Python Files for Part 2 and 3 of Assignment 1

Kenneth Lønbæk
Christian Bak

Download and extract the *zip* file

4 min

- On *Learn* under *Week2: Aerodynamics and blades*, you will find the *zip* file: `assignment1_part2_3.zip`
- Download `assignment1_part2_3.zip` and extract the content
- Open the folder in VSCode
- Open the `example_1MW.py` and look at it (if you have time)

What?

1. **Give an overview of the Python files shared for Part 2 and 3 of Assignment 1**
2. **In-class exercise of applying the *Step-by-Step* method to a 1MW example and varying the input**

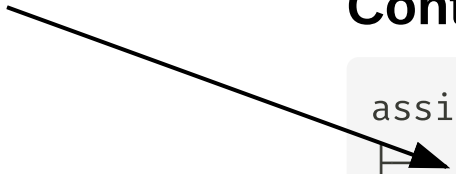
What is in the zip file?

Content of the *zip* file:

```
assignment1_part2_3.zip
├── overview.pdf
├── example_1MW.py
└── aero_design_functions.py
```

What is in the zip file?

These slides



Content of the *zip* file:

```
assignment1_part2_3.zip
├── overview.pdf
├── example_1MW.py
└── aero_design_functions.py
```

What is in the zip file?

These slides

Content of the *zip* file:

```
assignment1_part2_3.zip
├── overview.pdf
├── example_1MW.py
└── aero_design_functions.py
```

Script containing the inputs, running and plotting for the 1MW example design

This is the script that you should use as a basis for your 10MW design

What is in the zip file?

These slides

Content of the *zip* file:

```
assignment1_part2_3.zip
├── overview.pdf
├── example_1MW.py
└── aero_design_functions.py
```

Script containing the inputs, running and plotting for the 1MW example design

This is the script that you should use as a basis for your 10MW design

Script containing function to run the single-point design

You will not need to make changes in this file!

The 1MW design example (`example_1MW.py`)

- Runs a single point design for the 1MW example
- Uses the `aero_design_functions.py` (presented next)

```python
# %% Import modules
import matplotlib.pyplot as plt
import numpy as np
from aero_design_functions import get_design_functions_1MW, single_point_design


# Function for the absolute thickness vs span for the 35 m blade
def thickness(r, chord_root):
    """Absolute thickness [m] as a function of blade span [m] for 35-m blade"""
    p_edge = [
        9.35996e-8,
        -1.2911e-5,
        7.15038e-4,
        -2.03735e-2,
        3.17726e-1,
        -2.65357,
        10.2616,
    ]  # polynomial coefficients
    t_poly = np.polyval(p_edge, r)  # evaluate polynomial
```

The 1MW design example (`example_1MW.py`)

- Importing modules
- *Notice*: importing from `aero_design_functions.py`

```
# %% Import modules
import matplotlib.pyplot as plt
import numpy as np
from aero_design_functions import get_design_functions_1MW, single_point_design


# Function for the absolute thickness vs span for the 35 m blade
def thickness(r, chord_root):
    """Absolute thickness [m] as a function of blade span [m] for 35-m blade"""
    p_edge = [
        9.35996e-8,
        -1.2911e-5,
        7.15038e-4,
        -2.03735e-2,
        3.17726e-1,
        -2.65357,
        10.2616,
    ]  # polynomial coefficients
    t_poly = np.polyval(p_edge, r)  # evaluate polynomial
```

The 1MW design example (`example_1MW.py`)

- `get_design_function_1MW` : Function to get the design polynomials for the 1 MW example
- `single_point_design` : Main function to run the single-point-design

```python
# %% Import modules
import matplotlib.pyplot as plt
import numpy as np
from aero_design_functions import get_design_functions_1MW, single_point_design


# Function for the absolute thickness vs span for the 35 m blade
def thickness(r, chord_root):
    """Absolute thickness [m] as a function of blade span [m] for 35-m blade"""
    p_edge = [
        9.35996e-8,
        -1.2911e-5,
        7.15038e-4,
        -2.03735e-2,
        3.17726e-1,
        -2.65357,
        10.2616,
    ]  # polynomial coefficients
    t_poly = np.polyval(p_edge, r)  # evaluate polynomial
```

The 1MW design example (`example_1MW.py`)

- Setting inputs for `single_point_design`
- Comments breifly describing the parameter and units are added at the end of each line

```
22
23
24  # %% Inputs
25  R = 35  # Rotor radius [m]
26  tsr = 9.0  # Tip-Speed-Ratio [-]
27  r_hub = 1.0  # Hub radius [m]
28  r = np.linspace(r_hub, R - 0.1, 40)  # Rotor span [m]
29  chord_max = 3.0  # Maximum chord size [m]
30  chord_root = 2.7  # Chord size at the root [m]
31  t = thickness(r, chord_root)  # Absolute thickness [m]
32  B = 3  # Number of blades [#]
33  # Aero dynamic polar design functions and the values (t/c vs. cl, cd, aoa)
34  cl_scale = 1.0  # Change this value to scale the cl-values
35  cl_des, cd_des, aoa_des, tc_vals, cl_vals, cd_vals, aoa_vals = get_design_functions_1MW(
36      cl_scale
37  )
38
39
40  # %% Solving for the a single design
```

The 1MW design example (`example_1MW.py`)

- `thickness`: Function to compute *blade absolute thickness*. It defined above
- For your 10 MW redesign: compute `t` from the *AE-file* and scale it with you scaling-factor

```
22
23
24  # %% Inputs
25  R = 35  # Rotor radius [m]
26  tsr = 9.0  # Tip-Speed-Ratio [-]
27  r_hub = 1.0  # Hub radius [m]
28  r = np.linspace(r_hub, R - 0.1, 40)  # Rotor span [m]
29  chord_max = 3.0  # Maximum chord size [m]
30  chord_root = 2.7  # Chord size at the root [m]
31  t = thickness(r, chord_root)  # Absolute thickness [m]
32  B = 3  # Number of blades [#]
33  # Aero dynamic polar design functions and the values (t/c vs. cl, cd, aoa)
34  cl_scale = 1.0  # Change this value to scale the cl-values
35  cl_des, cd_des, aoa_des, tc_vals, cl_vals, cd_vals, aoa_vals = get_design_functions_1MW(
36      cl_scale
37  )
38
39
40  # %% Solving for the a single design
```

The 1MW design example (`example_1MW.py`)

- Function to compute blade absolute thickness from rotor span (`r`) and chord size at the root (`chord_root`)

```
5
6
7   # Function for the absolute thickness vs span for the 35 m blade
8   def thickness(r, chord_root):
9       """Absolute thickness [m] as a function of blade span [m] for 35-m blade"""
10      p_edge = [
11          9.35996e-8,
12          -1.2911e-5,
13          7.15038e-4,
14          -2.03735e-2,
15          3.17726e-1,
16          -2.65357,
17          10.2616,
18      ]  # polynomial coefficients
19      t_poly = np.polyval(p_edge, r)  # evaluate polynomial
20      t = np.minimum(t_poly, chord_root)  # clip at max thickness
21      return t
22
23
```

The 1MW design example (`example_1MW.py`)

- `get_design_functions_1MW` is defined in `aero_design_functions.py`
- `cl_scale` can be used to set the scale for C_l (you will use it for the in-class exercise at the end)

```
26  tsr = 9.0  # Tip-Speed-Ratio [-]
27  r_hub = 1.0  # Hub radius [m]
28  r = np.linspace(r_hub, R - 0.1, 40)  # Rotor span [m]
29  chord_max = 3.0  # Maximum chord size [m]
30  chord_root = 2.7  # Chord size at the root [m]
31  t = thickness(r, chord_root)  # Absolute thickness [m]
32  B = 3  # Number of blades [#]
33  # Aero dynamic polar design functions and the values (t/c vs. cl, cd, aoa)
34  cl_scale = 1.0  # Change this value to scale the cl-values
35  cl_des, cd_des, aoa_des, tc_vals, cl_vals, cd_vals, aoa_vals = get_design_functions_1MW(
36      cl_scale
37  )
38
39
40  # %% Solving for the a single design
41  chord, tc, twist, cl, cd, aoa, a, CLT, CLP, CT, CP = single_point_design(
42      r, t, tsr, R, cl_des, cd_des, aoa_des, chord_root, chord_max, B
43  )
44
```

The 1MW design example (`example_1MW.py`)

- Solving a single-point-design using input from above
- Using `single_point_design` defined in `aero_design_functions.py`

```
33  # Aero dynamic polar design functions and the values (t/c vs. cl, cd, aoa)
34  cl_scale = 1.0  # Change this value to scale the cl-values
35  cl_des, cd_des, aoa_des, tc_vals, cl_vals, cd_vals, aoa_vals = get_design_functions_1MW(
36      cl_scale
37  )
38
39
40  # %% Solving for the a single design
41  chord, tc, twist, cl, cd, aoa, a, CLT, CLP, CT, CP = single_point_design(
42      r, t, tsr, R, cl_des, cd_des, aoa_des, chord_root, chord_max, B
43  )
44
45  # %% Plotting design functions
46  tc_plot = np.linspace(0, 100, 100)
47  fig1, axs1 = plt.subplots(3, 1, num=1)
48
49  axs1[0].plot(tc_plot, cl_des(tc_plot), "k")
50  axs1[0].plot(tc_vals, cl_vals, "ok")
51  axs1[0].set_ylabel("$C_l$ [-]")
```

The 1MW design example (`example_1MW.py`)

- Plotting design functions

```
46  tc_plot = np.linspace(0, 100, 100)
47  fig1, axs1 = plt.subplots(3, 1, num=1)
48
49  axs1[0].plot(tc_plot, cl_des(tc_plot), "k")
50  axs1[0].plot(tc_vals, cl_vals, "ok")
51  axs1[0].set_ylabel("$C_l$ [-]")
52  axs1[0].set_xlim(0, 100)
53
54  axs1[1].plot(tc_plot, cd_des(tc_plot), "k")
55  axs1[1].plot(tc_vals, cd_vals, "ok")
56  axs1[1].set_ylabel("$C_d$ [-]")
57  axs1[1].set_xlim(0, 100)
58
59  axs1[2].plot(tc_plot, aoa_des(tc_plot), "k")
60  axs1[2].plot(tc_vals, aoa_vals, "ok")
61  axs1[2].set_ylabel(r"$\alpha$ [-]")
62  axs1[2].set_xlabel(r"$t/c$ [deg]")
63  axs1[2].set_xlim(0, 100)
64
```

The 1MW design example (`example_1MW.py`)

- Plotting Chord, Twist and Relative Thickness

```
68  fig2, axs2 = plt.subplots(3, 1, num=2, clear=True)
69
70  # Chord
71  axs2[0].plot(r, chord)
72  axs2[0].set_ylabel("Chord [m]")
73  axs2[0].set_xlim(0, R)
74
75  # Twist
76  axs2[1].plot(r, twist)
77  axs2[1].set_ylabel("Twist [deg]")
78  axs2[1].set_xlim(0, R)
79
80  # t/c
81  axs2[2].plot(r, tc)
82  axs2[2].set_ylabel("Rel. thickness [%]")
83  axs2[2].set_xlabel("Rotor span [m]")
84  axs2[2].set_xlim(0, R)
85
86  fig2.tight_layout()
```

The 1MW design example (`example_1MW.py`)

- Plotting Relative Thickness, Angle-of-Attack, Lift-Coefficient and Drag-Coefficient

```
88  # %% Plot r vs. t/c, aoa, cl, cd
89  fig3, axs3 = plt.subplots(2, 2, num=3, clear=True)
90
91  # t/c
92  axs3[0, 0].plot(r, tc)
93  axs3[0, 0].set_ylabel("t/c [%]")
94  axs3[0, 0].set_xlim(0, R)
95
96  # aoa
97  axs3[0, 1].plot(r, aoa)
98  axs3[0, 1].set_ylabel(r"$\alpha$ [deg]")
99  axs3[0, 1].set_xlim(0, R)
100 axs3[0, 1].yaxis.tick_right()
101 axs3[0, 1].yaxis.set_label_position("right")
102
103 # cl
104 axs3[1, 0].plot(r, cl)
105 axs3[1, 0].set_ylabel("$C_l$ [-]")
106 axs3[1, 0].set_xlabel("Span [m]")
```

The 1MW design example (`example_1MW.py`)

- Plotting Local-Thrust-Coefficient, Local-Power-Coefficient and Axial-Induction-Factor

```
119  # %% Plot r vs. CLT, CLP, a
120  fig4, axs4 = plt.subplots(3, 1, num=4, clear=True, figsize=(6.5, 5.5))
121
122  # Local-Thrust-Coefficient
123  axs4[0].plot(r, CLT)
124  axs4[0].axhline(y=8 / 9, ls="--", color="k", lw=1)
125  axs4[0].set_ylabel("Local thrust ($C_{LT}$) [-]")
126  axs4[0].set_ylim(0, 1.0)
127  axs4[0].set_xlim(0, R)
128
129  # Local-Power-Coefficient
130  axs4[1].plot(r, CLP)
131  axs4[1].axhline(y=16 / 27, ls="--", color="k", lw=1)
132  axs4[1].set_ylabel("Local Power ($C_{LP}$) [-]")
133  axs4[1].set_xlim(0, R)
134  axs4[1].set_ylim(-0.4, 0.6)
135
136  # Axial Induction
137  axs4[2].plot(r, a)
```

The Aero Design Function (`aero_design_functions.py`)

- Script containing functions used to make a *single point design*
- You will *not* need to make modification to the script!

```
1   import numpy as np
2   from lacbox.aero import max_twist, min_tc_chord, root_chord
3   from scipy.interpolate import PchipInterpolator
4   from scipy.optimize import brenth
5
6
7   def get_design_functions(i_des_funs):
8       """Gives the design functions for Cl [-], Cd [-], AoA [deg] as a function of Relative profile thickness (t/c) [
9       As well as the values used to construct the interpolation data.
10
11      Parameters
12      ----------
13      i_des_funs : int
14          Integer for selecting design functions (1-3)
15
16      Returns
17      -------
18      tuple
19
```

The Aero Design Function (`aero_design_functions.py`)

- Importing modules
- *Notice*: It uses the `lacbox`, which you are expected to have installed

```python
import numpy as np
from lacbox.aero import max_twist, min_tc_chord, root_chord
from scipy.interpolate import PchipInterpolator
from scipy.optimize import brenth


def get_design_functions(i_des_funs):
    """Gives the design functions for Cl [-], Cd [-], AoA [deg] as a function of Relative profile thickness (t/c) [
    As well as the values used to construct the interpolation data.

    Parameters
    ----------
    i_des_funs : int
        Integer for selecting design functions (1-3)

    Returns
    -------
    tuple

```

The Aero Design Function (`aero_design_functions.py`)

- Design functions for 1 MW example
- Returns functions for Cl, Cd and AoA as a function of relative thickness (as well a values used for the interpolation)

```
135 def get_design_functions_1MW(cl_scale=1.0):
136     """Returns the design functions for Cl [-], Cd [-], AoA [deg] as a function of Relative profile thickness (t/c)
137 
138     Parameters
139     ----------
140     cl_scale : float
141         Scale factor multiplied for the Cl-values. Default is `cl_scale=1.0` yielding the baseline cl_design.
142 
143     Returns
144     -------
145     tuple
146 
147         1. Lift coefficient function (`cl(tc)`)
148         2. Drag coefficient function (`cd(tc)`)
149         3. Angle-of-Attack function (`aoa(tc)`)
150         4. Values of relative airfoil profile thickness (`tc`)
151         5. Values of the lift coefficient (`cl`)
152         6. Values of the drag coefficient (`cd`)
153         7. Values of angle-of-attack (`aoa`)
```

The Aero Design Function (`aero_design_functions.py`)

- Accepts a single input `cl_scale`, which is used to scale the Cl-values
- You will use this in the assignment at the end

```
148             2. Drag coefficient function (`cd(tc)`)
149             3. Angle-of-Attack function (`aoa(tc)`)
150             4. Values of relative airfoil profile thickness (`tc`)
151             5. Values of the lift coefficient (`cl`)
152             6. Values of the drag coefficient (`cd`)
153             7. Values of angle-of-attack (`aoa`)
154
155         """
156         tc = [0, 15, 18, 24, 30, 36, 100, 105]
157         cl = np.array([0.9, 0.9, 0.8, 0.8, 0.7, 0.6, 0.0, 0.0]) * cl_scale
158         cd = [0.00850, 0.00850, 0.00604, 0.0107, 0.0139, 0.0155, 0.5, 0.5]
159         aoa = [5.0, 5.0, 4.3, 4.3, 4.0, 0.5, 0.0, 0.0]
160         return (
161             PchipInterpolator(tc, cl),
162             PchipInterpolator(tc, cd),
163             PchipInterpolator(tc, aoa),
164             np.array(tc[1:-2]),
165             np.array(cl[1:-2]),
166             np.array(cd[1:-2]),
```

The Aero Design Function (`aero_design_functions.py`)

- Design functions which you should use for your 10 MW redesign
- You should extract your own design point from the *PC-file* and chose the `i_des_funs` that matches it the best

```
7   def get_design_functions(i_des_funs):
8       """Gives the design functions for Cl [-], Cd [-], AoA [deg] as a function of Relative profile thickness (t/c) [
9       As well as the values used to construct the interpolation data.
10
11      Parameters
12      ----------
13      i_des_funs : int
14          Integer for selecting design functions (1-3)
15
16      Returns
17      -------
18      tuple
19
20          1. Lift coefficient function (`cl(tc)`)
21          2. Drag coefficient function (`cd(tc)`)
22          3. Angle-of-Attack function (`aoa(tc)`)
23          4. Values of relative airfoil profile thickness (`tc`)
24          5. Values of the lift coefficient (`cl`)
25          6. Values of the drag coefficient (`cd`)
```

The Aero Design Function (`aero_design_functions.py`)

- Main function for the *single-point-design*
- Computes the *ideal rotor design* as well as applying constraints like, maximum chord, root chord and maximum twist

```
727  def single_point_design(
728      r, t, tsr, R, cl_des, cd_des, aoa_des, chord_root, chord_max, B, a=1 / 3
729  ):
730      """Function for creating (chord, twist, relative-thickness) and evaluating the design (CP, CT, ..)
731
732      Parameters
733      ----------
734      r : np.ndarray
735          rotor span (including hub radius) [m]
736      t : np.ndarray
737          Absolute blade thickness [m]
738      tsr : float
739          tip-speed-ratio (`rotor_speed*R/V0`) [-]
740      R : float
741          Rotor radius [m]
742      cl_des : callable
743          function that returns Cl for a given t/c [%] (t/c=0-100) [-]
744      cd_des : callable
745          function that returns Cd for a given t/c [%] (t/c=0-100) [-]
```

The Aero Design Function (`aero_design_functions.py`)

- First solve for the ideal *Relative-Thickness* (`tc`) for the given input

```
788
789             10. CT : float
790                 Global Thrust Coefficient [-]
791
792             11. CP : float
793                 Global Power Coefficient [-]
794
795         """
796         # Solving for t/c
797         tc_ideal = solve_tc(cl_des, r, t, tsr, R, a, B)
798
799         # Compute chord and twist
800
801         # Getting ideal aero cl (before changing chord)
802         cl_ideal = cl_des(tc_ideal)  # [-]
803
804         # Chord [m]
805         chord_ideal = chord_fun(r, tsr, R, a, B, cl_ideal)  # calculating ideal chord
806         chord = root_chord(
```

The Aero Design Function (`aero_design_functions.py`)

- Then find the ideal chord (`chord_ideal`)

```
794
795       """
796       # Solving for t/c
797       tc_ideal = solve_tc(cl_des, r, t, tsr, R, a, B)
798
799       # Compute chord and twist
800
801       # Getting ideal aero cl (before changing chord)
802       cl_ideal = cl_des(tc_ideal)  # [-]
803
804       # Chord [m]
805       chord_ideal = chord_fun(r, tsr, R, a, B, cl_ideal)  # calculating ideal chord
806       chord = root_chord(
807           r, chord_ideal, chord_root, chord_max
808       )  # transition from ideal to root chord
809       chord = min_tc_chord(chord, t)  # maintain minimum t/c at the tip
810
811       # Updating t/c and polar design values
812       tc = t / chord * 100
```

The Aero Design Function (`aero_design_functions.py`)

- Then apply the chord constraints (maximum chord and chord and the root)

```
795         """
796         # Solving for t/c
797         tc_ideal = solve_tc(cl_des, r, t, tsr, R, a, B)
798 
799         # Compute chord and twist
800 
801         # Getting ideal aero cl (before changing chord)
802         cl_ideal = cl_des(tc_ideal)  # [-]
803 
804         # Chord [m]
805         chord_ideal = chord_fun(r, tsr, R, a, B, cl_ideal)  # calculating ideal chord
806         chord = root_chord(
807             r, chord_ideal, chord_root, chord_max
808         )  # transition from ideal to root chord
809         chord = min_tc_chord(chord, t)  # maintain minimum t/c at the tip
810 
811         # Updating t/c and polar design values
812         tc = t / chord * 100
813         cl = cl_des(tc)  # [-]
814         cd = cd_des(tc)  # [-]
```

The Aero Design Function (`aero_design_functions.py`)

- Update the *Relative-Thickness* (`tc`) with the new chord with constraints applied

```
804       # Chord [m]
805       chord_ideal = chord_fun(r, tsr, R, a, B, cl_ideal)  # calculating ideal chord
806       chord = root_chord(
807           r, chord_ideal, chord_root, chord_max
808       )  # transition from ideal to root chord
809       chord = min_tc_chord(chord, t)  # maintain minimum t/c at the tip
810
811       # Updating t/c and polar design values
812       tc = t / chord * 100
813       cl = cl_des(tc)  # [-]
814       cd = cd_des(tc)  # [-]
815       aoa_ideal = aoa_des(tc)  # [deg]
816
817       # Twist [deg]
818       twist_ideal = twist_deg_fun(r, tsr, R, a, aoa_ideal)  # [deg]
819       twist = max_twist(twist_ideal, 20)  # Limiting the twist angle at 20 degrees
820
821       # Updating a
822       a = solve_bem(r, tsr, R, chord, cl, B)
```

The Aero Design Function (`aero_design_functions.py`)

- Compute the ideal twist and apply the the maximum twist constraint

```
809       chord = min_tc_chord(chord, t)  # maintain minimum t/c at the tip
810
811       # Updating t/c and polar design values
812       tc = t / chord * 100
813       cl = cl_des(tc)  # [-]
814       cd = cd_des(tc)  # [-]
815       aoa_ideal = aoa_des(tc)  # [deg]
816
817       # Twist [deg]
818       twist_ideal = twist_deg_fun(r, tsr, R, a, aoa_ideal)  # [deg]
819       twist = max_twist(twist_ideal, 20)  # Limiting the twist angle at 20 degrees
820
821       # Updating a
822       a = solve_bem(r, tsr, R, chord, cl, B)
823
824       # Updating aoa
825       aoa = (
826           twist_deg_fun(r, tsr, R, a, 0) - twist
827       )  # Updating the design aoa for the constraint twist
```

The Aero Design Function (`aero_design_functions.py`)

- Solve the BEM equation with the chord constraint
- *Notice*: This BEM can not solve for the change in twist (would require data for AoA vs Cl)

```
813       cl = cl_des(tc)  # [-]
814       cd = cd_des(tc)  # [-]
815       aoa_ideal = aoa_des(tc)  # [deg]
816
817       # Twist [deg]
818       twist_ideal = twist_deg_fun(r, tsr, R, a, aoa_ideal)  # [deg]
819       twist = max_twist(twist_ideal, 20)  # Limiting the twist angle at 20 degrees
820
821       # Updating a
822       a = solve_bem(r, tsr, R, chord, cl, B)
823
824       # Updating aoa
825       aoa = (
826           twist_deg_fun(r, tsr, R, a, 0) - twist
827       )  # Updating the design aoa for the constraint twist
828
829       # Compute local- and global-thrust-coefficient as well as local- and global-power-coefficient
830       CLT = CLT_fun(r, tsr, R, a, B, cl, cd, chord)
831       CLP = CLP_fun(r, tsr, R, a, B, cl, cd, chord)
```

The Aero Design Function (`aero_design_functions.py`)

- Update AoA with the Twist constraint

```
816
817       # Twist [deg]
818       twist_ideal = twist_deg_fun(r, tsr, R, a, aoa_ideal)  # [deg]
819       twist = max_twist(twist_ideal, 20)  # Limiting the twist angle at 20 degrees
820
821       # Updating a
822       a = solve_bem(r, tsr, R, chord, cl, B)
823
824       # Updating aoa
825       aoa = (
826           twist_deg_fun(r, tsr, R, a, 0) - twist
827       )  # Updating the design aoa for the constraint twist
828
829       # Compute local- and global-thrust-coefficient as well as local- and global-power-coefficient
830       CLT = CLT_fun(r, tsr, R, a, B, cl, cd, chord)
831       CLP = CLP_fun(r, tsr, R, a, B, cl, cd, chord)
832       CT = CT_fun(r, R, CLT)
833       CP = CP_fun(r, R, CLP)
834       return chord, tc, twist, cl, cd, aoa, a, CLT, CLP, CT, CP
```

The Aero Design Function (`aero_design_functions.py`)

- Compute loading/power value: Local-Thrust/Power-Coefficient, Global-Thrust/Power-Coefficient

```
816
817       # Twist [deg]
818       twist_ideal = twist_deg_fun(r, tsr, R, a, aoa_ideal)  # [deg]
819       twist = max_twist(twist_ideal, 20)  # Limiting the twist angle at 20 degrees
820
821       # Updating a
822       a = solve_bem(r, tsr, R, chord, cl, B)
823
824       # Updating aoa
825       aoa = (
826           twist_deg_fun(r, tsr, R, a, 0) - twist
827       )  # Updating the design aoa for the constraint twist
828
829       # Compute local- and global-thrust-coefficient as well as local- and global-power-coefficient
830       CLT = CLT_fun(r, tsr, R, a, B, cl, cd, chord)
831       CLP = CLP_fun(r, tsr, R, a, B, cl, cd, chord)
832       CT = CT_fun(r, R, CLT)
833       CP = CP_fun(r, R, CLP)
834       return chord, tc, twist, cl, cd, aoa, a, CLT, CLP, CT, CP
```

The Aero Design Function (`aero_design_functions.py`)

- Return computed values

```
816
817       # Twist [deg]
818       twist_ideal = twist_deg_fun(r, tsr, R, a, aoa_ideal)  # [deg]
819       twist = max_twist(twist_ideal, 20)  # Limiting the twist angle at 20 degrees
820
821       # Updating a
822       a = solve_bem(r, tsr, R, chord, cl, B)
823
824       # Updating aoa
825       aoa = (
826           twist_deg_fun(r, tsr, R, a, 0) - twist
827       )  # Updating the design aoa for the constraint twist
828
829       # Compute local- and global-thrust-coefficient as well as local- and global-power-coefficient
830       CLT = CLT_fun(r, tsr, R, a, B, cl, cd, chord)
831       CLP = CLP_fun(r, tsr, R, a, B, cl, cd, chord)
832       CT = CT_fun(r, R, CLT)
833       CP = CP_fun(r, R, CLP)
834       return chord, tc, twist, cl, cd, aoa, a, CLT, CLP, CT, CP
```

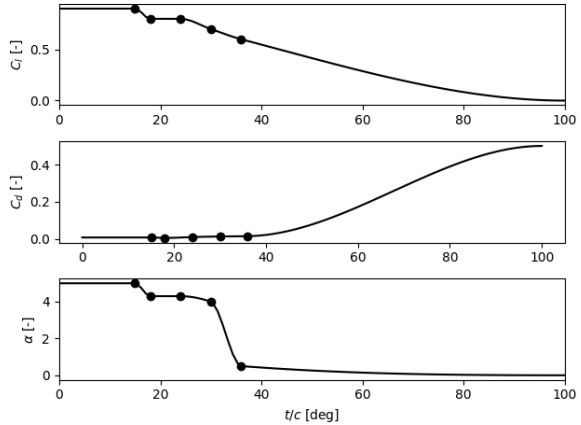
Exercise: Do the following modification in the `example_1MW.py`

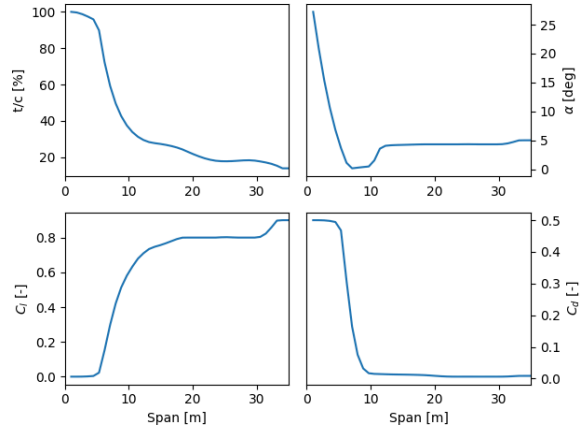
- Run the code and ensure that it generates the right plots (see plots on the next slide)

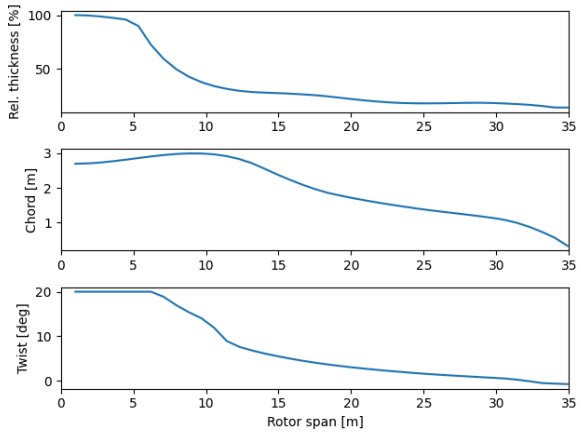
Answer the following questions:

- Step 4: Reduce the thickness of the entire blade $\rightarrow$ Multiply the entire thickness distribution (equation 13) with 0.9. What happened to the planform compared to the original planform, CP and CT?
- Step 6: Increase the design lift $\rightarrow$ Multiply the design lift distribution (equation 14) with 1.1. What happened to the planform compared to the original planform, CP and CT?
- Step 7: Decrease TSR $\rightarrow$ Decrease TSR (equation 10) to 8. What happened to the planform compared to the original planform, CP and CT?

Default output: `example_1MW.py`







C_T=0.782, C_P=0.456