [334…

```
WT.to_base_units()
```

Out [334… 0.0019844666187500007 kilogram

Our lift matches the weight of the model. Now we have equations we can use!

Next, let's calculate the drag force

In [331…
```
BWD = SS * DP * CD(alpha_wing)
DTD = SS * DP * CD(alpha_stab)
Drag = (BWD + DTD)
Drag.to('oz')
```

Out[331…
```
0.0020000000000000005 ounce
```

Level Flight Power

The power required to maintain level flight must equal the total drag. From Erbach's paper, that power should be this:

In [328…
```
power = 0.196 * u.inch * u.oz / u.sec
```

Dividing this by the flight velocity should give us the drag force., the drag force shound be:

In [329…
```
drag_e = (power/VE)
drag_e.to('oz')
```

Out[329…
```
0.004114789945586079 ounce
```

Well, we have a problem here. The units match, but the numbers are not agreeing. We will need to return to this issue later.

Pitching Moment

The *pitching moment* aC_m is defined by the following equation:

$$C_m = \frac{M}{qSc}$$

Where:

- **M** is the total moment (about the *CG*)
- **q** is the dynamic pressure
- **S** is a reference area
- **c** is the wing chord

We have the lift and drag values available, all we need to do this calculation is to figure out the moment arms (distances from the CG) for each of those forces)

Erbach's code uses these formulas for the distances defined in the model diagram above

In [280…
```
import math
CG = 0.5
WC = 5.5 * u.inch
```

```
TA = 17.0 * u.inch
DW = WC * (CG - 0.25)
EWLA = WH * math.sin(alpha_stab) + DW * math.cos(alpha_stab)
FWDA = WH * math.cos(alpha_stab) - DW * math.sin(alpha_stab)
GTLA = (TA - DW)*math.cos(alpha_stab)
HTDA = (TA - DW)* math.sin(alpha_stab)

JMWL = AWL * EWLA
KWMD = BWD * FWDA
LMTL = -CTL * GTLA
MWTD = DTD * HTDA

M = (JMWL + KWMD + LMTL + MWTD)/u.gravity
M.to('in oz')
```

Out[280… -0.17169460772962453 inch ounce

In [ ]:

```
This is close to the value Walt published, but again is not exact. More issues t
```

Parametric Study

The rest of the cod ein Walt's program set up a parametric study that produce "terrifying yards of tabulated data". Unfortunately, Walt had to manually convert this data into plots for his report. We are much too lazy for that sort of thins, so we will let *Python* and **matplotlib** do that for us.

For this example, we will recreate *Figure 2* and *Figure 3* drom Walt's report.

To generate this plot, we need to write a few functions that we cna use for a parametric study.

Fortunately, **numpy** will be a big help here!

In [281…

```
def VE_alpha(wa, sa):
    """Given an list of alpha values for wing and stab, return a list of velocit
    f1 = (WT * 2 * u.gravity) / (density * SW)
    return (f1 / (CL(wa) + PCW * CL(sa)) ) ** 0.5
```

lets set up a test case:

In [309…

```
stab_angles = np.linspace(-2,12,8)
stab_angles
```

Out[309…

```
array([-2.,  0.,  2.,  4.,  6.,  8., 10., 12.])
```

In [312…

```
wing_angles = stab_angles + 4
```

In [313…

```
wing_angles
```

Out[313…

```
array([ 2.,  4.,  6.,  8., 10., 12., 14., 16.])
```

Isn't **numpy** neat?

In [314…

```
VEset = VE_alpha(wing_angles, stab_angles)
```

In [315…

```
VEset.to('ft/sec')
```

Out[315…

```
Magnitude  [3.9694209301872245 3.4073313781819103 3.0476098333931185 2.800560168056
           02
            2.617866827702326 2.4668183349749864 2.342856441594096 2.23588166179080
           5]

    Units  foot/second
```

Next we generate functions that will calculate the lift and drag sts from the velocity sets.

In [335…

```
def lift_alpha(V, wa, sa):
    DP = density / u.gravity /2 * V **2
    AWL = SW * DP * CL(wa)
    CTL = SS * DP * CL(sa)
    return (AWL + CTL)

def drag_alpha(V, wa, sa):
    DP = density / u.gravity /2 * V **2
    BWD = SS * DP * CD(wa)
    DTD = SS * DP * CD(sa)
    return (BWD + DTD)
```

In [337…

```
lift_set = lift_alpha(VEset, wing_angles, stab_angles)
lift_set
```

Out[337…

```
Magnitude  [0.07000000000000002 0.05157894736842106 0.041263157894736856
            0.03484444444444445 0.03044660194174758 0.027034482758620693
            0.02438569206842924 0.022209631728045326]

    Units  ounce
```

In [338…

```
drag_set = drag_alpha(VEset, wing_angles, stab_angles)
drag_set
```

Out[338…

```
Magnitude  [0.0022500000000000003 0.0016578947368421052 0.0013263157894736846
            0.0011200000000000003 0.0009786407766990293 0.0008689655172413793
            0.0007838258164852255 0.0007138810198300282]

    Units  ounce
```

In [349…

```
def CMset(wa,sa, dp):
    a = SW * dp * CL(wa)
    c = SS * dp * CL(sa)
    b = SS * dp * CD(wa)
    d = SS * dp * CD(sa)
    CG = 0.5
    WC = 5.5 * u.inch
    TA = 17.0 * u.inch
    DW = WC * (CG - 0.25)
    EWLA = WH * np.sin(wa) + DW * np.cos(sa)
    FWDA = WH * np.cos(wa) - DW * np.sin(sa)
    GTLA = (TA - DW)*np.cos(sa)
    HTDA = (TA - DW)* np.sin(sa)

    JMWL = a * EWLA
    KWMD = b * FWDA
    LMTL = -c * GTLA
    MWTD = d * HTDA

    return (JMWL + KWMD + LMTL + MWTD)/u.gravity
```

In [350…

```
DPset = density / u.gravity /2 * VEset **2
CMvals = CMset(wing_angles, stab_angles, DPset)
```

In [351…

```
CMvals
```

Out[351…

```
Magnitude  [0.16929238819306713 -0.24799672293913405 0.030028044498012643
            0.2634192278931684 -0.2675519133869824 -0.04161888053748872
            0.31478818055404745 -0.2300868318104063]

    Units  inch ounce/standard_gravity
```

In [ ]: