Airfoil2BECAS: A preprocessor for the cross-section analysis software BECAS

Robert David Bitsche*
Technical University of Denmark, Department of Wind Energy

October 20, 2014

1. Concept

Airfoil2BECAS is a set of python functions that allow for the generation of input files for the analysis of a wind turbine blade cross-section with BECAS [Blasques, 2011]. A list of airfoil coordinates, layup information and material data are required as input. A 2D-mesh of the cross-section and the corresponding material and orientation assignments in BECAS format are generated. Figure 1 shows an example 2D mesh generated by airfoil2BECAS.py.

The shape of the airfoil is defined by a list of nodes as shown in Figure 2(a). The spacing between these nodes defines the circumferential element size of the final 2D mesh. The material is always placed on the left side when moving from low to high node numbers. The first and last node in the list of nodes must *not* coincide, as shown in Figure 2(a). The airfoil is automatically closed as shown in Figure 2(b).

An arbitrary number of shear webs can be defined as straight lines connecting two airfoil nodes as shown in Figure 2(b). These lines define the mid-surface of the shear webs. The individual shear webs are referred to as web 1, web 2, and so on.

Any number of nodes can be marked as "keypoints" (KP) as shown in Figure 2(c). The regions between these keypoints are referred to as region 1, region 2, etc. Layup information can be

*robi@dtu.dk

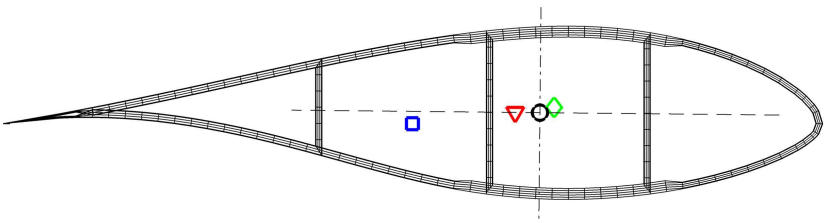


Figure 1: Example 2D mesh generated by airfoil2BECAS.py.

assigned to these regions. n keypoints define n regions on the airfoil. The last region (region 8 in Figure 2(c)) lies between the last and the first keypoint defined.

Any number of elements can be used to discretize the shell thickness. The minimum value, however, is the largest number of layers of different material anywhere in the cross-section.

As explained above, layup and thickness information is assigned to regions on the airfoil. At the boundary between two regions thickness discontinuities usually occur. In order to facilitate mesh generation, airfoil2BECAS.py removes these discontinuities by defining a continuous (node-based) thickness distribution. This is achieved by defining the thickness at the boundary of two regions as the smaller of the two thickness. This default behavior can be changed by defining "dominant regions". If dominant regions are defined, the thickness at region boundaries is governed by the dominant region. For a wind turbine blade it may be a good idea to define important load carrying parts (e.g. the caps) as dominant, because even small thickness changes of elements representing these parts may have a non-negligible effect on the cross section stiffness properties. As the representation of the composite layup by the elements next to region boundaries is only approximate, as a general rule, several elements should be used to represent each region circumferentially.

Technically, airfoil2BECAS creates a dummy Abaqus finite element shell model, writes the model to a .inp-file and calls shellexpander.py to process this file. See the shellexpander documentation for further details.

2. Use

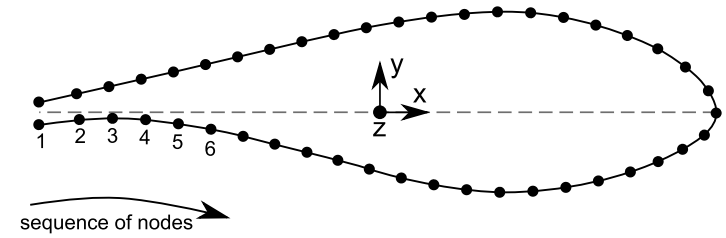
The use of airfoil2BECAS is demonstrated in the extensively commented file `example.py`.
To run the example type
`python example.py`
at the command prompt. The source code in the file `example.py` is also listed in Appendix A.

3. Limitations

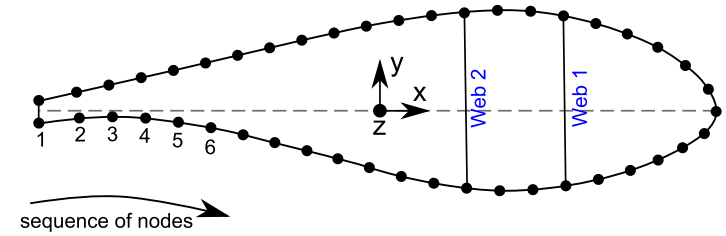
Any number of regions can be defined on the airfoil. However, only one region per shear web is available for layup assignment.

4. Software Requirements

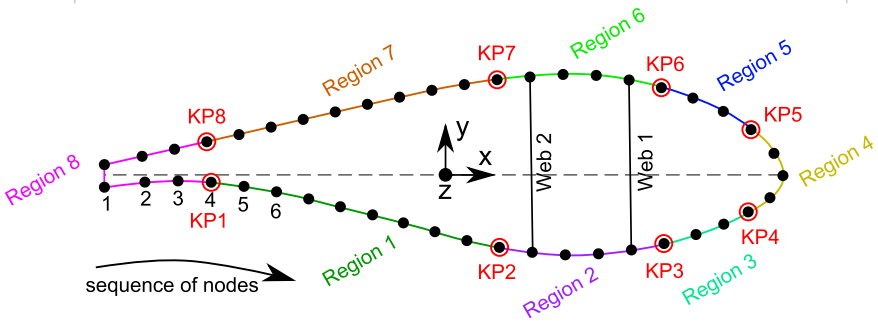
Shellexpander was written for Python 3.2 or newer and the NumPy module. Python can be downloaded from `http://www.python.org`. The NumPy module is available from `http://www.scipy.org`.



(a) List of nodes defining the airfoil. The first and last node must not coincide.



(b) The airfoil is automatically closed. Webs can be defined as straight lines connecting two airfoil nodes.



(c) n nodes are identified as keypoints defining n regions.

Figure 2: Input conventions for airfoil2BECAS.py.

5. Licensing

Airfoil2BECAS is distributed as part of BECAS [Blasques, 2011] and covered by the BECAS license. More information about BECAS is available at `http://www.becas.dtu.dk`

References

J. P. Blasques. User's manual for BECAS - a cross section analysis tool for anisotropic and inhomogeneous beam sections of arbitrary geometry. Technical Report Risø-R-1785(EN), Risø DTU, National Laboratory for Sustainable Energy, 2011.

A. Example file example.py

The use of airfoil2BECAS is demonstrated in the file `example.py` listed below.

```
#!/usr/bin/env python

# Airfoil2BECAS EXAMPLE
# ######################
# The example in this file demonstrates how the functions
# in airfoil2becas.py can be used to create BECAS input files
# for a wind turbine blade cross section.
#
# This program simply creates a "dummy" Abaqus input file
# and calls shellexpander.py to process this file.
# Python 3.x and the numpy module are required for this to work.
#
# Robert Bitsche, DTU Wind Energy, 2014
# Please report bugs to: robi@dtu.dk

# Load modules
# #############
import airfoil2becas, numpy

# Generate object for storing the program input
# ##############################################
print('Defining airfoil2becas input...')
inp = airfoil2becas.inp()

# Load airfoil coordinate list from file
# #######################################
inp.coords = numpy.loadtxt('example_airfoilcoords.dat', comments='#', delimiter=
    None)

# Scale the airfoil (loaded data was in unit coordinates)
# ########################################################
sf = 6.0
inp.coords = inp.coords * sf

# Define directory where the BECAS input files should be generated.
# If the directory already exists, it will be deleted!
# #####################################################################
inp.outputdir = 'BECAS_input'

# Define keypoints defining the boundaries between regions for
# composite layup assignment. Point numbering starts with 1, not 0.
# ###################################################################
inp.keypoints = [4, 6, 30, 42, 51, 55, 63, 75, 100, 102]

# Define the number of elements used to discretize the shell thickness.
# The minimum value is the largest number of layers of different material
# anywhere in the airfoil.
# ##############################################################################
inp.element_layers = 16

# Define Shear Webs:
# Shear webs are defined as straight lines connecting two airfoil
```

```
# nodes. Any number of shear webs can be defined. The syntax is:
# inp.shear_webs = [ [web1_from_node, web1_to_node],
#                    [web2_from_node, web2_to_node] ]
# The corresponding regions for layup assignment are:
# WEB01, WEB02, ...
inp.shear_webs = [ [ 8, 98],
                   [32, 73],
                   [40, 65] ]

# Define the number of elements in each shear web.
# For succesfull mesh generation the "height" of the elements
# representing the web must be larger than the thickness of the cap.
# Check the mesh at the intersection of the shear web and the cap.
# A list of integer numbers (one for each shear web) must be given.
# e.g. number_of_web_elements = [10, 12, 8]
# ########################################################################
inp.number_of_web_elements = [4, 18, 18]

# Composite Layup Defintion:
# The layup is defined as a list of lists using the following syntax:
# [ [layerthickness1, materialname1, orientationangle1],
#   [layerthickness2, materialname2, orientationangle2],
#   [layerthickness3, materialname3, orientationangle3] ]
# The first layer is the outer most layer.
# ########################################################################

inp.layup_of_reg['REGION01'] = [ [0.003, 'TRIAX', 0.0],
                                 [0.005, 'UNIAX', 0.0],
                                 [0.070, 'CORE',  0.0],
                                 [0.005, 'UNIAX', 0.0],
                                 [0.003, 'TRIAX', 0.0] ]

inp.layup_of_reg['REGION02'] = [ [0.003, 'TRIAX', 0.0],
                                 [0.070, 'CORE',  0.0],
                                 [0.003, 'TRIAX', 0.0] ]

inp.layup_of_reg['REGION03'] = [ [0.038, 'UNIAX', 0.0] ]

inp.layup_of_reg['REGION04'] = [ [0.0025, 'TRIAX', 0.0],
                                 [0.0002, 'UNIAX', 0.0],
                                 [0.0350, 'CORE',  0.0],
                                 [0.0002, 'UNIAX', 0.0],
                                 [0.0025, 'TRIAX', 0.0] ]

inp.layup_of_reg['REGION05'] = [ [0.0025, 'TRIAX', 0.0],
                                 [0.0010, 'UNIAX', 0.0],
                                 [0.0350, 'CORE',  0.0],
                                 [0.0010, 'UNIAX', 0.0],
                                 [0.0025, 'TRIAX', 0.0] ]

inp.layup_of_reg['REGION06'] = inp.layup_of_reg['REGION04']

inp.layup_of_reg['REGION07'] = inp.layup_of_reg['REGION03']

inp.layup_of_reg['REGION08'] = inp.layup_of_reg['REGION02']

inp.layup_of_reg['REGION09'] = inp.layup_of_reg['REGION01']
```

```
109
inp.layup_of_reg['REGION10'] = [ [0.003, 'TRIAX', 0.0],
111                                [0.005, 'UNIAX', 0.0],
                                   [0.040, 'CORE',  0.0],
113                                [0.005, 'UNIAX', 0.0],
                                   [0.003, 'TRIAX', 0.0] ]
115
inp.layup_of_reg['WEB01'] = [ [0.002, 'BIAX', 0.0],
117                             [0.025, 'CORE', 0.0],
                                [0.002, 'BIAX', 0.0] ]
119
inp.layup_of_reg['WEB02'] = [ [0.004, 'BIAX', 0.0],
121                             [0.050, 'CORE', 0.0],
                                [0.004, 'BIAX', 0.0] ]
123
inp.layup_of_reg['WEB03'] = inp.layup_of_reg['WEB02']
125


127 # Define Dominant Regions.
    # See the shellexpander documentation for details.
129 # Generally, it is a good idea to make struturally important regions
    # (like the spar caps) dominant.
131 # ###################################################################
    inp.dom_regions = [3, 7]
133
    # Material Parameters.
135 # The definition of an orthotropic material requires 9 elastic constants:
    # E1, E2, E3, nu12, nu13, nu23, G12, G13, G23.
137 # In the case of an isotropic material use:
    # E1 = E2 = E3 = E
139 # nu12 = nu13 = nu23 = nu
    # G12 = G13 = G23 = E/(2*(1+nu))
141 # 'rho' is the mass density
    # ###################################################################
143
    inp.material_properties['UNIAX'] = {
145 'E1':   40E9,
    'E2':   10E9,
147 'E3':   10E9,
    'nu12': 0.28,
149 'nu13': 0.28,
    'nu23': 0.4,
151 'G12':  4E+9,
    'G13':  4E+9,
153 'G23':  3.571E+9,
    'rho':  1900 }
155
    inp.material_properties['BIAX'] = {
157 'E1':   12E9,
    'E2':   12E9,
159 'E3':   10E9,
    'nu12': 0.5,
161 'nu13': 0.28,
    'nu23': 0.28,
163 'G12':  10E+9,
    'G13':  3.8E+9,
165 'G23':  3.8E+9,
```

```
'rho':  1890 }

inp.material_properties['TRIAX'] = {
'E1':   20E9,
'E2':   10E9,
'E3':   10E9,
'nu12': 0.5,
'nu13': 0.28,
'nu23': 0.28,
'G12':  7.5E+9,
'G13':  4E+9,
'G23':  4E+9,
'rho':  1860 }

inp.material_properties['CORE'] = {
'E1':   50E6,
'E2':   50E6,
'E3':   50E6,
'nu12': 0.4,
'nu13': 0.4,
'nu23': 0.4,
'G12':  17.857E6,
'G13':  17.857E6,
'G23':  17.857E6,
'rho':  80 }


# Create a finite element shell mesh
# ###################################
print('Creating finite element shell mesh...')
mesh = airfoil2becas.create_shell_mesh(inp)

# Write dummy input file for shellexapnder
# ########################################
print('Writintg input file for shellexapnder...')
airfoil2becas.write_shellexpander_input(inp,mesh)

# Call shellexpander to generate BECAS input files
# ################################################
print('Calling shellexpander...')
airfoil2becas.call_shellexpander(inp, mesh)
```

../src/example.py