5.1] $\alpha = 3.5^\circ$ $V = 30.5\,\text{m/s}$ $c = 3\,\text{ft} = 0.9\,\text{m}$

$C_m = 0.8$ $C_l = 1.0$

$L' = C_l \rho V^2 c = (1)(1.2\,\text{kg/m}^3)(30.5\,\text{m/s})^2(0.9\,\text{m})$

$\boxed{L' = 1004.7\,\text{N/m}}$

$M' = \frac{1}{2}\rho u_\infty^2 c^2 C_m$

$= \frac{1}{2}(1.2\,\text{kg/m}^3)(30.5\,\text{m/s})^2(0.9\,\text{m})^2(0.8)$

$\boxed{M' = 361.68\,\text{N}}$

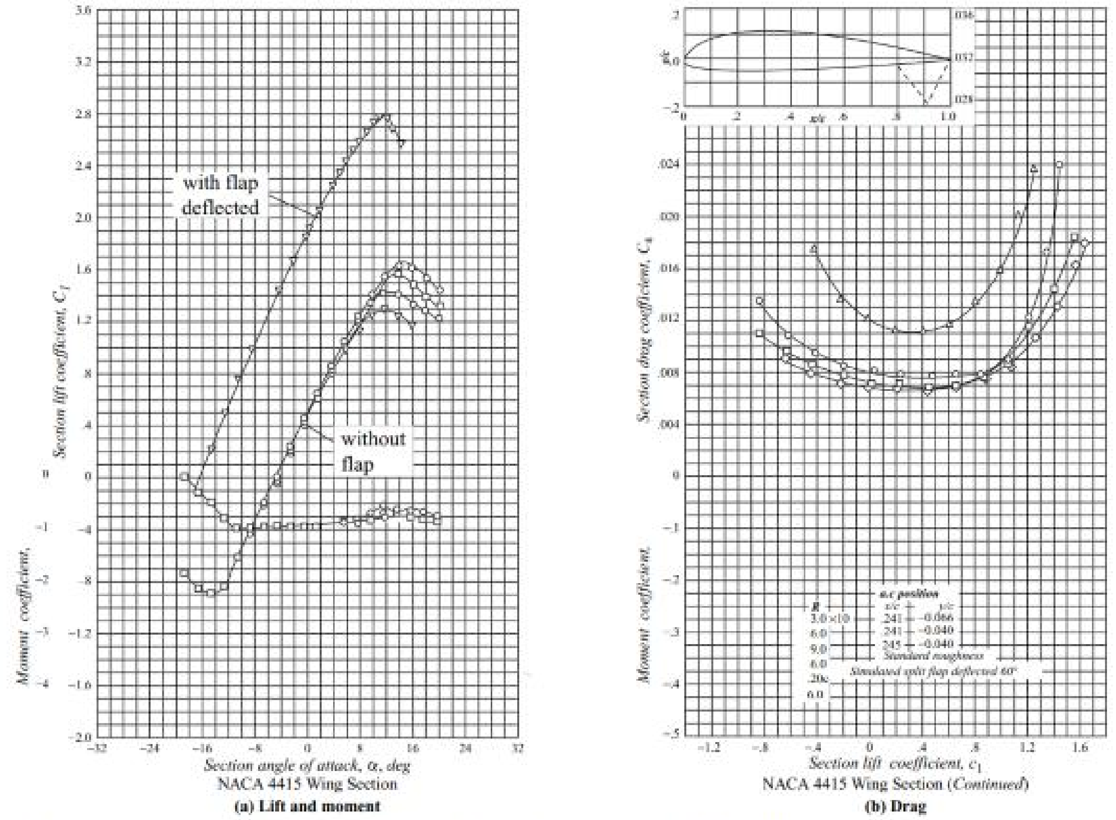


Figure 5.3. Two-dimensional airfoil behavior (NACA 4415, (Abbott and Van Doenhoff, 1959)).

5.4] $C_l = 2\pi\left[\alpha + \frac{1}{\pi}\int_0^{\pi}\frac{d\bar{z}}{dx}(\cos\theta - 1)\,d\theta\right]$

$$\frac{d\bar{z}}{dx} = \begin{cases} 0.2 - \dfrac{x}{2c} & \text{for } 0 \le \frac{x}{c} \le 0.4 \\ 0.09 - 0.22\dfrac{x}{c} & \text{for } 0.4 \le \frac{x}{c} \le 1.0 \end{cases}$$

go from $x \to \theta$: $x = \frac{c}{2}(1 - \cos\theta)$

$$\frac{d\bar{z}}{dx} = \begin{cases} 0.25\cos(\theta) - 0.05 & 0 \le \theta \le 1.37 \\ 0.11\cos(\theta) - 0.02 & 1.37 \le \theta \le \pi \end{cases}$$

$$\alpha_{L=0} = -\frac{1}{\pi}\int_0^{\pi}\frac{dz}{dx}(\cos\theta - 1)\,d\theta$$

$$= -\frac{1}{\pi}\int_0^{1.37}[0.25\cos(\theta) - 0.05](\cos\theta - 1)\,d\theta - \frac{1}{\pi}\int_{1.37}^{\pi}[0.11\cos(\theta) - 0.02](\cos\theta - 1)\,d\theta$$

(((−1) / pi) × integrate(((0.25 × cos(x radians)) − 0.05) × (cos(x radians) − 1), 0, 1.37)) − ((1 / pi) × (integrate(((0.11 × cos(x radians)) − 0.02) × (cos(x radians) − 1), 1.37, pi) × radians))

#9

$\boxed{\approx -4.007\ 140\ 169°}$

$C_l = 2\pi(\alpha - \alpha_{L=0})$

$C_l = 2\pi(\alpha + 4.007)$

$$\boxed{C_l = 6.28\alpha + 25.18}$$

5.5] $C_{m_{c/4}} = \frac{\pi}{4}(A_2 - A_1)$

$$A_2 = \frac{2}{\pi}\int_0^{\pi} \frac{d\bar{z}}{dx}\cos(2\theta)\,d\theta$$

$$A_1 = \frac{2}{\pi}\int_0^{\pi} \frac{d\bar{z}}{dx}\cos(\theta)\,d\theta$$

$$\frac{d\bar{z}}{dx} = \begin{cases} 0.25\cos(\theta) - 0.05 & 0 \le \theta \le 1.37 \\ 0.11\cos(\theta) - 0.02 & 1.37 \le \theta \le \pi \end{cases}$$

$$C_{m_{c/4}} = \frac{1}{2}\Big[\int_0^{1.37}(0.25\cos\theta - 0.05)\cos(2\theta)\,d\theta + \int_{1.37}^{\pi}(0.11\cos\theta - 0.02)\cos(2\theta)\,d\theta$$

$$- \int_0^{1.37}(0.25\cos\theta - 0.05)\cos(\theta)\,d\theta - \int_{1.37}^{\pi}(0.11\cos\theta - 0.02)\cos\theta\,d\theta\Big]$$

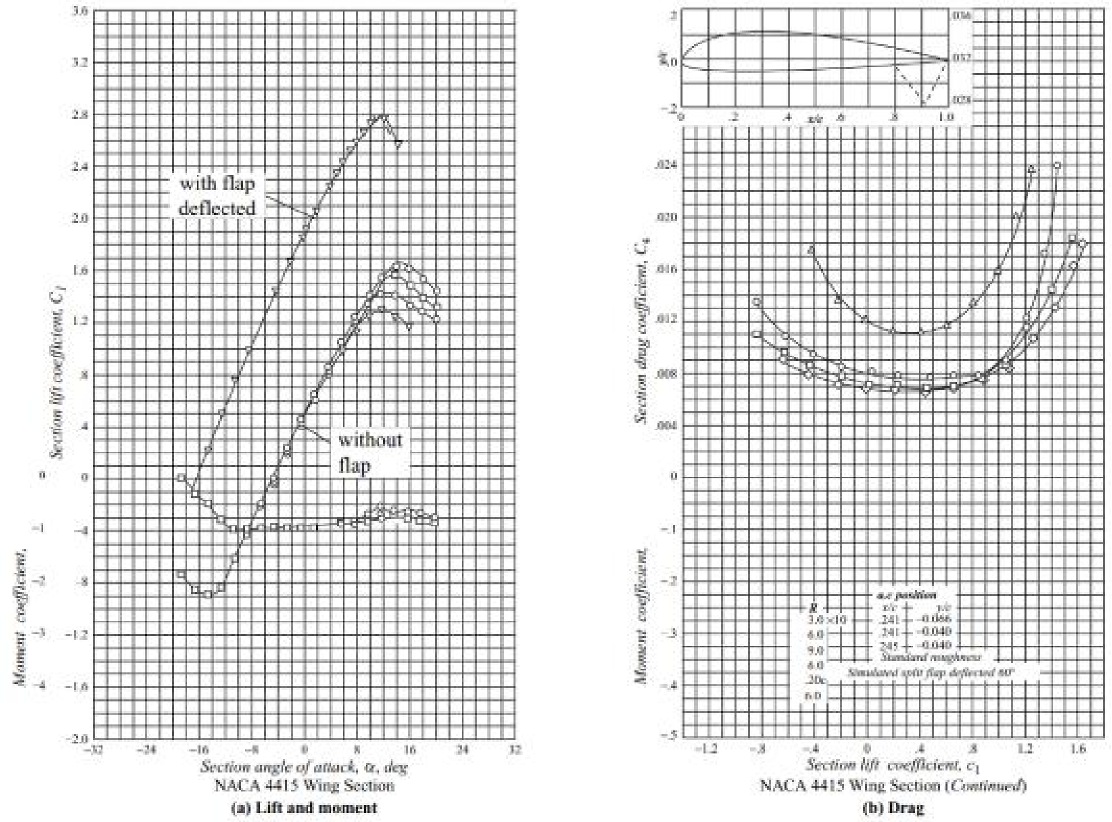


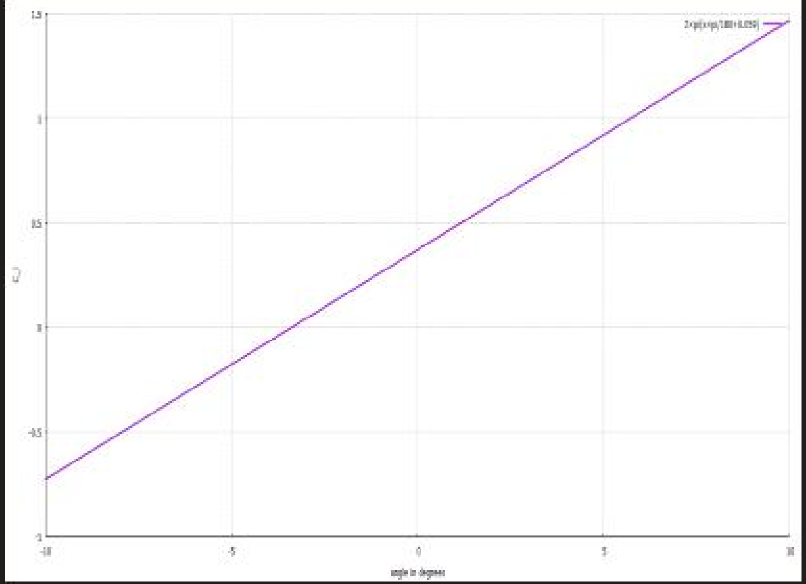
Figure 5.3. Two-dimensional airfoil behavior (NACA 4415, (Abbott and Van Doenhoff, 1959)).

```
(1 / 2) × (integrate(((0.25 × cos(x radians)) − 0.05) × cos((2 × x) radians), 0, 1.37) + integrate(((0.11 × cos(x radians)) − 0.02) × cos((2 × x) radians), 1.37, pi) −
integrate(((0.25 × cos(x radians)) − 0.05) × cos(x radians), 0, 1.37) − integrate(((0.11 × cos(x radians)) − 0.02) × cos(x radians), 1.37, pi))
#1
≈ −0.104 733 665 0
```

$$\boxed{C_{m_{c/4}} = -0.104}$$

$$\frac{x_{cp}}{c} = \frac{1}{4} - \frac{\pi}{4}\frac{(A_2 - A_1)}{C_\ell} = \frac{1}{4} - \frac{C_{mac}}{C_\ell}.$$

$$\left[C_\ell = 6.28\alpha + 25.18 \right] \longrightarrow$$



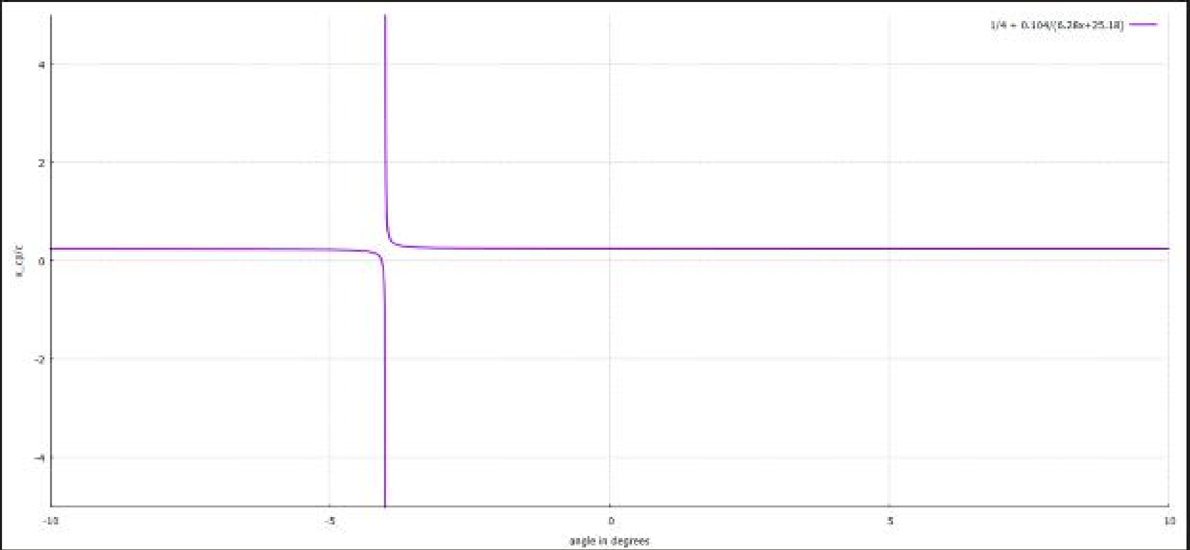
$$\frac{x_{cp}}{c} = \frac{1}{4} - \frac{\pi}{4C_\ell} \cdot \frac{2}{\pi}\left[\int_0^{1.37} (0.25\cos\theta - 0.05)\cos(2\theta)\,d\theta + \int_{1.37}^{\pi} (0.11\cos\theta - 0.02)\cos(2\theta)\,d\theta \right.$$

$$\left. - \int_0^{1.37} (0.25\cos\theta - 0.05)\cos(2\theta)\,d\theta - \int_{1.37}^{\pi} (0.11\cos\theta - 0.02)\cos\theta\,d\theta \right]$$

$$\frac{x_{cp}}{c} = \frac{1}{4} - \frac{C_{m_{c/4}}}{C_\ell}$$ oh... so that's what C_{mac} means haha

$$\frac{x_{cp}}{c} = \frac{1}{4} + \frac{0.104}{(6.28\alpha + 25.18)}$$

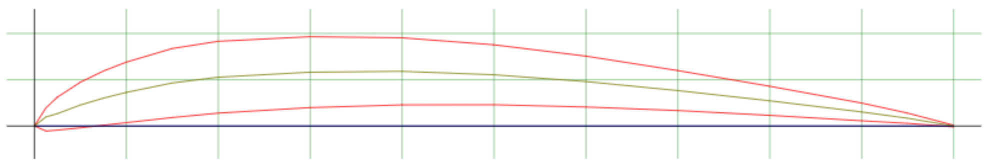
they're almost identical which demonstrates some drawbacks of thin airfoil theory!



1/4 + 0.104/(6.28x+25.18)
4
2
0
-2
-4
-10
-5
0
5
10
angle in degrees

Analyzing an albatross wing

by: Neil Sawhney



Import the mean camber line from http://airfoiltools.com/airfoil/details?airfoil=goe173-il

In [ ]:
```python
import pandas as pd

# Read the CSV file
df = pd.read_csv("albatross_foil-camber_line.csv")

# Display the first 5 rows
display(df.head())

x_foil = df["X(mm)"].tolist()
y_foil = df["Y(mm)"].tolist()
```

	X(mm)	Y(mm)
0	0.000	0.000000
1	1.241	0.978000
2	2.486	1.335821
3	4.978	2.261891
4	7.473	3.007482

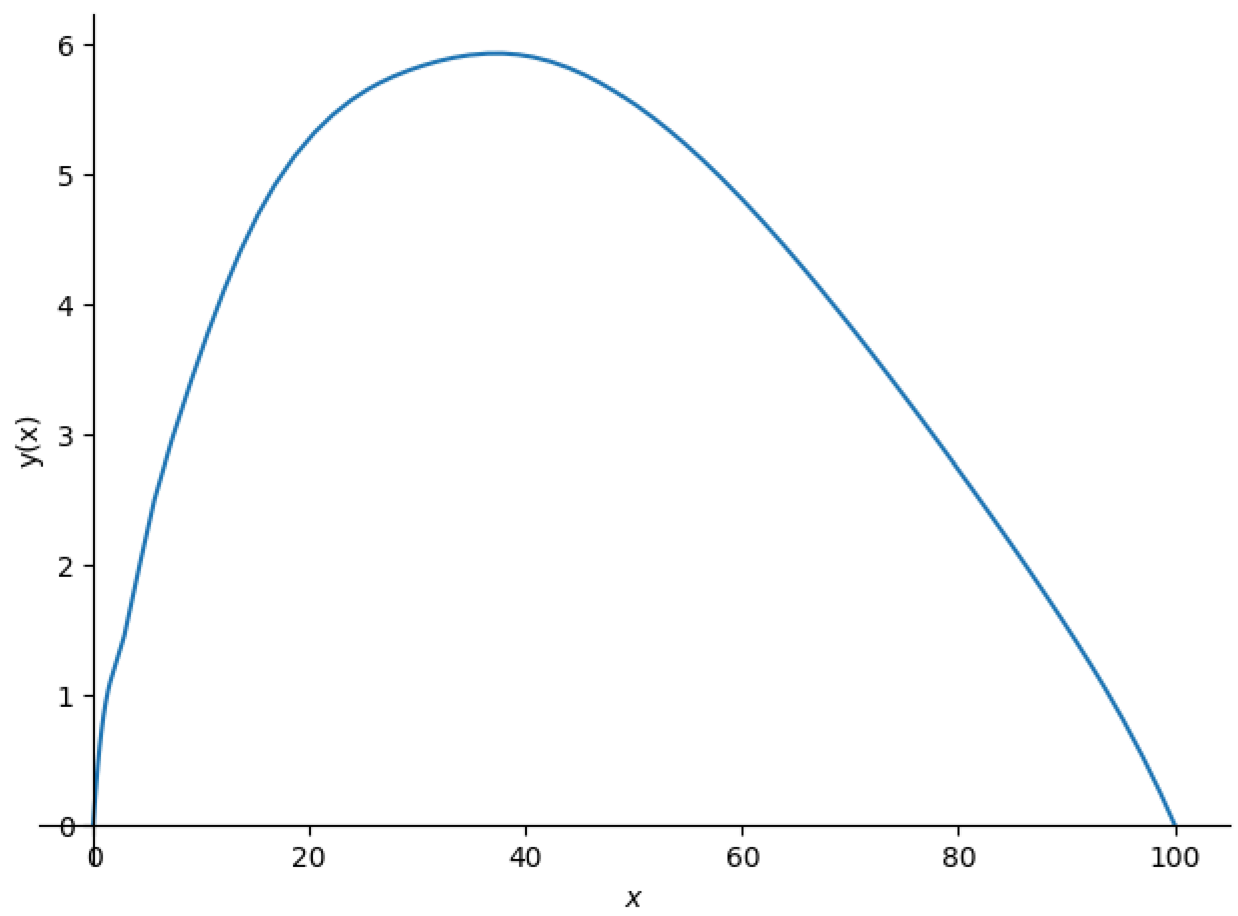
Interpolate the data into a function so that we can keep things analytical and symbolic

In [ ]:
```python
import sympy as sp
from IPython.display import Markdown

alpha, theta, x = sp.symbols("alpha, theta, x")
chord_length_data = 100 # mm

# Construct an interpolating polynomial for x_foil and y_foil
y_func_x = sp.interpolating_spline(3, x, x_foil, y_foil)
display(Markdown("$y(x) = " + sp.latex(y_func_x) + "$"))
sp.plot(y_func_x, (x, 0, chord_length_data), ylabel="y(x)")
```

$$y(x) = \begin{cases} 0.0757438198145068x^3 - 0.48369211108273x^2 + 1.27168442585302x & \text{for}\ x \geq 0 \wedge x \leq 2.486 \\ -0.0164005768176812x^3 + 0.203520799000128x^2 - 0.436726868612968x + 1.41570349268081 & \text{for}\ x \geq 2.486 \wedge x \leq 4.978 \\ 0.0051175805847793x^3 - 0.117831363648218x^2 + 1.16296419705049x - 1.23871721561009 & \text{for}\ x \geq 4.978 \wedge x \leq 7.473 \\ -0.000365930224799395x^3 + 0.00510346519172705x^2 + 0.244272221129586x + 1.0497444964089 & \text{for}\ x \geq 7.473 \wedge x \leq 9.968 \\ -0.000199278194540061x^3 + 0.000119902878852579x^2 + 0.293948370264324x + 0.884687211550524 & \text{for}\ x \geq 9.968 \wedge x \leq 14.962 \\ 0.000245238196808445x^3 - 0.0198326598632167x^2 + 0.592478614011156x - 0.604182624096177 & \text{for}\ x \geq 14.962 \wedge x \leq 19.958 \\ 0.00012274930637512x^3 - 0.0124987600374118x^2 + 0.446108641287747x + 0.369568014441756 & \text{for}\ x \geq 19.958 \wedge x \leq 29.956 \\ -3.90384183972492 \cdot 10^{-5}x^3 + 0.00204077921243152x^2 + 0.0105622035194419x + 4.71864437770423 & \text{for}\ x \geq 29.956 \wedge x \leq 39.956 \\ 2.88369943703519 \cdot 10^{-5}x^3 - 0.00609531076519523x^2 + 0.335647814665496x + 0.388937484720252 & \text{for}\ x \geq 39.956 \wedge x \leq 49.96 \\ 1.873961567553 \cdot 10^{-5}x^3 - 0.0045819156464153x^2 + 0.260038594531253x + 1.64808303068919 & \text{for}\ x \geq 49.96 \wedge x \leq 59.965 \\ 2.10662735920923 \cdot 10^{-5}x^3 - 0.00500046977231526x^2 + 0.28513719269084x + 1.14640388447584 & \text{for}\ x \geq 59.965 \wedge x \leq 69.972 \\ 6.97096097190038 \cdot 10^{-6}x^3 - 0.00204163812833513x^2 + 0.0781018248982628x + 5.97529680286998 & \text{for}\ x \geq 69.972 \wedge x \leq 79.98 \\ -1.90634019269676 \cdot 10^{-5}x^3 + 0.00420504690561922x^2 - 0.421508044117408x + 19.2948959108278 & \text{for}\ x \geq 79.98 \wedge x \leq 89.989 \\ -0.000149790157233174x^3 + 0.0394969568553698x^2 - 3.59739172858552x + 114.559761538028 & \text{for}\ x \geq 89.989 \wedge x \leq 100.0 \end{cases}$$



Out[ ]: `<sympy.plotting.plot.Plot at 0x7f35222ea110>`

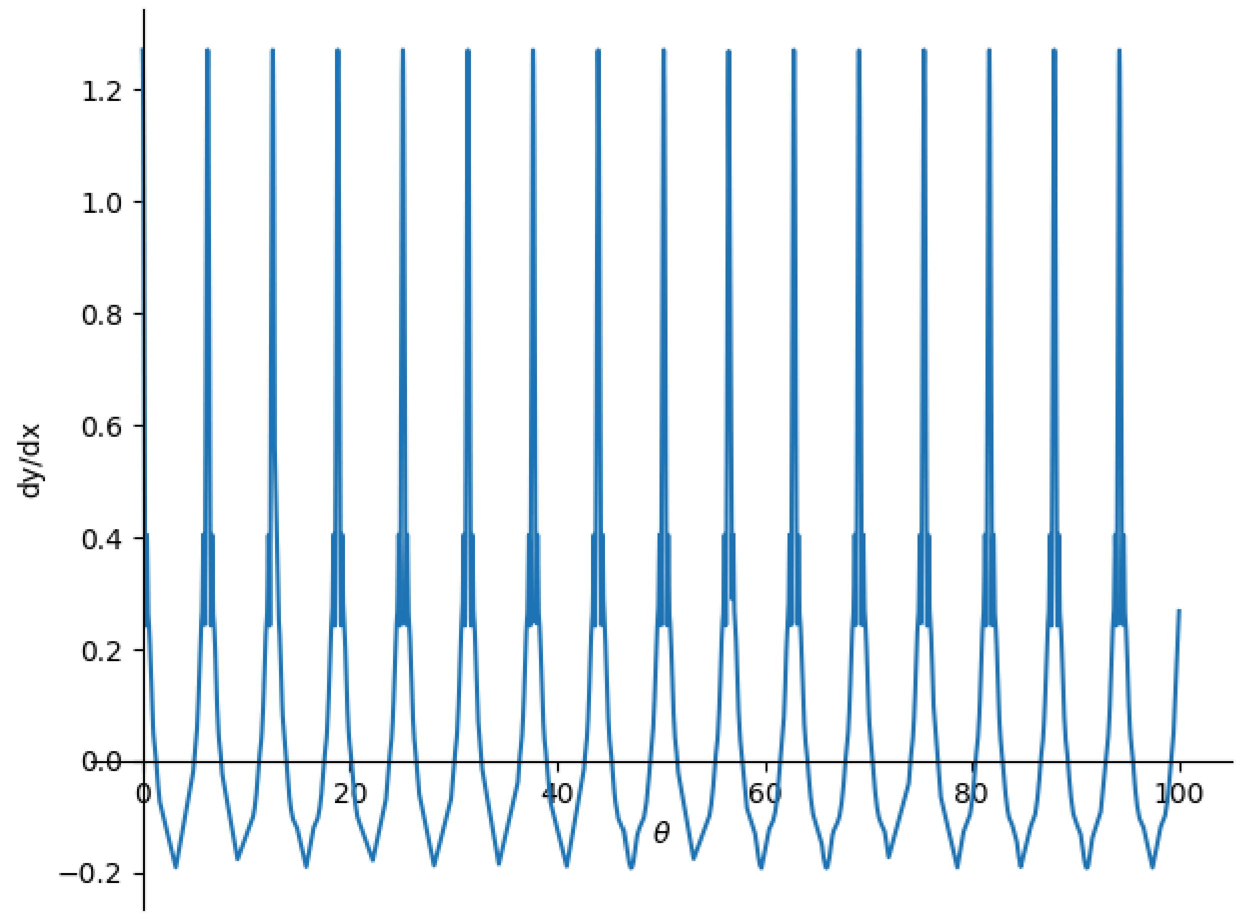
Now let's get derivatives and do a variable substitution to get the function in terms of θ instead of x

In [ ]:
```python
# Differentiate y_func_x with respect to x
dydx_func_x = y_func_x.diff(x)

# Variable substitution from x to theta
```

```python
dydx_func_theta = dydx_func_x.subs(x, chord_length_data / 2 * (1 - sp.cos(theta)))
display(Markdown("$\\frac{dy}{dx}(\\theta) = " + sp.latex(dydx_func_theta) + "$"))
sp.plot(dydx_func_theta, (theta, 0, chord_length_data), ylabel="dy/dx")
```

$$\frac{dy}{dx}(\theta) = \begin{cases} 568.078648608801(1-\cos(\theta))^2 + 48.369211108273\cos(\theta) - 47.09752668242 & \text{for } 50.0\cos(\theta) - 50.0 \geq -2.486 \wedge 50.0\cos(\theta) - 50.0 \leq 0 \\ -123.004326132609(1-\cos(\theta))^2 - 20.3520799000128\cos(\theta) + 19.9153530313998 & \text{for } 50.0\cos(\theta) - 50.0 \geq -4.978 \wedge 50.0\cos(\theta) - 50.0 \leq -2.486 \\ 38.3818543858447(1-\cos(\theta))^2 + 11.7831363648218\cos(\theta) - 10.6201721677713 & \text{for } 50.0\cos(\theta) - 50.0 \geq -7.473 \wedge 50.0\cos(\theta) - 50.0 \leq -4.978 \\ -2.74447668599546(1-\cos(\theta))^2 - 0.510346519172705\cos(\theta) + 0.754618740302291 & \text{for } 50.0\cos(\theta) - 50.0 \geq -9.968 \wedge 50.0\cos(\theta) - 50.0 \leq -7.473 \\ -1.49458645905046(1-\cos(\theta))^2 - 0.0119902878852579\cos(\theta) + 0.305938658149582 & \text{for } 50.0\cos(\theta) - 50.0 \geq -14.962 \wedge 50.0\cos(\theta) - 50.0 \leq -9.968 \\ 1.83928647606334(1-\cos(\theta))^2 + 1.98326598632167\cos(\theta) - 1.39078737231051 & \text{for } 50.0\cos(\theta) - 50.0 \geq -19.958 \wedge 50.0\cos(\theta) - 50.0 \leq -14.962 \\ 0.9206197978134(1-\cos(\theta))^2 + 1.24987600374118\cos(\theta) - 0.803767362453429 & \text{for } 50.0\cos(\theta) - 50.0 \geq -29.956 \wedge 50.0\cos(\theta) - 50.0 \leq -19.958 \\ -0.292788137979369(1-\cos(\theta))^2 - 0.204077921243152\cos(\theta) + 0.214640124762594 & \text{for } 50.0\cos(\theta) - 50.0 \geq -39.956 \wedge 50.0\cos(\theta) - 50.0 \leq -29.956 \\ 0.216277457777639(1-\cos(\theta))^2 + 0.609531076519523\cos(\theta) - 0.273883261854027 & \text{for } 50.0\cos(\theta) - 50.0 \geq -49.96 \wedge 50.0\cos(\theta) - 50.0 \leq -39.956 \\ 0.140547117566475(1-\cos(\theta))^2 + 0.45819156464153\cos(\theta) - 0.198152970110277 & \text{for } 50.0\cos(\theta) - 50.0 \geq -59.965 \wedge 50.0\cos(\theta) - 50.0 \leq -49.96 \\ 0.157997051940692(1-\cos(\theta))^2 + 0.500046977231526\cos(\theta) - 0.214909784540686 & \text{for } 50.0\cos(\theta) - 50.0 \geq -69.972 \wedge 50.0\cos(\theta) - 50.0 \leq -59.965 \\ 0.0522822072892529(1-\cos(\theta))^2 + 0.204163812833513\cos(\theta) - 0.12606198793525 & \text{for } 50.0\cos(\theta) - 50.0 \geq -79.98 \wedge 50.0\cos(\theta) - 50.0 \leq -69.972 \\ -0.142975514452257(1-\cos(\theta))^2 - 0.420504690561922\cos(\theta) - 0.00100335355548548 & \text{for } 50.0\cos(\theta) - 50.0 \geq -89.989 \wedge 50.0\cos(\theta) - 50.0 \leq -79.98 \\ -1.1234261792488(1-\cos(\theta))^2 - 3.94969568553698\cos(\theta) + 0.352303956951467 & \text{for } 50.0\cos(\theta) - 50.0 \geq -100.0 \wedge 50.0\cos(\theta) - 50.0 \leq -89.989 \end{cases}$$



Out[ ]: <sympy.plotting.plot.Plot at 0x7f35222dbe50>

Now we can get the lift coefficient!

In [ ]:
```python
# Calculate the lift coefficient
coeff_lift_albatross = (
    2
    * sp.pi
    * (
        alpha
        + (1 / sp.pi)
        * sp.integrate(dydx_func_theta * (sp.cos(theta) - 1), (theta, 0, sp.pi))
    )
).evalf()

# Display the equation
display(Markdown("$C_l = " + sp.latex(coeff_lift_albatross) + "$"))
```

$C_l = 6.28318530717959\alpha + 0.63959809779726$

Calculate lift for a speed v, with a span of b, chord_length of c, through air with density ρ

In [ ]:
```python
cruising_velocity, span_length, density, chord_length, coeff_lift = sp.symbols("u_{\\infty}, b, rho, c, C_l")

# Calculate the lift
lift = sp.Rational(1, 2) * density * cruising_velocity ** 2 * span_length * chord_length * coeff_lift
display(Markdown("$L = " + sp.latex(lift) + "$"))
```

$L = \frac{C_l b c \rho u_{\infty}^2}{2}$

Find the angle of attack at steady altitude (α_0)

In [ ]:
```python
albatross_weight = sp.symbols("W")

# Find the angle of attack at steady altitude
alpha_0 = sp.solve((lift - albatross_weight).subs(coeff_lift, coeff_lift_albatross), alpha, dict=True)[0][alpha]
display(Markdown("$\\alpha_0 = " + sp.latex(alpha_0) + "$"))
```

$\alpha_0 = \frac{0.318309886183791W}{b c \rho u_{\infty}^2} - 0.101795198856607$

Evaluating for properties of an Albatross moving at cruising speed

In [ ]:
```python
albatross_data = {
    "W": 8*9.81,  # N
    "u_{\\infty}": 20,  # m/s
    "b": 3,  # m
    "rho": 1.225,  # kg/m^3
    "c": 0.3,  # m
    "C_l": coeff_lift_albatross,
}


def radians_to_degrees(radians):
    return 180 / sp.pi * radians


def degrees_to_radians(degrees):
    return sp.pi / 180 * degrees


# Compute the angle of attack at steady altitude in degrees
alpha_0_albatross = radians_to_degrees(alpha_0).evalf(subs=albatross_data)
display(Markdown("$\\alpha_0 = " + sp.latex(round(alpha_0_albatross, 2)) + "^\\circ $"))
```

$\alpha_0 = -2.59^\circ$

Lets compare the lift to a small aircraft with a 50 mph cruise speed, 6 ft span, and 1 ft chord length, using the NACA 4412 airfoil at 2 degrees angle of attack

In [ ]:

```
aircraft_data = {
    "u_{\\infty}": 22.352,  # m/s
    "b": 1.8288,  # m
    "rho": 1.225,  # kg/m^3
    "c": 0.3048,  # m
    "C_l": 6.28 * degrees_to_radians(2) + 25.18,
}

# Compute the lift at a 2 degree angle of attack for the albatross
lift_albatross = lift.subs(albatross_data).subs(alpha, degrees_to_radians(2)).evalf()
display(Markdown("Albatross: $L = " + sp.latex(round(lift_albatross, 2)) + " N$"))

# Compute the lift at a 2 degree angle of attack for the aircraft
lift_aircraft = lift.subs(aircraft_data).subs(alpha, degrees_to_radians(2)).evalf()
display(Markdown("Aircraft: $L = " + sp.latex(round(lift_aircraft, 2)) + " N$"))
```

Albatross: $L = 189.39N$

Aircraft: $L = 4332.52N$