PreVABS User's Manual

Version 1.4.1

Su Tian, Haodong Du, Xin Liu and Wenbin Yu

Sep 03, 2021

TABLE OF CONTENTS

CHAPTER

ONE

HOW TO RUN PREVABS

PreVABS is a command line based program which acts as a general-purpose preprocessor and postprocessor based on parametric inputs necessary for designing a cross section.

Download the examples package from cdmHUB (https://cdmhub.org/resources/1597/supportingdocs), and unpack it to any location.

1.1 Quick start

If you have already added the folder where you stored VABS, Gmsh and PreVABS to the system or user environment variable `PATH`, to execute PreVABS, you can open any command line tool (Command Prompt or PowerShell on Windows, Terminal on Linux), change directory to the root of the PreVABS package, and type the following command:

- On Windows:

```
prevabs -i examples\ex_airfoil\mh104.xml -h -v
```

- On Linux:

```
prevabs -i examples/ex_airfoil/mh104.xml -h -v
```

The first option `-i` indicates the path and name for the cross section file (`ex_airfoil\mh104.xml` for this case). The second option `-h` indicates the analysis to compute cross-sectional properties (this analysis is also called homogenization), where meshed cross section will be built and VABS input file will be generated. The last option `-v` is for visualizing the meshed cross section.

PreVABS will read the parametric input files and generate the meshed cross section.

Once finished, PreVABS will invoke Gmsh, a tool for visualization, to show the cross section with the corresponding meshes, as shown in Fig. 1.1. Three files are generated in the same location at this moment, a VABS input file `mh104.sg`, a Gmsh geometry file `mh104.geo`, a Gmsh mesh file `mh104.msh`, and a Gmsh option file `mh104.opt`. The geometry file is used to inspect errors when meshing cannot be accomplished. The latter three files are generated only when visualization is needed.

Then user can run VABS using the generated input file.

Note: PreVABS and Gmsh are free and open source. The source codes of PreVABS and Gmsh are available on cdmHUB at https://cdmhub.org/resources/1597. You can make changes to both codes by modifying its source codes. However, VABS is a commercial code and you need to request the code and a valid license from AnalySwift (http://analyswift.com/).

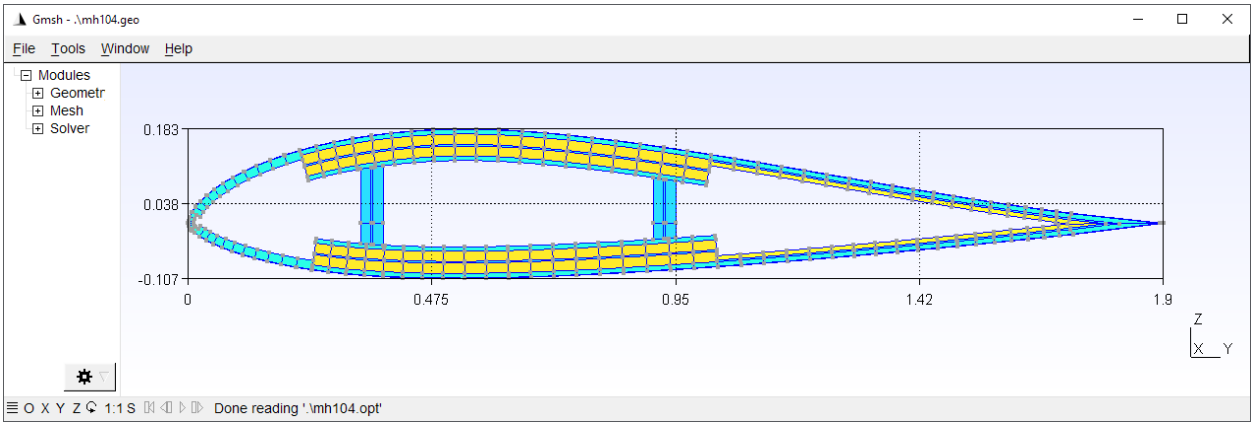

Figure 1.1: Cross section with meshes generated by PreVABS and visualized by Gmsh. Example: `examples\ex_airfoil\mh104.xml`.

1.2 Command line options

PreVABS is executed using command `prevabs` with other options. If no option is given, a list of available arguments will be printed on the screen.

```
prevabs -i <main_input_file_name.xml> [options]
```

Table 1.1: Command line options

Option	Description
-h	Build the cross-section for homogenization
-d	Read 1D beam analysis results and update VABS/SwiftComp input file for dehomogenization
-fi	Initial failure indices and strength ratios
-f	Initial failure strength analysis (SwiftComp only)
-fe	Initial failure envelope (SwiftComp only)
-vabs	Use VABS format (Default)
-sc	Use SwiftComp format
-int	Run integrated solver (VABS only)
-e	Run standalone solver
-v	Visualize meshed cross section for homogenization or contour plots of stresses and strains after recovery
-debug	Debug mode

Note: After version 1.4, the executable file name called by PreVABS is `VABS`.

Note: When VABS is called by PreVABS (using the option `-e`), the actual name of the executable used here is `VABSIII`.

1.3 Running cases

Some possible use cases are given below.

1.3.1 Case 1: Build cross section from parametric input files

```
prevabs -i <cross_section_file_name.xml> -h -v
```

In this case, parametric input files are prepared for the first time, and one may want to check the correctness of these files and whether the cross section can be built as designed. One may also want to try different meshing sizes before running the analysis.

1.3.2 Case 2: Carry out homogenization without visualization

```
prevabs -i <cross_section_file_name.xml> -h -e
```

The command will build the cross section model, generate the input, and run VABS to calculate the cross-sectional properties, without seeing the plot, since visualization needs extra computing time and resources. One can also make modifications to the design (change the parametric inputs) and do this step repeatedly. If you already have generated the input file *cross_section_vabs.dat*, and want to only run VABS, you can invoke VABS directly using `VABS` *cross_section_vabs.dat*.

Since PreVABS 1.4 and VABS version 4.0, a dynamic link library of VABS is provided. Users can run the cross-sectional analysis using the library instead of the standalone executable file. This will remove the time cost by writing and reading the VABS input file, and reducing the total running time. The command is:

```
prevabs -i <cross_section_file_name.xml> -h -vabs -int
```

1.3.3 Case 3: Recover 3D stress/strain and plot

```
prevabs -i <cross_section_file_name.xml> -d -e -v
```

After getting the results from a 1D beam analysis, one may want to find the local strains and stresses of a cross section at some location along the beam. This command will let PreVABS read those results, update the VABS input file, carry out recovery analysis, and finally draw contour plots in Gmsh (Fig. 1.2). An example of the recover analysis can be found in *this example*.

Note: Before any recovery run, a homogenization (with option `-h`) run must be carried out first for a cross section file. In other words, the file *cross_section.dat.opt* must be generated before the recovery run. Besides, results from the 1D beam analysis need to be added into the *cross_section.xml* file. Preparation of this part of data is explained in Section: *Specifications for recovery*.

Note: Plotted data are the nodal strains and stresses in the global coordinate system.

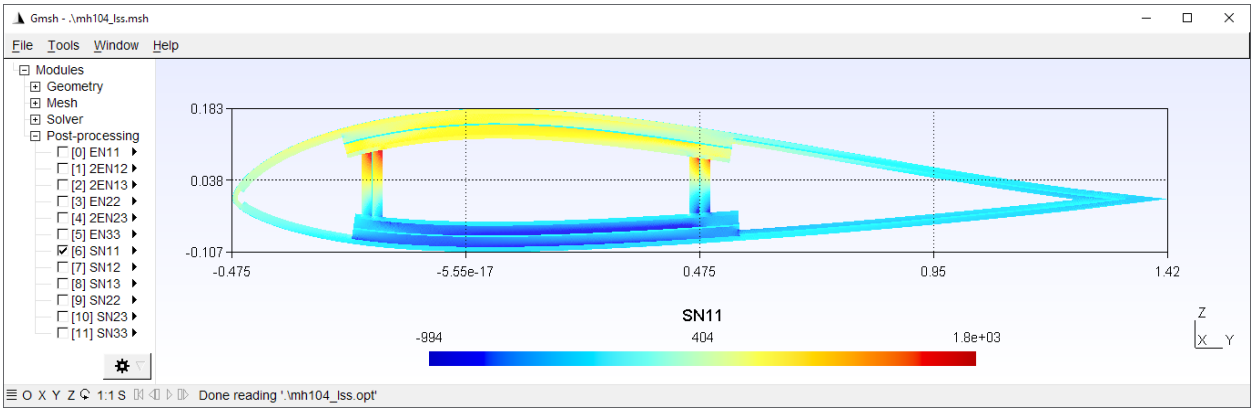

Figure 1.2: Visualization of strains and stresses in Gmsh.

CHAPTER

TWO

TUTORIAL

This tutorial provides a walkthrough for how to prepare input files for PreVABS given a cross section design.

2.1 The design of a cross section

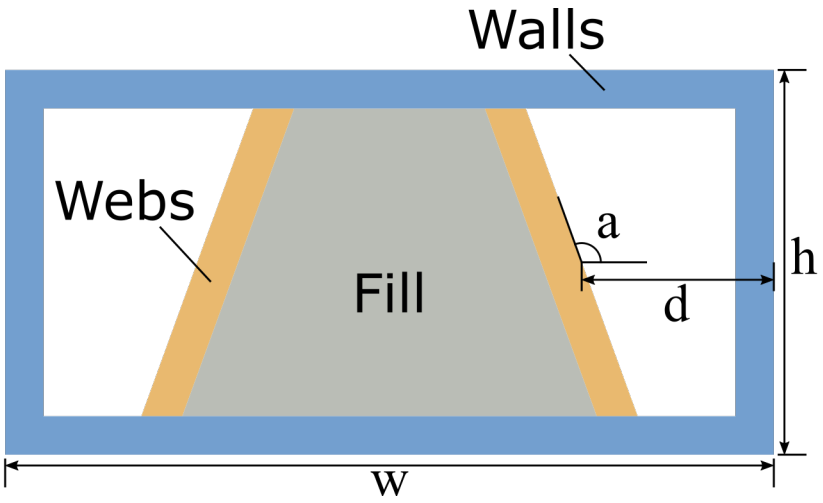

Figure 2.1: A box-beam cross section.

The box-beam cross section shown in Fig. 2.1 will be used in this tutorial. All four walls and two webs are made from composite laminates. The two webs are symmetric about a middle vertical line. They are inclined such that the upper portion of both webs are closer to each other than the lower portion. The central space enclosed by the top and bottom walls and two webs is filled with an isotropic material.

The overall shape is described using four parameters, with the following values:

- The width $w = 4$ m;
- The height $h = 2$ m;
- The distance $d = 1$ m;
- The angle $a = 100°$.

The materials used in this cross section are listed in Table 2.1 and Table 2.2. The layups used in this cross section are listed in Table 2.3.

Table 2.1: Isotropic material properties

Name	Density	E	ν
	$\mathrm{kg/m^3}$	10^3 Pa	
m0	1.00	25.00	0.30

Table 2.2: Orthotropic material properties

Name	Density	E_1	E_2	E_3	G_{12}	G_{13}	G_{23}	ν_{12}	ν_{13}	ν_{23}
	10^3 $\mathrm{kg/m^3}$	GPa	GPa	GPa	GPa	GPa	GPa			
m1	1.86	37.00	9.00	9.00	4.00	4.00	4.00	0.28	0.28	0.28
m2	1.83	10.30	10.30	10.30	8.00	8.00	8.00	0.30	0.30	0.30

Table 2.3: Layups

Component	Name	Material	Ply thickness	Orientation	Number of plies
			m	degree	
Walls	layup_1	m1	0.02	45	1
		m1	0.02	-45	1
		m2	0.05	0	1
		m1	0.02	0	2
Webs	layup_2	m2	0.05	0	1
		m1	0.02	0	3
		m2	0.05	0	1

2.2 Steps of preparing input files

In PreVABS, a cross section is composed of components, each of which can either be a laminate or a fill. The laminate type component is composed of segments with different layups. For a realistic cross section, there may be several different ways to decompose it, which may need different information in the input files to build the model.

Hence, the first step is to decide a way of decomposition of the cross section.

For this example, the first step is trivial. The cross section can be decomposed into three components: the walls, the webs, and the fill.

At the same time, the order of creating components should also be decided. This is done by figuring out the dependence relationships between components. For this example, the exact shape of each web depends on the shape of the walls, and the shape of the fill depends on both the webs and the walls. Then the order of creation should be: first creating the walls, then the webs, and at last the fill, as shown in Fig. 2.2.

Figure 2.2: Order of components creation.

The next step is to prepare various input files based on all design parameters listed above. In this example, all files are in the XML format. A brief summary of these input files is listed below.

- A baseline file (*baselines.xml*), storing definitions of geometric elements.

- A material file (*MaterialDB.xml*), storing definitions of materials and laminae.
- A layup file (*layups.xml*), storing definitions of layups.
- A main cross section file (*box.xml*), storing definitions of components and other configurations of modeling and analysis.

For this tutorial, all files can have arbitrary file names and be placed at any working directory, except the material database, which must be named as *MaterialDB.xml* and placed at the same location as where the PreVABS executable is.

Another option is to use a local file in the working directory with an arbitrary name storing the material properties. The requirement of using this local file is to explicitly provide the material file name in the main input file. Please check Section: *Other input settings*.

2.2.1 Prepare geometric elements

As shown in Fig. 2.3, seven points are used to define the shape of the cross section. Points p1 to p4 define the walls, p5 and p6 define the webs, and p0 indicates the space which should be filled with some material. The origin of the coordinate system is placed at the centroid of the rectangular. Based on the design parameters w, h and d, coordinates of all points are found and listed in Table 2.4.

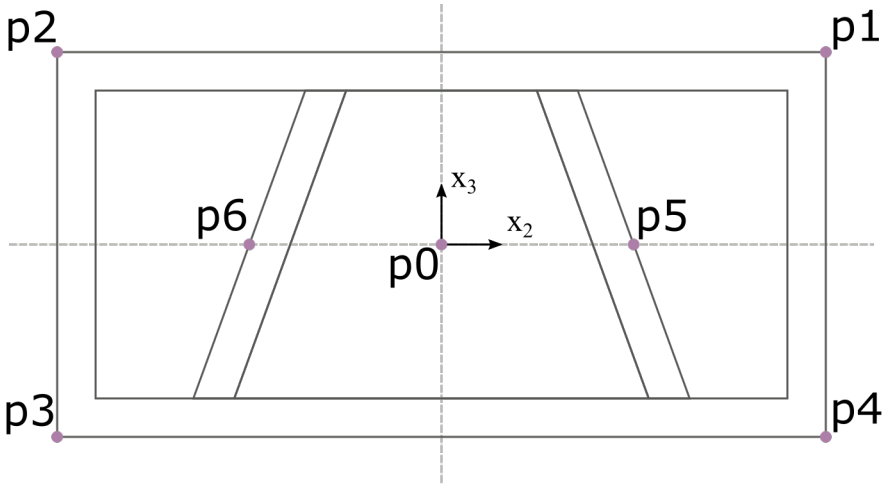

Figure 2.3: Key points defining the shape of the cross section.

Table 2.4: Key points

Name	Coordinate
p0	(0, 0)
p1	(2, 1)
p2	(-2, 1)
p3	(-2, -1)
p4	(2, -1)
p5	(1, 0)
p6	(-1, 0)

Base lines are created based on key points defined above. As shown in Fig. 2.4, three lines are created. Line 1 is defined by connecting the four points (p1 -> p2 -> p3 -> p4 -> p1). Line 2 and 3 are defined by a single point with an incline angle. For line 2, it is the point p5 with an angle of 100 degrees. For line 3, it is the point p6 with an angle of 80 degrees.

The direction of each base line is important. It is related with how the laminate is created for each segment, and how the local coordinate system is defined for each element.

The completed input file for geometry is shown in Listing 2.1.

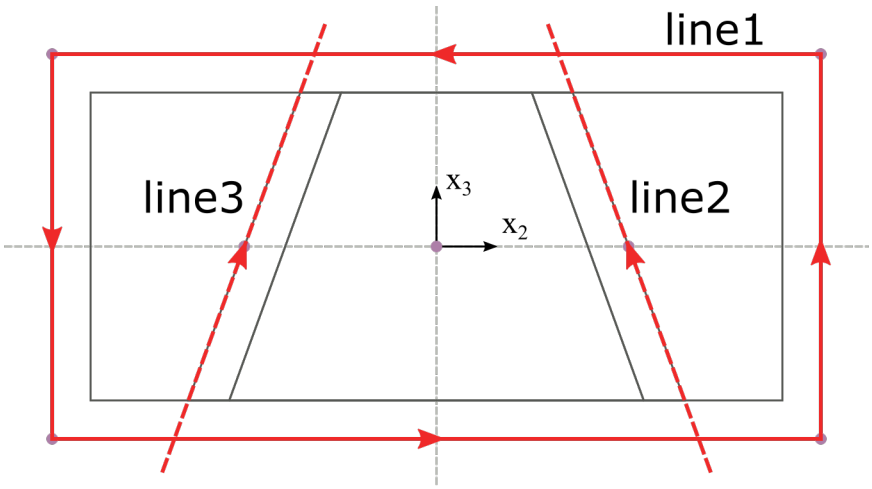

Figure 2.4: Base lines defining the shape of the cross section.

Listing 2.1: Input file for geometric elements (*baseline.xml*).

```
<baselines>
  <basepoints>
    <point name="p0">0  0</point>
    <point name="p1">2  1</point>
    <point name="p2">-2  1</point>
    <point name="p3">-2  -1</point>
    <point name="p4">2  -1</point>
    <point name="p5">1  0</point>
    <point name="p6">-1  0</point>
  </basepoints>
  <baseline name="line1" type="straight">
    <points>p1,p2,p3,p4,p1</points>
  </baseline>
  <baseline name="line2" type="straight">
    <point>p5</point>
    <angle>100</angle>
  </baseline>
  <baseline name="line3" type="straight">
    <point>p6</point>
    <angle>80</angle>
  </baseline>
</baselines>
```

2.2.2 Prepare materials and layups

Material data are stored in the material database. As stated above, this file must be named as *MaterialDB.xml* and placed at the directory where the PreVABS executable is. The format of this file is shown in Listing 2.2. Arrangement of data under the `<elastic>` element are different for different material types, which can be isotropic, orthotropic, or anisotropic.

Another part of the file is the lamina data. A lamina is a unique combination of material and ply thickness. For this example, according to the layup table (:Table 2.3) given above, there are two laminae. One has material m1 and thickness 0.02, and another one has material m2 and thickness 0.05. It is the lamina, instead of the material, that will be used to define each layer of the layup.

Listing 2.2: Input file for materials (*MaterialDB.xml*).

```
<materials>
  <material name="m0" type="isotropic">
    <density>1.0</density>
    <elastic>
      <e>25.0e3</e>
      <nu>0.3</nu>
    </elastic>
  </material>
  <!-- ============================================================ -->
  <material name="m1" type="orthotropic">
    <density>1.86E+03</density>
    <elastic>
      <e1>3.70E+10</e1>
      <e2>9.00E+09</e2>
      <e3>9.00E+09</e3>
      <g12>4.00E+09</g12>
      <g13>4.00E+09</g13>
      <g23>4.00E+09</g23>
      <nu12>0.28</nu12>
      <nu13>0.28</nu13>
      <nu23>0.28</nu23>
    </elastic>
  </material>
  <lamina name="la_m1_002">
    <material>m1</material>
    <thickness>0.02</thickness>
  </lamina>
  <!-- ============================================================ -->
  <material name="m2" type="orthotropic">
    <density>1.83E+03</density>
    <elastic>
      <e1>1.03E+10</e1>
      <e2>1.03E+10</e2>
      <e3>1.03E+10</e3>
      <g12>8.00E+09</g12>
      <g13>8.00E+09</g13>
      <g23>8.00E+09</g23>
      <nu12>0.30</nu12>
      <nu13>0.30</nu13>
      <nu23>0.30</nu23>
    </elastic>
  </material>
  <lamina name="la_m2_005">
    <material>m2</material>
```

(continues on next page)

(continued from previous page)

```
    <thickness>0.05</thickness>
  </lamina>
</materials>
```

Layup information is stored in a separate file. Based on the layup table, the input file can be prepared with the format shown in Listing 2.3. The order of the layers defines the laying sequence from the base line. The number in the `<layer>` element stands for the orientation. If there is a colon, then the number behind it stands for the number of plies in this layer. If there is no number at all, then by default this layer has only one ply with orientation of 0 degree.

Listing 2.3: Input file for layups (*layups.xml*).

```
<layups>
  <layup name="layup1">
    <layer lamina="la_m1_002">45</layer>
    <layer lamina="la_m1_002">-45</layer>
    <layer lamina="la_m2_005">0</layer>
    <layer lamina="la_m1_002">0:2</layer>
  </layup>
  <layup name="layup2">
    <layer lamina="la_m2_005"></layer>
    <layer lamina="la_m1_002">0:3</layer>
    <layer lamina="la_m2_005"></layer>
  </layup>
</layups>
```

2.2.3 Create components

There are two types of components in PreVABS, laminate and fill. For this example, based on the decomposition we did at the begining of this section, there are three laminate-type components and one fill-type component. Besides, the sequence of creating components is important as stated at the beginning of this section. This is defined by declaring which components are needed before creating the current one.

Each laminate-type component is composed of one or more segments. Each segment is a unique combination of a base line and a layup. The layup can be created on either side of the base line. This is controlled by an attribute `direction`, which can be either left or right. This can be understood as the left- or right-hand side when walking along the base line, as shown in Fig. 2.5.

As for the ordering, the walls should be completed first, since its inner boundary is needed to trim the base lines of both webs. Hence, each web component depends on the walls component, as shown in Listing 2.4.

Listing 2.4: Input elements for the laminate-type components

```
<component name="walls">
  <segment>
    <baseline>line1</baseline>
    <layup>layup1</layup>
  </segment>
</component>
<component name="web1" depend="walls">
  <segment>
    <baseline>line2</baseline>
    <layup>layup2</layup>
  </segment>
</component>
<component name="web2" depend="walls">
```

(continues on next page)

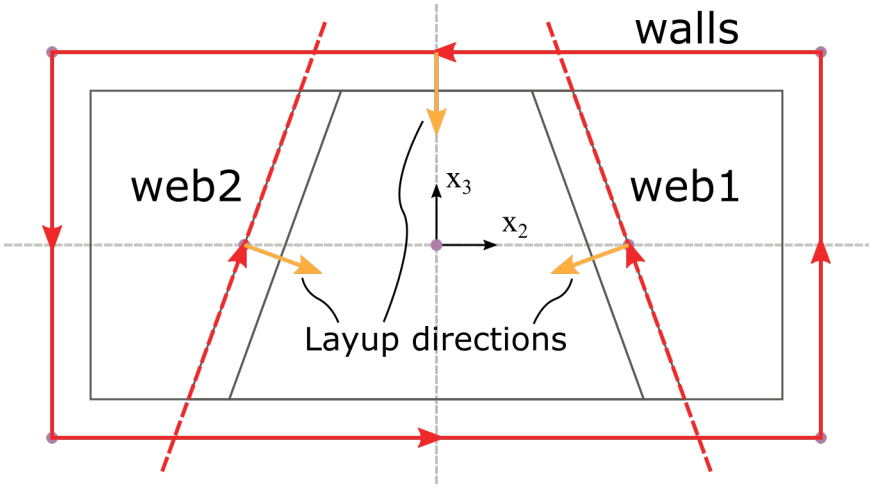

Figure 2.5: Layup directions for each segment.

(continued from previous page)

```
14    <segment>
15      <baseline>line3</baseline>
16      <layup direction="right">layup2</layup>
17    </segment>
18  </component>
```

The fill-type component is defined by a point and a material. Here, the point p0 is used to indicate that the material should be filled in the space in the middle enclosed by the walls and both webs. At the same time, the dependent components are also decided, as shown in Listing 2.5.

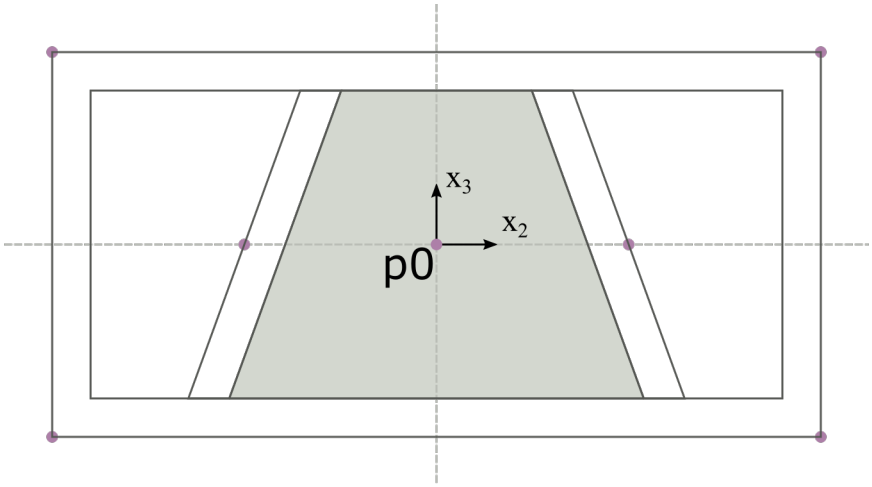

Figure 2.6: The fill-type component.

Listing 2.5: Input elements for the fill-type component

```
1  <component name="fill" type="fill" depend="walls,web1,web2">
2    <location>p0</location>
3    <material>m0</material>
4  </component>
```

2.2.4 Set other configurations and complete the main input file

Besides the definitions of components, the main input file of the cross section contains several other required or optional settings.

The first part is the <include> settings, which is required. This contains names of the base lines and layups files.

The second part is the <analysis> settings, which is optional. This contains configurations used by VABS for the cross-sectional analysis. For this example, the <model> setting is set to 1, which means that the Timoshenko beam model will be used and the 6x6 stiffness matrix will be calculated.

The last part is the <general> settings, which is optional. This part contains global configurations for the shape and meshing of the cross section. Here, the global mesh size is set to 0.02.

The completed main input file for the cross section is shown in Listing 2.6.

Listing 2.6: Input file for the cross section

```
<cross_section name="box">
  <include>
    <baseline>baselines</baseline>
    <layup>layups</layup>
  </include>
  <analysis>
    <model>1</model>
  </analysis>
  <general>
    <mesh_size>0.02</mesh_size>
  </general>
  <component name="walls">
    <segment>
      <baseline>line1</baseline>
      <layup>layup1</layup>
    </segment>
  </component>
  <component name="web1" depend="walls">
    <segment>
      <baseline>line2</baseline>
      <layup>layup2</layup>
    </segment>
  </component>
  <component name="web2" depend="walls">
    <segment>
      <baseline>line3</baseline>
      <layup direction="right">layup2</layup>
    </segment>
  </component>
  <component name="fill" type="fill" depend="walls,web1,web2">
    <location>p0</location>
    <material>m0</material>
  </component>
</cross_section>
```

2.3 Execution and results

Once all input files are prepared, the cross section can be created and homogenized using the following command:

```
prevabs -i box.xml -h -v -e
```

If everything works successfully, Gmsh will be called and the cross section will be plotted as shown in Fig. 2.7 (the display of element edges is turned off in the figure for clarity), and VABS homogenization analysis will be carried out and effective beam properties can be found in the file `box.sg.K`. The effective Timoshenko stiffness matrix is listed in Table 2.5.

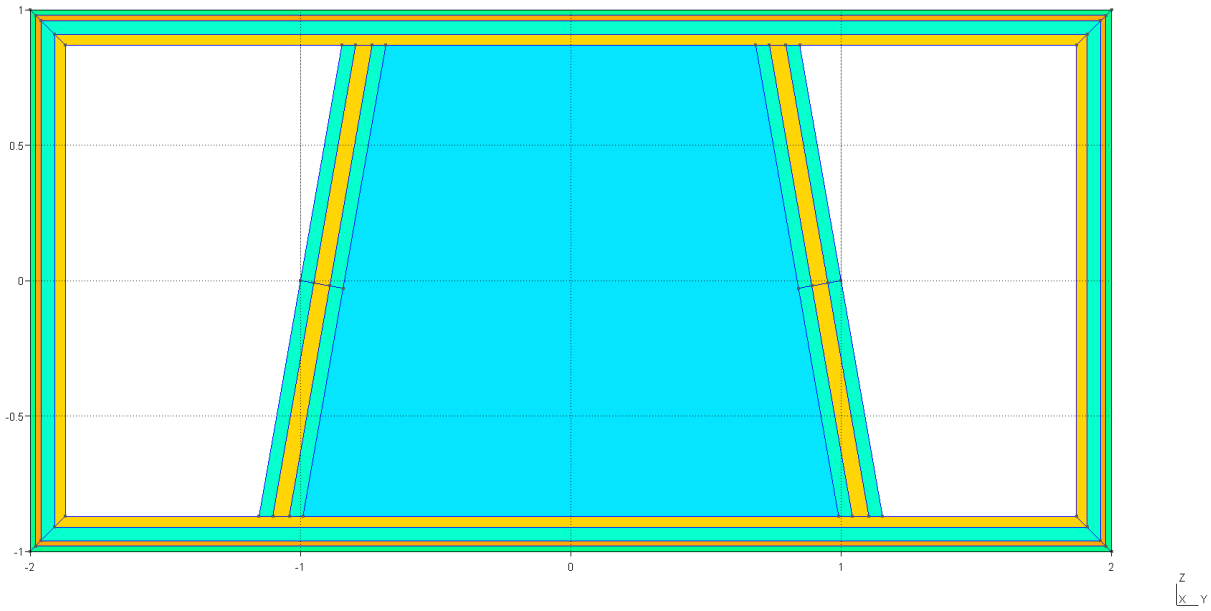

Figure 2.7: The cross section created by PreVABS and plotted by Gmsh.

Table 2.5: Timoshenko stiffness matrix in the file *box.sg.K*

3.991E+10	-1.662E+05	-4.037E+01	4.204E+07	-4.358E+03	2.867E+01
-1.662E+05	6.743E+09	3.945E+04	-2.918E+08	-1.577E+07	-4.150E+03
-4.037E+01	3.945E+04	6.165E+09	8.220E+03	-2.897E+03	-8.700E+06
4.204E+07	-2.918E+08	8.220E+03	1.972E+10	2.721E+05	3.303E+02
-4.358E+03	-1.577E+07	-2.897E+03	2.721E+05	2.173E+10	-3.025E+04
2.867E+01	-4.150E+03	-8.700E+06	3.303E+02	-3.025E+04	6.728E+10

CHAPTER

THREE

GUIDE FOR PREPARING INPUT FILES

In PreVABS, a cross section is defined through two aspects: components and global configuration, as shown in Fig. 3.1. Components are built from geometry and materials. The geometry aspect comprises definitions of base points and base lines. The material aspect includes material properties, lamina thicknesses, layup stacking sequences, etc. The global configuration contains the files included, transformation, meshing options, and analysis settings.

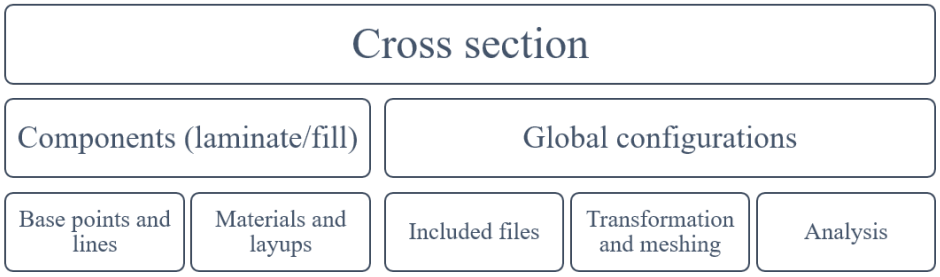

Figure 3.1: Cross section definition in PreVABS.

To create the cross section, at least four input files are needed. Definitions of the geometry, materials and layups are stored in three files separately. A top level cross section file stores the definitions of components and global configurations. In some cases, a separate file storing base points is needed (such as an airfoil). Except the base points file, which has a file extension .dat, all other files use an XML format.

A top level cross section file stores all information that will be discussed in the following sections. The input syntax for this main file is shown in Listing 3.1.

Listing 3.1: Input syntax for the cross section file.

```
<cross_section name="" format="">
  <include>...</include>
  <analysis>...</analysis>
  <general>...</general>
  <baselines>...</baselines>
  <layups>...</layups>
  <component>...</component>
  <component>...</component>
  ...
</cross_section>
```

Data of geometry and layup can be arranged in two ways, controlled by the attribute `format`:

- If omitted or set to 0, then the code will look for definitions of shapes and layups in separated files, specified in the element `<include>`.
- If set to 1, then the code will look for the definitions in the current main input file.

Specification

- **<cross_section>**
  - *name* - Name of the cross-section.
  - *format* - Format of the input file. See explaination above.
- **<include>** - File names of separately stored data.
- **<analysis>** - Configurations of cross-sectional analysis.
- **<general>** - Overall settings of the cross-section.
- **<baselines>** - Definitions of geometry and shape.
- **<layups>** - Definitions of layups.
- **<component>** - Definitions of cross-sectional components.
- **<recover>** - Inputs for recovery.

3.1 Coordinate systems

There are three coordinate frames used in PreVABS as shown in Fig. 3.2:

- **z** is a basic frame, for the normalized airfoil data points for instance;
- **x** is the final frame;
- **y** is the local frame for each element.

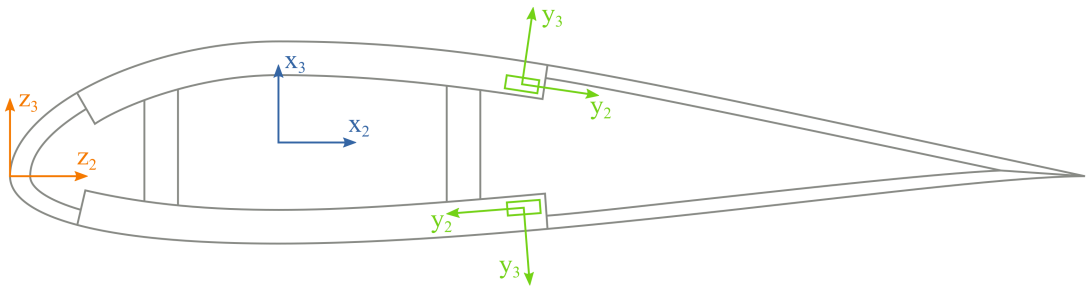

Figure 3.2: The basic, cross-sectional and elementary frames in a cross section.

Here, z_1, x_1 and y_1 are parallel to the tangent of the beam reference line and pointing out of the paper. The basic frame is where base points are defined. The cross-sectional and elementary frames have the same definitions as those in VABS. User can define the topology of a cross section in the basic frame **z** and use manipulations like translation, scaling and rotation to generate the actual geometry in **x**. For an airfoil cross section, airfoil surface data points downloaded from a database having chord length 1 are in the frame **z**, and they are transformed into the frame **x** through translation (re-define the origin), scaling (multiplied by the actual chord length), and rotation (attack angle) if necessary, as shown in Fig. 3.3. More details about this transformation can be found in Section: *Other input settings* below.

In PreVABS, the definition of the elementary frame **y** follows the rule that the positive direction of y_2 axis is always the same as the direction of the base line, and then y_3 is generated based on y_1 and y_2 according to the right-hand rule. More details about the base line can be found in *Geometry and shapes*.

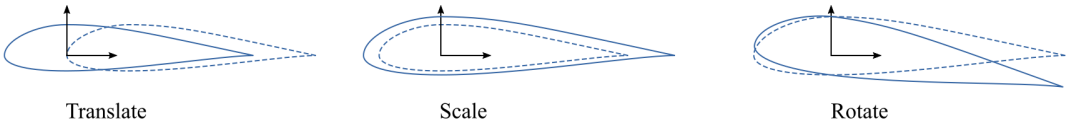

Figure 3.3: Three manipulations to transform a cross section.

3.2 Geometry and shapes

3.2.1 Base points

Base points are used to draw base lines, which form the skeleton of a cross section. They can be either actually on the base lines, or not, as reference points, for example the center of a circle. Points that are directly referred to in the definitions of base lines are called key points, such as starting and ending points of a line or an arc, or the center of a circle. The rest are normal points. The coordinates provided in the input file are defined in the basic frame **z** and then transformed into the cross-sectional frame **x**, through processes like translating, scaling and rotating. If none of those operations are needed, then those data also define the position of each point in the frame **x**.

Users can define points using sub-element `<point>` one-by-one, or list all points in a sepsrate file and include it here.

Points defined in a data file

The file storing these data is a plain text file, with a file extension `.dat`. This block of data has three columns for the name and coordinate in the cross-sectional plane.

Listing 3.2: Template for a separate file of points (point_list_file_name.dat)

```
label_1  z2  z3
label_2  z2  z3
label_3  z2  z3
...
```

Specification

- Three columns are separated by spaces.
- `label` can be the combination of any letters, numbers and underscores "_".

Note: Normal points' names can be less meaningful, even identical.

Note: If a base line is defined using the range method (explained below), e.g. 'a:b', then all points from 'a' to 'b' will be used. In this case, the order of points is important. Otherwise, points can be arranged arbitrarily.

Note: This data file can be used for storing airfoil data.

Points defined in the XML file

In this method, all points are enclosed by the XML element `<basepoints>`.

Listing 3.3: Base points input syntax.

```
<basepoints>
  <include> point_list_file_name </include>
  <point name="p1"> 0 0 </point>
  <point name="p2"> 1 0 </point>
  <point name="M" on="L" by="x2"> 3 </point>
  ...
</basepoints>
```

Using the XML format, points can be defined in the following ways:

1. Specify the complete coordinates with two numbers.
2. Confine the point on a line and specify the horizontal coordinate (z_2 or x_2), as shown in Fig. 3.4.

   `<point name="M" on="L" by="x2">3</point>`

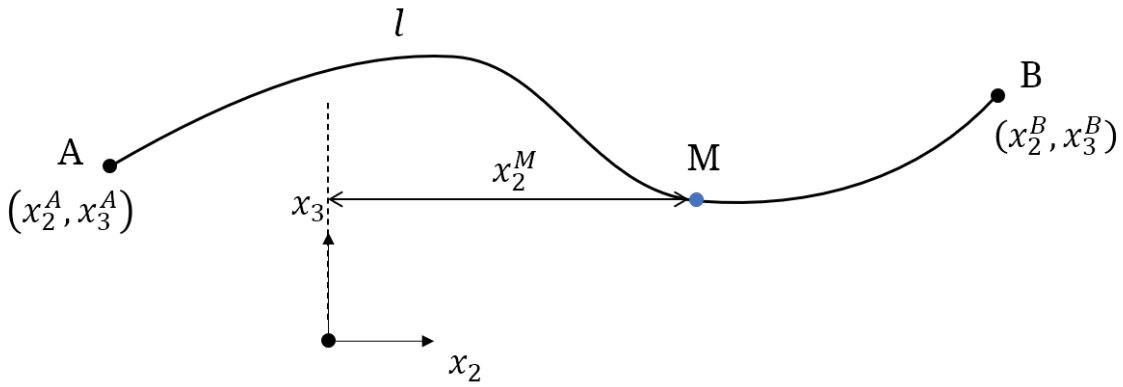

Figure 3.4: Define a new point on a line.

Specification

- **<include>** - Name of the point data file, without the file extension.
- **<point>** - Coordinates of the point. For the first method, two numbers are needed and seperated by blanks. For the second method, only one number is needed.
  - *name* - Name of the point. Required.
  - *on* - Name of the line confining the point. Optional.
  - *by* - Axis along which the coordinate is specified. Required if the point is defined on a line. Currently `x2` is the only option.

3.2.2 Base lines

Base lines form the skeleton of a cross section. PreVABS can handle three types of base lines, straight, arc and circle, as shown in Fig. 3.5. Some types have several ways to define the base line. In PreVABS, all curved base lines are in the end converted into a chain of short straight lines. User can provide those short straight lines directly for spline, arc and circle. Or, for arc or circle, user can use simple rules to draw the shape first and then PreVABS will discretize it.

Data for base points and lines are stored in an XML formatted file. The general arrangement of data is shown in Listing 3.4. The root element is `<baselines>`. Under the root element, there is a sub-element `<basepoints>` storing all definitions of points.

Each `<baseline>` element is a definition of a base line. Each one has a unique `name` and a `type`, which can be `straight`, `arc` or `circle`. Inside the `baseline` element, the `straight` and `arc` types have several different ways of definition, and thus the arrangements of data are different, which will be explained in details below.

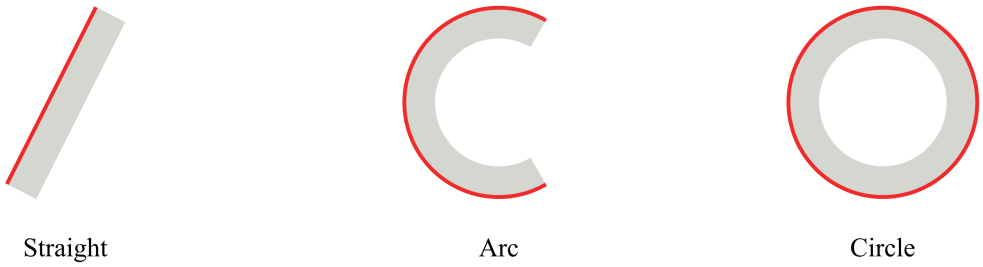

Figure 3.5: Three types of *Base lines* in PreVABS.

Listing 3.4: *Base line* input syntax.

```
<baselines>
  <basepoints>
    ...
  </basepoints>
  <baseline name="name1" type="straight">...</baseline>
  <baseline name="name2" type="arc">...</baseline>
  <baseline name="name3" type="circle">...</baseline>
  ...
</baselines>
```

Specification

- **<baseline>** - Definition of a base line (explained below).
  - *name* - Name of the base line.
  - *type* - Type of the base line. Choose one from 'straight', 'arc' and 'circle'.

Straight

For this type, the basic idea is to provide key points for a chain of straight lines The direction of a base line is defined by the order of the point list. There are three ways defining a base line of this type, as shown in Fig. 3.6.

- Use an explicit list of two or more points delimited by commas to define a polyline (i, ii).
- Use two points delimited by a colon to represent a range of points (iii). The first two methods can be used in combination.
- Use a point and a incline angle to define an straight line (iv). In this case, PreVABS will calculate the second key point (a') and generate the base line. The PreVABS-computed second key point will always be "not lower" than the user-provided key point, which means the base line will always be pointing to the upper left or upper right, or to the right if it is horizontal.

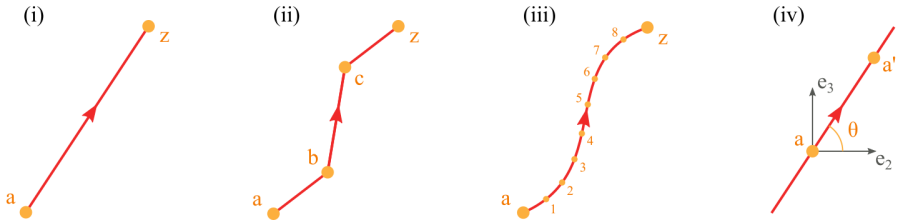

Figure 3.6: Different ways of defining a straight type base line in PreVABS.

Note: Use `type="straight"` for splines.

Listing 3.5: Input syntax for the base lines shown in Fig. 3.6

```
<baselines>
  ...
  <baseline name="i" type="straight">
    <points> a,z </points>
  </baseline>

  <baseline name="ii" type="straight">
    <points> a,b,c,z </points>
  </baseline>

  <baseline name="iii" type="straight">
    <points> a:z </points>
  </baseline>

  <baseline name="iv" type="straight">
    <point> a </point>
    <angle> theta </angle>
  </baseline>
  ...
</baselines>
```

Specification

- **<points>** - Names of points defining the base line, delimited by commas (explicit list), or colons (range). Blanks are not allowed.
- **<point>** - Name of a point.

- **<angle>** - Incline angle of the line. The positive angle (degree) is defined from the positive z_2 axis, counter-clockwise.

Arc

A real arc can also be created using a group of base points, in which case the straight type should be used. The arc type provides a parametric way to build this type of base line, then PreVABS will discretize it. To uniquely define an arc, user needs to provide at least four of the following six items: center, starting point, ending point, radius, angle and direction, as shown in Fig. 3.7.

There are two ways of defining an arc as shown in Fig. 3.8.

- Use center, starting point, ending point and direction.
- Use center, starting point, angle and direction.

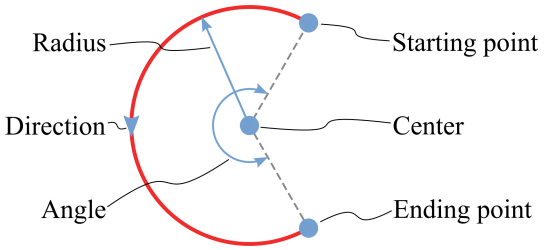

Figure 3.7: Items in an arc.

Figure 3.8: Two possible cases of defining an arc.

Listing 3.6: Input syntax for the base lines shown in Fig. 3.8

```
<baselines>
  ...
  <baseline name="left" type="arc">
    <center> c </center>
    <start> s </start>
    <end> e </end>
    <direction> ccw </direction>
    <discrete by="angle> 9 </discrete>
  </baseline>

  <baseline name="right" type="arc">
    <center> c </center>
```

(continues on next page)

(continued from previous page)

```
    <start> s </start>
    <angle> a </angle>
    <!-- here the direction is the default value 'ccw' -->
    <discrete by="number"> 10 </discrete>
  </baseline>
  ...
</basepoints>
```

Specification

- ** ** - Name of the center point.
- **<start>** - Name of the starting point.
- **<end>** - Name of the ending point.
- **<direction>** - Direction of the circular arc. Choose from 'cw' (clockwise) and 'ccw' (counter-clockwise). Default is 'ccw'.
- **<angle>** - Central angle of the arc.
- **<discrete>** - Number of discretization. If 'by="angle"', then new points are created every specified degrees of angle. If 'by="number"', then specified number of new points are created and evenly distributed on the arc.
  - *by* - Choose one from 'angle' and 'number'. Default is 'angle'.

Circle

Defining a circle is simpler than an arc. User only need to provide a center with radius or another point on the circle. The corresponding element tags are `<center>`, `<radius>` and `<point>`. A sample input file demonstrating the two methods is presented in Listing 3.7.

There are two ways of defining a circle.

- Use center and radius.
- Use center and a point on the circle.

Listing 3.7: Input syntax for the 'circle' type base line.

```
<baselines>
  ...
  <baseline name="circle1" type="circle">
    <center> c </center>
    <radius> r </radius>
  </baseline>

  <baseline name="circle2" type="circle">
    <center> c </center>
    <point> p </point>
    <direction> cw </direction>
  </baseline>
  ...
</baselines>
```

Specification

- ** ** - Name of the center point.
- **<radius>** - Radius of the circle.

- **<point>** - Name of a point on the circle.
- **<direction>** - Direction of the circle. Choose from 'cw' (clockwise) and 'ccw' (counter-clockwise). Default is 'ccw'.
- **<discrete>** - Number of discretization. If 'by="angle"', then new points are created every specified degrees of angle. If 'by="number"', then specified number of new points are created and evenly distributed on the circle.
  - *by* - Choose one from 'angle' and 'number'. Default is 'angle'.

3.3 Materials and layups

PreVABS uses the keyword `material` for the physical properties attached to any materials, while `lamina` for material plus thickness, which in a sense is fixed by manufacturers. This can be thought as the basic commercially available "material", such as a composite preprag. A layer is a stack of laminae with the same fiber orientation. The thickness of a layer can only be a multiplier of the lamina thickness. Layup is several layers stacked together in a specific order. This relationship is illustrated in Fig. 3.9. More details can be found below.

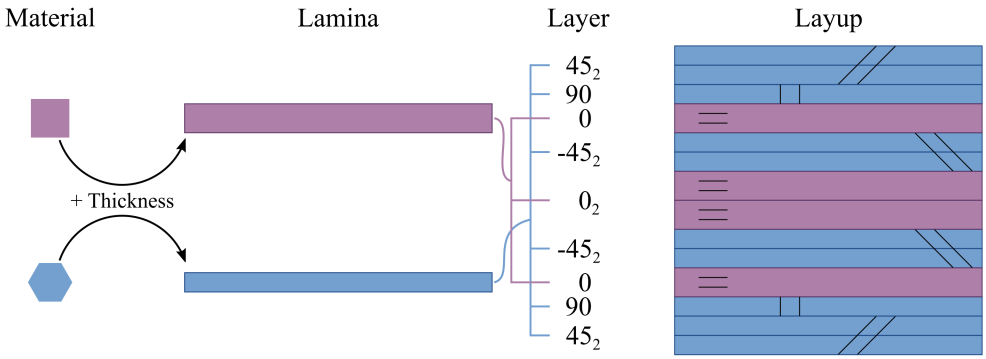

Figure 3.9: Relationship between material, lamina, layer and layup.

3.3.1 Material and lamina

Both materials and laminae are stored in one XML file, under the single root element `<materials>`. A template of this file is shown in Listing 3.8. Each material must have a name and type. Under each `<material>` element, there are a `<density>` element and an `<elastic>` element.

Listing 3.8: Input syntax for materials.

```
<materials>
  ...
  <material name="iso1" type="isotropic">
    <density>...</density>
    <elastic>
      <e>...</e>
      <nu>...</nu>
    </elastic>
  </material>

  <material name="lam1" type="lamina">
    <density>...</density>
```

(continues on next page)

(continued from previous page)

```
    <elastic>
      <e1>...</e1>
      <e2>...</e2>
      <nu12>...</nu12>
      <g12>...</g12>
    </elastic>
  </material>

  <material name="orth1" type="orthotropic">
    <density>...</density>
    <elastic>
      <e1>...</e1>
      <e2>...</e2>
      <e3>...</e3>
      <g12>...</g12>
      <g13>...</g13>
      <g23>...</g23>
      <nu12>...</nu12>
      <nu13>...</nu13>
      <nu23>...</nu23>
    </elastic>
  </material>

  <material name="aniso1" type="anisotropic">
    <density>...</density>
    <elastic>
      <c11>...</c11>
      <c12>...</c12>
      <c13>...</c13>
      <c14>...</c14>
      <c15>...</c15>
      <c16>...</c16>
      <c22>...</c22>
      <c23>...</c23>
      <c24>...</c24>
      <c25>...</c25>
      <c26>...</c26>
      <c33>...</c33>
      <c34>...</c34>
      <c35>...</c35>
      <c36>...</c36>
      <c44>...</c44>
      <c45>...</c45>
      <c46>...</c46>
      <c55>...</c55>
      <c56>...</c56>
      <c66>...</c66>
    </elastic>
  </material>
  ...
</materials>
```

For the `lamina` type material, the code will internally convert it to the `orthotropic` type, by assigning the rest five constants in the following way:

- e3 = e2
- nu13 = nu12

- nu23 = 0.3
- g13 = g12
- g23 = e2 / (2 * (1 + nu23))

Specification

- **<material>** - Root element for each material.
  - *name* - Name of the material.
  - *type* - Type of the material. Choose one from 'isotropic', 'orthotropic', 'anisotropic' and 'lamina'.
  - **<density>** - Density of the material. Defualt is 1.0.
  - **<elastic>** - Elastic properties of the material. Specifications are different for different types.
    - If *type="isotropic"* - 2 constants: 'e' and 'nu'.
    - If *type="lamina"* - 4 constants: 'e1', 'e2', 'nu12' and 'g12'.
    - If *type="orthotropic"* - 9 constants: 'e1', 'e2', 'e3', 'g12', 'g13', 'g23', 'nu12', 'nu13' and 'nu23'.
    - If *type="anisotropic"* - 21 constants: 'c11', 'c12', 'c13', 'c14', 'c15', 'c16', 'c22', 'c23', 'c24', 'c25', 'c26', 'c33', 'c34', 'c35', 'c36', 'c44', 'c45', 'c46', 'c55', 'c56' and 'c66'. These constants are defined in Equation (3.1).
    - If *type="lamina"* - 4 constants: 'e1', 'e2', 'g12' and 'nu12'. Internally, this type of material will be converted to the 'orthotropic' material. The default values for the rest components are: 'e3=e2', 'nu13=nu12', 'nu23=0.3', 'g13=g12' and 'g23=e3/(2*(1+nu23))'. These default values can be overwritten by custom values.

$$\begin{Bmatrix} \sigma_{11} \\ \sigma_{12} \\ \sigma_{13} \\ \sigma_{22} \\ \sigma_{23} \\ \sigma_{33} \end{Bmatrix} = \begin{bmatrix} c_{11} & c_{12} & c_{13} & c_{14} & c_{15} & c_{16} \\ c_{12} & c_{22} & c_{23} & c_{24} & c_{25} & c_{26} \\ c_{13} & c_{23} & c_{33} & c_{34} & c_{35} & c_{36} \\ c_{14} & c_{24} & c_{34} & c_{44} & c_{45} & c_{46} \\ c_{15} & c_{25} & c_{35} & c_{45} & c_{55} & c_{56} \\ c_{16} & c_{26} & c_{36} & c_{46} & c_{56} & c_{66} \end{bmatrix} \begin{Bmatrix} \epsilon_{11} \\ 2\epsilon_{12} \\ 2\epsilon_{13} \\ \epsilon_{22} \\ 2\epsilon_{23} \\ \epsilon_{33} \end{Bmatrix} \tag{3.1}$$

Each lamina element has a name, and contains a material name and a value for thickness. The input syntax is shown in Listing 3.9.

Listing 3.9: Input syntax for lamina.

```
<materials>
  ...
  <lamina name="lamina1">
    <material> orth1 </material>
    <thickness>...</thickness>
  </lamina>
  ...
</materials>
```

Specification

- **<lamina>** - Root element for the definition of each lamina.
  - *name* - Name of the lamina.
- **<material>** - Name of the material of the lamina.
- **<thickness>** - Thickness of the lamina.

3.3.2 Layups

In general, there are two ways to define a layup, explicit list and stacking sequence code. For the explicit list, a laminate is laid onto the base line from the first layer in the list to the last one, in the direction given by the user. For the stacking sequence, the layup starts from left to the right. User should pay attention to the relations among the base line direction, elemental frame **y** and fiber orientation θ_3 of each layer, as shown in Fig. 3.10. Change of direction of the base line will change the elemental frame **y** as defined, which will further require the user to change the fiber orientations accordingly, even though nothing changes physically. All layup information are included in one XML file. A template of this file can be found in Listing 3.10.

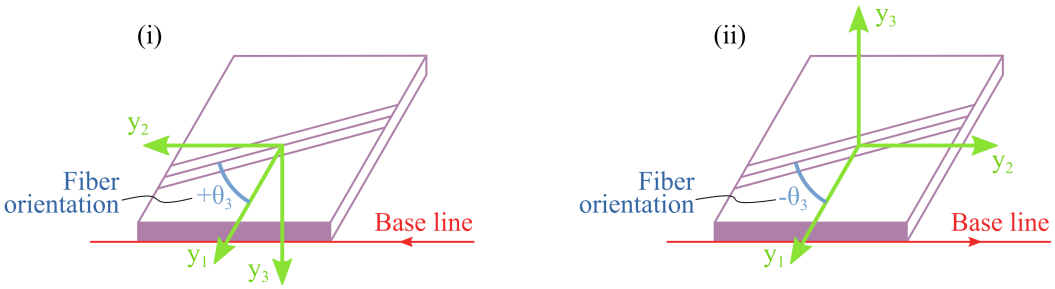

Figure 3.10: Relations among the base line direction, elemental frame **y** and fiber orientation (Note y_2 is parallel to the base line direction).

Listing 3.10: Input syntax for layups.

```
<layups>
  <layup name="layup1" method="explicit list">
    <layer lamina="lamina1">...</layer>
    ...
  </layup>
  <layup name="layup2" method="stack sequence">...</layup>
  <layup name="layup3" method="ply percentage">...</layup>
  ...
</layups>
```

Specification

- **<layup>** - Root element for the definition of each layup.
  - *name* - Name of the layup.
  - *method* - Method of defining the layup. Choose one from 'layer list' (or 'll') and 'stack sequence' (or 'ss'). Default is 'layer list'.

Explicit list

This method requires user to write down the lamina name, fiber orientation and number of successive laminas with the same fiber orientation, layer by layer. Alternatively, a layer can also be defined by the name of a sublayup. A sublayup must appear before the layup where it is referenced. A template for one layer is shown below.

Listing 3.11: A template for the layup layer list input.

```
<layups>
  ...
  <layup name="...">
```

(continues on next page)

(continued from previous page)

```
    <layer lamina="lamina_name"> angle:stack </layer>
    <layer lamina="lamina_name"> angle:stack </layer>
    <layer lamina="lamina_name"> angle:stack </layer>
    <layer layup="sublayup_name"/>
    ...
  </layup>
  ...
</layups>
```

Specification

- **<layer>** - Material orientation and number of plies separated by a colon 'angle:stack'. Default values are 0 for 'angle' and 1 for 'stack'. If there is only one number presented in this element, then it is read in as 'angle', not 'stack', which is 1 by default.
  - *lamina* - Optional. Name of the lamina used in the current layer.
  - *layup* - Optional. Name of the sublayup used in the current layer. One and only one of *lamina* and *layup* should be used. 'angle' and 'stack' are not needed when *layup* is used.

An example for the layup shown in Fig. 3.9 is given in Listing 3.13 and Listing 3.14.

Stacking sequence code

This method requires users to provide one lamina name and the stacking sequence code. A template is shown below.

Listing 3.12: A template for the layup stacking sequence input.

```
<layups>
  ...
  <layup name="..." method="stack sequence">
    <lamina>...</lamina>
    <code>...</code>
  </layup>
  ...
</layups>
```

Specification

- **<lamina>** - Name of the lamina used.
- **<code>** - Stacking sequency code. Explained below.

Rules of writing the stacking sequence code

- All fiber orientations should be put between a pair of square brackets `[]`;
- Different fiber orientations are separated by slash `/`;
- After the right bracket, user can add `ns` to indicate symmetry of the layup, where `n` is the number of the symmetry operations needed to generate the complete layup;
- Successive laminae with the same fiber orientation can be expressed using colon like `angle:stack`, where `angle` is the fiber orientation and `stack` is the number of plies;
- If a group of fiber orientations is repeated, user needs to close them in a pair of round brackets `()`.

Examples

Table 3.1: Examples of stacking sequence code

Code	Complete sequence
`[0/90]2s`	0, 90, 90, 0, 0, 90, 90, 0
`[(45/-45):2/0:2]s`	45, -45, 45, -45, 0, 0, 0, 0, -45, 45, -45, 45

Listing 3.13: Example material and lamina input file for the layup shown in Fig. 3.9.

```
<materials>
  <material name="square" type="orthotropic">
    <density>...</density>
    <elastic>...</elastic>
  </material>

  <material name="hexagon" type="orthotropic">
    <density>...</density>
    <elastic>...</elastic>
  </material>

  <lamina name="la_square_15">
    <material> square </material>
    <thickness> 1.5 </thickness>
  </lamina>

  <lamina name="la_hexagon_10">
    <material> hexagon </material>
    <thickness> 1.0 </thickness>
  </lamina>
</materials>
```

Listing 3.14: Example layup input file for the layup shown in Fig. 3.9.

```
<layups>
  <layup name="layup_el" method="explicit list">
    <layer lamina="la_hexagon_10"> 45:2 </layer>
    <layer lamina="la_hexagon_10"> 90 </layer>
    <layer lamina="la_square_15"></layer>
    <layer lamina="la_hexagon_10"> -45:2 </layer>
    <layer lamina="la_square_15"> 0:2 </layer>
    <layer lamina="la_hexagon_10"> -45:2 </layer>
    <layer lamina="la_square_15"></layer>
    <layer lamina="la_hexagon_10"> 90 </layer>
    <layer lamina="la_hexagon_10"> 45:2 </layer>
  </layup>

  <layup name="layup_ss" method="stack sequence">
    <lamina> la_square_15 </lamina>
    <code> [(45/-45):2/0:4/90]2s </code>
  </layup>
</layups>
```

3.4 Components

Listing 3.15: Input syntax for components.

```
<component name="" type="" depend="">...</component>
```

Specification

- **<component>** - Root element for the definition of each component.
  - *name* - Name of the component.
  - *type* - Type of the component. Choose one from 'laminate' and 'fill'. Default is 'laminate'.
  - *depend* - List of name of dependent components, delimited by commas.

3.4.1 Laminate-type component

For a cross section in PreVABS, laminates are created as segments. A segment is a unique combination of a base line and a layup. Segments are connected through different ways as shown in Fig. 3.11. According to this, segments can be grouped into components. The rule of thumb is: if two segments are connected in the first two ways ('V1' and 'V2'), then they belong to one component; if they are connected as the third way ('T'), then they should be put into different components, and component 2 will be created after the finish of component 1.

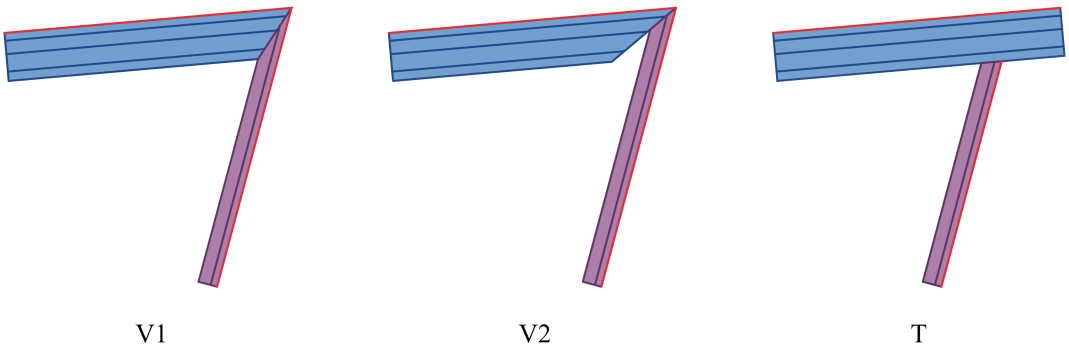

Figure 3.11: Three types of connections that can be created in PreVABS.

A schematic plot of a segment is shown in Fig. 3.12. The base line provides a reference for the position and direction of the layup. Layers can be laid to the left or right of the base line. The direction is defined as one's left or right, assuming one is walking along the direction of the base line.

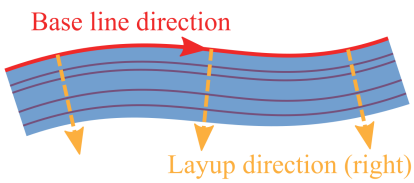

Figure 3.12: A typical segment in PreVABS and the relation between base line direction and layup direction.

All segments definitions are stored in the main input file. The complete format will be discussed later since the overall configurations are also included in this file, which will be explained in Section: *Other input settings*. The input syntax for laminate-type components are given in Listing 3.16. Each component can have multiple segments.

There are two ways to define segments.

DEFINITION 1: Define segments individually

Use a base line and a single layup to create the segment. In this way, the layup covers the entire of the base line. Each `<segment>` element has one attribute, `name`, and two child elements, `<baseline>` and `<layup>`. The `<layup>` element has another attribute `direction` (see Fig. 3.12).

Joint of two connecting segments can be changed from 'V2' (default) to 'V1' by using a `<joint>` element. It requires the names of two segments, delimited by a comma (',') and an attribute `style` specifying the joint type.

DEFINITION 2: Define segments collectively

Use a base line and multiple layups to create multiple segments. Each layup can be assigned to a portion of the base line, using a beginning and an ending locations. These locations are normalized parametric positions on the base line. The beginning location must be smaller than the ending one. If the line is open, the location can only be a number between 0 and 1. If the line is closed, the location can be any number, even negative, as long as the length is not greater than 1. Then PreVABS will split the base line, combine layups and create segments automatically.

An example is provided below (Fig 3.13, Fig 3.14, Listing 3.17).

Listing 3.16: Input syntax for the laminate-type components.

```
<component name="cmp_surface">
  <segment name="sgm_1">
    <baseline> baseline1 </baseline>
    <layup direction="right"> layup1 </layup>
  </segment>
  <segment name="sgm_2">
    <baseline>...</baseline>
    <layup>...</layup>
  </segment>
  ...
  <joint style="2"> sgm_1,sgm_2 </joint>
  ...
</component>


<component name="cmp_web">
  <segment>
    <baseline> baseline2 </baseline>
    <layup direction="left"> layup2 </layup>
  </segment>
  ...
</component>


<component name="...">
  <segments>
    <baseline> base_line_3_name </baseline>
    <layup_side> left </layup_side>
    <layup> layup_1_name </layup>
    <layup begin="0.1"> layup_2_name </layup>
    <layup end="0.7"> layup_3_name </layup>
    <layup begin="0.2" end="0.9"> layup_4_name </layup>
    ...
```

(continues on next page)

(continued from previous page)

```
  </segments>
</component>
```

Example

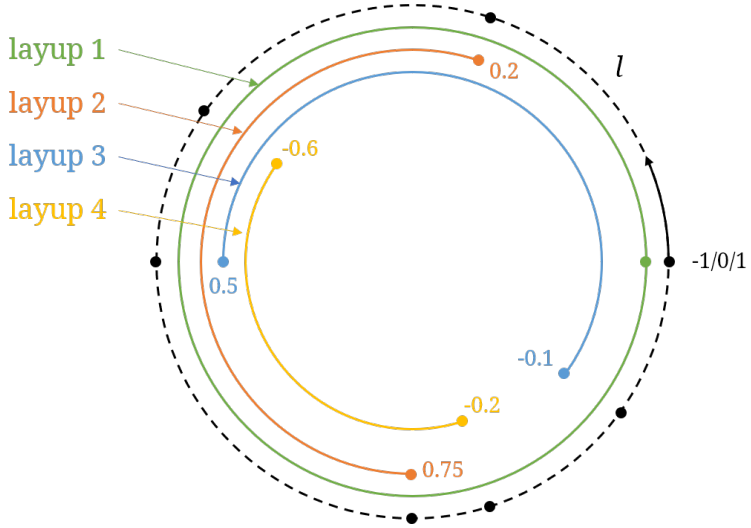

Figure 3.13: Segment layup range definition.

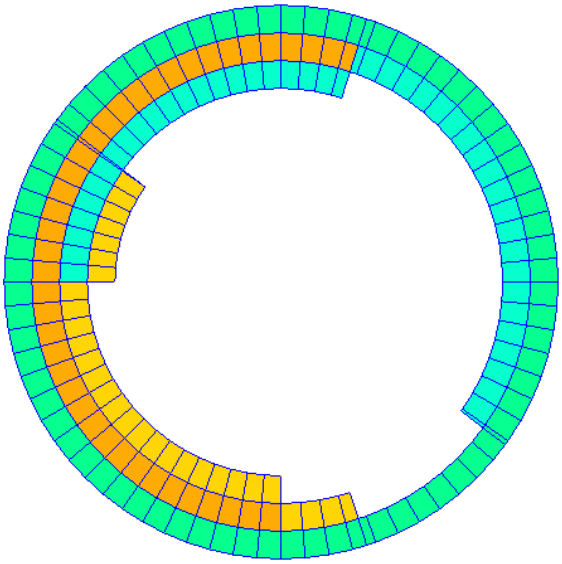

Figure 3.14: Segments plot.

Listing 3.17: Example input.

```
<component name="...">
  <segments>
    <baseline> l </baseline>
```

(continues on next page)

(continued from previous page)

```
    <layup> layup1 </layup>
    <layup begin="0.2" end="0.75"> layup2 </layup>
    <layup begin="-0.1" end="0.5"> layup3 </layup>
    <layup begin="-0.6" end="-0.2"> layup4 </layup>
  </segments>
</component>
```

Specification

DEFINITION 1

- **** - Root element of the definition of the segment.
  - *name* - Name of the segment.
- **<baseline>** - Name of the base line defining this segment.
- **<layup>** - Name of the layup defining this segment.
  - *direction* - Direction of layup. Choose one from 'left' and 'right'. Default is 'left'.
- **<joint>** - Names of two segments delimited by a comma (',') that will be joined.
  - *style* - Style of the joint. Choose one from '1' and '2'. Default is '1'.

DEFINITION 2

- **<segments>** - Root element of the definition.
- **<baseline>** - Name of the base line definint these segments.
- **<layup_side>** - Direction of the following layups. Choose one from 'left' and 'right'. Default is 'left'.
- **<layup>** - Name of the layup.
  - *begin* - Normalized parametric beginning location of the layup on the base line. Defualt is '0.0'.
  - *end* - Normalized parametric ending location of the layup on the base line. Defualt is '1.0'.

3.4.2 Fill-type component

Besides creating laminates, user can use one material to fill a region. A typical usage is a nose mass in an airfoil type cross section. A schematic plot is shown in Fig. 3.15.

The key to this type of component is the indication of the fill region. There are two things to pay attention to. First, make sure that the boundary of the region is well-defined. The region can be surrounded by a number of components. User can also create some extra base lines as new boundaries. Second, locate the region where the material will be filled into. Then can be done by using a point inside the region, or if there are extra base lines, user can indicate the fill side with respect to one of the lines, similar to defining layup side in a segment.

Fill-type components are also defined in the main input file. A template is shown in Listing 3.18. A fill-type component is indicated by the attribute `type="fill"`. A `<material>` child element is required. A `<baseline>` element is optional and is used to create extra boundaries. This sub-element has one attribute `fillside`, which can be either left or right. A `<location>` element is used to store the name of a point that is inside the desired fill region, and is also optional.

Listing 3.18: Input syntax for the fill-type components.

```
<component name="cmp_fill_1" type="fill">
  <location> point_fill </location>
  <material> material1 </material>
```

(continues on next page)

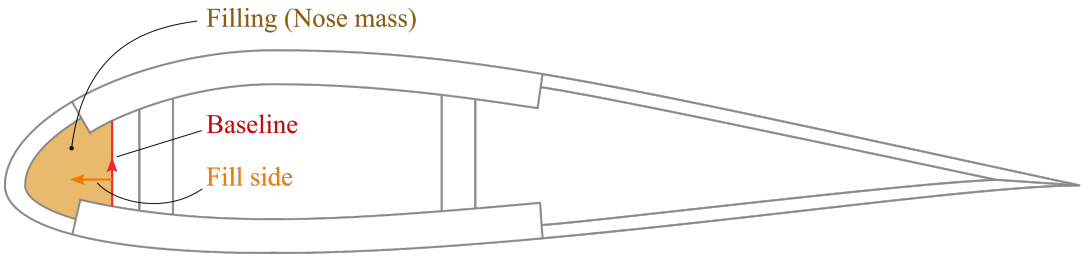

Figure 3.15: Example usage of a fill-type component as a nose mass in an airfoil cross section.

(continued from previous page)

```
</component>


<component name="cmp_fill_2" type="fill">
  <baseline> bsl </baseline>
  <location> point_fill </location>
  <material> material1 </material>
  <mesh_size at="p1,p2"> 0.1 </mesh_size>
</component>


<component name="cmp_fill_3" type="fill">
  <baseline fillside="right"> bsl_3 </baseline>
  <material> material1 </material>
</component>
```

Local mesh size

Besides the mesh size set globally in the main input file, filling type components can be assigned local mesh sizes. This is usually for the purpose of reducing the total number of elements and computational cost. This feature is based on the embedded objects (points and lines) provided by Gmsh. Hence, one or more points need to be specified as the 'seed' of local mesh sizes.

```
<mesh_size at="p1,p2">0.1</mesh_size>
```

The mesh size will vary gradually from the local to the global setting. Hence, if only one point is used, then the local mesh size will only affect a circular region. If multiple points are used, then lines will be created by connecting points seqentially and assigned the local mesh size as well.

An example is provided below (Fig 3.16, Fig 3.17, Listing 3.19).

Listing 3.19: Example input.

```
<component name="filling 1" type="fill" depend="...">
  <location> A </location>
  <material> material1 </material>
  <mesh_size at="A"> 0.2 </mesh_size>
</component>

<component name="filling 2" type="fill" depend="...">
  <location> B </location>
  <material> material2 </material>
```

(continues on next page)

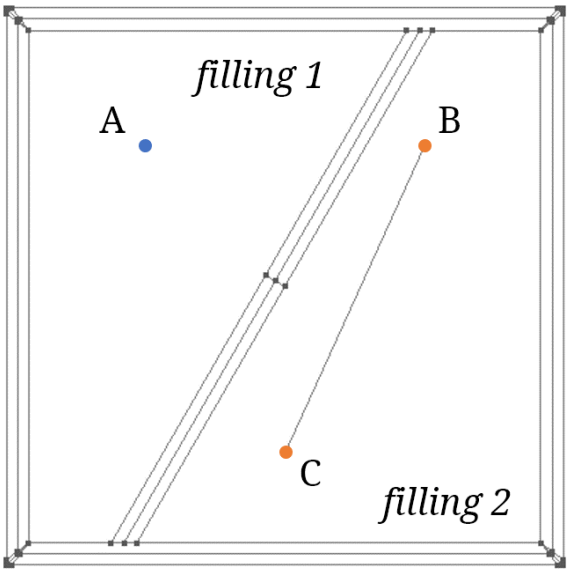

Figure 3.16: Local mesh definition.

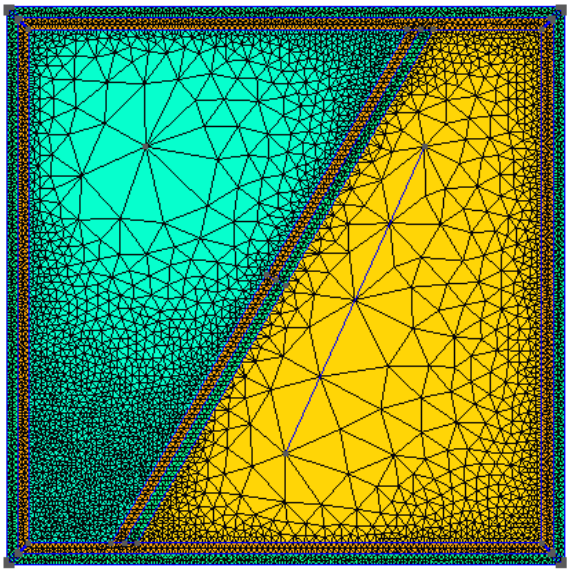

Figure 3.17: Local mesh plot.

(continued from previous page)

```
  <mesh_size at="B,C"> 0.2 </mesh_size>
</component>
```

Specification

- **<material>** - Name of the material to be filled. Required.
- **<location>** - Name of the point located in the fill region. Optional.
- **<baseline>** - Name of the base line defining part or complete boundary. Optional.
  - *fillside* - Side of the fill with respect to the base line. Optional.
- **<theta1>** - Rotating angle in degree about the x_1 axis. Optional. Default is 0 degree.
- **<theta3>** - Rotating angle in degree about the y_3 axis. Optional. Default is 0 degree.
- **<mesh_size>** - Local mesh size. Optional.
  - *at* - A list of names of points where the local mesh size will be assgined. Required.

3.4.3 Components dependency

If two components are connected in the 'T' type (Fig. 3.11), then the order of creation of the two components must be specified. This is done by specifying the dependency. For the case shown in the figure, the creation of the thin vertical component (purple) is dependent on the creation of the thick horizontal component (blue). Hence, in the definition of the purple component, a `depend` attribute should be specified as shown below.

Listing 3.20: Dependency.

```
<component name="cmp_blue" type="laminate">...</component>
<component name="cmp_purple" type="laminate" depend="cmp_blue">...</component>
```

If a component is dependent on multiple components, their names shoule all be listed, delimited by comma.

3.5 Specifications for recovery

3.5.1 VABS

Once the 1D beam analysis is finished, those results can be used to recover local 3D strains and stresses at every point of the cross section. As stated in the VABS manual, the following data are required to carry out the recover analysis at a selected location along the beam:

- 1D beam displacements;
- rotations in the form of a direction cosine matrix;
- sectional forces and moments;
- distributed forces and moments, and their first three derivatives with respect to x_1.

Note: The current version of PreVABS only performs recovery for Euler-Bernoulli model and Timoshenko model. If you want to perform recovery for other models such as the Vlasov model, please refer to the VABS manual to modify the input file yourself.

These data are included in a <global> element, added into the <cross_section> element in the main input file. The nine entries in the direction cosine matrix are flattened into a single array in the <rotations> element. This matrix is defined in Eq. (3.2). Note that displacements and rotations are needed only for recovering 3D displacements. A template for this part of data is shown in Listing 3.21. Numbers in this template are defualt values. Once this file is updated correctly, the VABS recover analysis can be carried out as shown in Section: *Command line options*.

$$\mathbf{B}_i = C_{i1}\mathbf{b}_1 + C_{i2}\mathbf{b}_2 + C_{i3}\mathbf{b}_3 \quad \text{with} \quad i = 1, 2, 3 \tag{3.2}$$

where $\mathbf{B}_1$, $\mathbf{B}_2$, and $\mathbf{B}_3$ are the base vectors of the deformed triad and $\mathbf{b}_1$, $\mathbf{b}_2$, and $\mathbf{b}_3$ are the base vectors of the undeformed triad.

Listing 3.21: A template for the recover data in a *Cross section* file.

```
<cross_section name="cs1">
  ...
  <dehomo> 1 </dehomo>
  <global>
    <displacements> 0 0 0 </displacements>
    <rotations> 1 0 0 0 1 0 0 0 1 </rotations>
    <loads> 0 0 0 0 0 0 </loads>
    <distributed>
      <forces> 0 0 0 </forces>
      <forces_d1> 0 0 0 </forces_d1>
      <forces_d2> 0 0 0 </forces_d2>
      <forces_d3> 0 0 0 </forces_d3>
      <moments> 0 0 0 </moments>
      <moments_d1> 0 0 0 </moments_d1>
      <moments_d2> 0 0 0 </moments_d2>
      <moments_d3> 0 0 0 </moments_d3>
    </distributed>
  </global>
</cross_section>
```

Specification

- **<global>** - Global analysis results. Required.
- **<displacements>** - Three components (u_1, u_2, u_3) of the global displacements. Optional. Default values are $\{0, 0, 0\}$.
- **<rotations>** - Nine components (C_{11}, C_{12}, C_{13}, C_{21}, C_{22}, C_{23}, C_{31}, C_{32}, C_{33}) of the global rotations (direction cosine matrix). Optional. Default values are $\{1, 0, 0, 0, 1, 0, 0, 0, 1\}$.
- **<loads>** - The sectional loading components. For the Euler-Bernoulli model, four numbers (F_1, M_1, M_2, M_3) are needed. For the Timoshenko model, six numbers (F_1, F_2, F_3, M_1, M_2, M_3) are needed. Optional. Default values are zero loads.
- **<distributed>** - Distributed sectional loads per unit span (Timoshenko model only). Optional. Default values are zero distributed loads.
  - **<forces>** - Distributed sectional forces (f_1, f_2, f_3).
  - **<forces_d1>** - First derivative of distributed sectional forces (f_1', f_2', f_3').
  - **<forces_d2>** - Second derivative of distributed sectional forces (f_1'', f_2'', f_3'').
  - **<forces_d3>** - Third derivative of distributed sectional forces (f_1''', f_2''', f_3''').
  - **<moments>** - Distributed sectional moments (m_1, m_2, m_3).
  - **<moments_d1>** - First derivative of distributed sectional moments (m_1', m_2', m_3').

  - **<moments_d2>** - Second derivative of distributed sectional moments (m_1'', m_2'', m_3'').
  - **<moments_d3>** - Third derivative of distributed sectional moments (m_1''', m_2''', m_3''').
- **<dehomo>** - Order of theory for dehomogenization, 1 for nonlinear beam theory and 2 for linear beam theory. Optional. Default is 2.

3.5.2 SwiftComp

Listing 3.22: Syntax for the dehomogenization inputs for SwiftComp.

```
<global measure="stress">
  <displacements> 0 0 0 </displacements>
  <rotations> 1 0 0 0 1 0 0 0 1 </rotations>
  <loads> 0 0 0 0 0 0 </loads>
</global>
```

Specification

- **<global>** - Global analysis results. Required.
  - *measure* - Type of the sectional loads, `stress` for generalized stresses and `strain` for generalized strains. Required.
- **<displacements>** - Three components (u_1, u_2, u_3) of the global displacements. Optional. Default values are $\{0, 0, 0\}$.
- **<rotations>** - Nine components (C_{11}, C_{12}, C_{13}, C_{21}, C_{22}, C_{23}, C_{31}, C_{32}, C_{33}) of the global rotations (direction cosine matrix). Optional. Default values are $\{1, 0, 0, 0, 1, 0, 0, 0, 1\}$.
- **<loads>** - The sectional loading components. For the Euler-Bernoulli model, four numbers (F_1, M_1, M_2, M_3) are needed. For the Timoshenko model, six numbers (F_1, F_2, F_3, M_1, M_2, M_3) are needed. Optional. Default values are zero loads.

3.6 Other input settings

There are three groups of these settings.

3.6.1 Included files

This part contains file names for base lines, materials (local) and layups as shown in Listing 3.23. The `<material>` sub-element is optional. If this is included, PreVABS will read extra materials from this local file, besides the global material database (MaterialDB.xml). If there are materials and laminae with the same names, data in the local material file will overwrite those in the global database.

- `path` in the `<include>` element is the relative path to the main input file;
- File extensions `.xml` can be omitted.

Listing 3.23: Input syntax for the included files.

```
<include>
  <baseline> path/baseline_file_name </baseline>
  <material> path/material_file_name </material>
  <layup> path/layup_file_name </layup>
</include>
```

Specification

- **<baseline>** - Name of the included base line file.
- **<material>** - Name of the included local material file.
- **<layup>** - Name of the included layup file.

3.6.2 Analysis options

The second part contains settings for the analysis in VABS. `<model>` can be 0 for classical beam model, or 1 for refined (Timoshenko) model.

Listing 3.24: A template for the analysis options in a main input file.

```
<analysis>
  <model> 1 </model>
</analysis>
```

Specification

- **<model>** - Beam model. Choose one from '0' (classical) and '1' (refined/Timoshenko).

3.6.3 Global shape and mesh settings

The last part contains optional global geometry and meshing settings, which are all stored in a `<general>` sub-element.

User can set the global transformations of the cross section. The three transformation operations have been discussed in Section: *Coordinate systems*.

- The order of transformation operation is: translating, scaling, and rotating;
- All operations are performed on the cross section, not the frame;
- The scaling operation will only scale the shape (base lines), and have no effect on the thicknesses of laminae;
- The rotating angle starts from the positive half of the x_2 axis, and increases positively counter-clockwise (right-hand rule).

There are two meshing options available now, global meshing size `<mesh_size>` and type of elements `<element_type>`.

- If not setting, the global meshing size will be the minimum layer thickness by default;
- Two options for the element type are linear and quadratic.

Listing 3.25: A template for the global shape and mesh settings in a main input file.

```
<general>
  <translate> e2  e3 </translate>
  <scale> scaling_factor </scale>
  <rotate> angle </rotate>
  <mesh_size> a </mesh_size>
  <element_type> quadratic </element_type>
  <tolerance> 1e-9 </tolerance>
</general>
```

Specification

- **<translate>** - Horizontal and vertical translation of the cross-section. The origin will be moved to (-e2, -e3).
- **<scale>** - Scaling factor of the cross-section.
- **<rotate>** - Rotation angle of the cross-section.
- **<mesh_size>** - Global mesh size.
- **<element_type>** - Order of elements. `linear` or `quadratic` (default).
- **<tolerance>** - Tolerance used in geometric computation. Optional. Default value is 1e-12.

Note: Only triangular element is available in the current version.

CHAPTER

FOUR

EXAMPLES

4.1 Box beam

4.1.1 Problem description

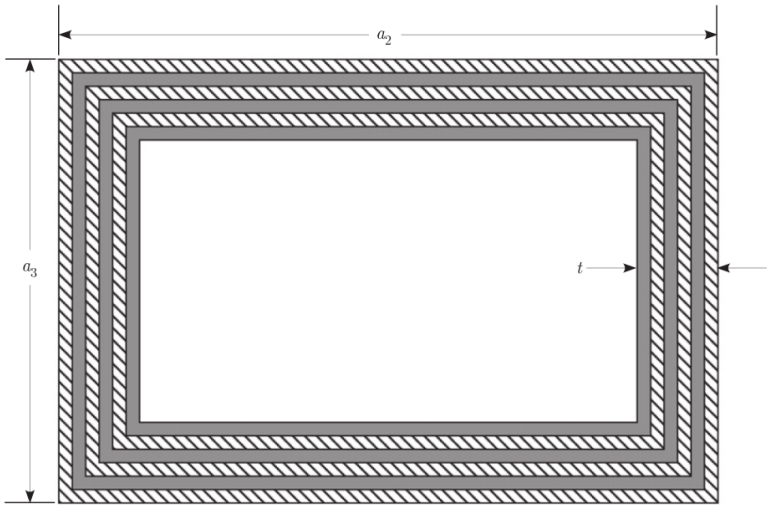

Figure 4.1: Cross section of the box beam [YU2012].

This example is a thin-walled box beam whose cross section is depicted in Fig. 4.1 [YU2012]. The width $a_2 = 0.953$ in, height $a_3 = 0.530$ in, and thickness $t = 0.030$ in. Each wall has six plies of the same composite material and the same fiber orientation of $15°$. Material properties and layup scheme are listed in Table 4.1 and Table 4.2. Cross-sectional properties are given in Table 4.3 and compared with those in Ref. [YU2012]. The tiny differences are due to different meshes. Complete input files can be found in `examples\ex_box\`, including `box.xml` and `materials.xml`.

Table 4.1: Material properties

Name	Density	E_1	E_2	E_3	G_{12}	G_{13}	G_{23}	ν_{12}	ν_{13}	ν_{23}
	$\mathrm{lb}\cdot\mathrm{sec}^2/\mathrm{in}^4$	10^6 psi	10^6 psi	10^6 psi	10^6 psi	10^6 psi	10^6 psi			
mat_1	0.0001353	20.59	1.42	1.42	0.87	0.87	0.696	0.30	0.30	0.34

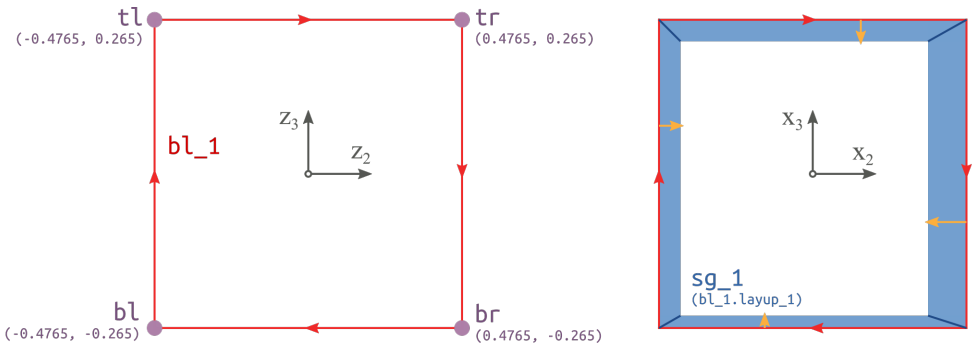

Figure 4.2: *Base point*s, *Base line*s and *Segment*s of the box beam cross section.

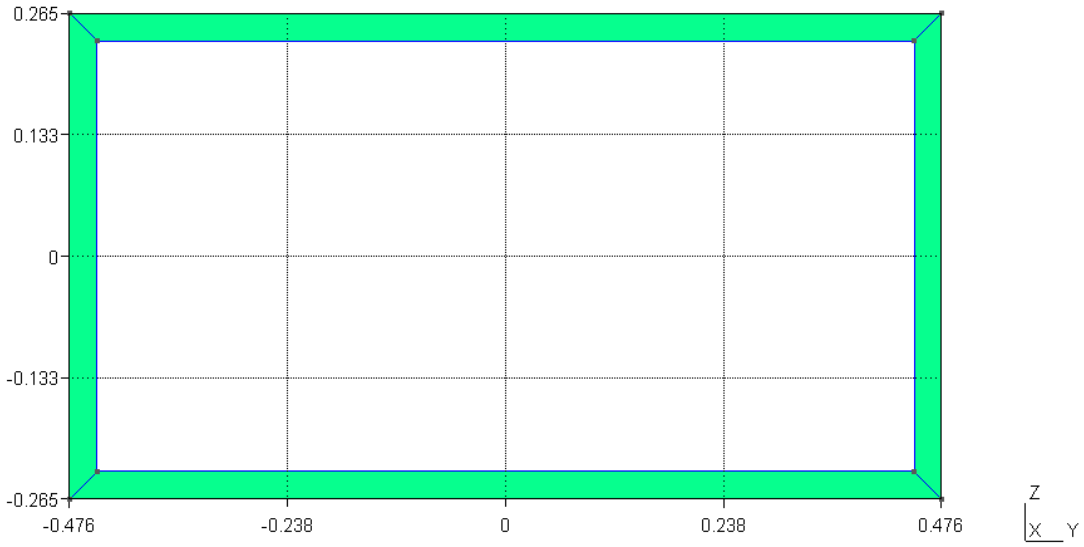

Figure 4.3: Meshed cross section viewed in Gmsh.

Table 4.2: Layups

Name	Layer	Material	Ply thickness	Orientation	Number of plies
			in	○	
layup1	1	mat_1	0.05	-15	6

4.1.2 Result

Table 4.3: Results

Component	Value	Reference [YU2012]
S_{11} [lb]	1.437×10^6	1.437×10^6
S_{22} [lb]	9.026×10^4	9.027×10^4
S_{33} [lb]	3.941×10^4	3.943×10^4
S_{14} [lb · in]	1.074×10^5	1.074×10^5
S_{25} [lb · in]	-5.201×10^4	-5.201×10^4
S_{36} [lb · in]	-5.635×10^4	-5.635×10^4
S_{44} [lb · in^2]	1.679×10^4	1.679×10^4
S_{55} [lb · in^2]	6.621×10^4	6.621×10^4
S_{66} [lb · in^2]	1.725×10^5	1.725×10^5

4.2 Pipe

4.2.1 Probelm description

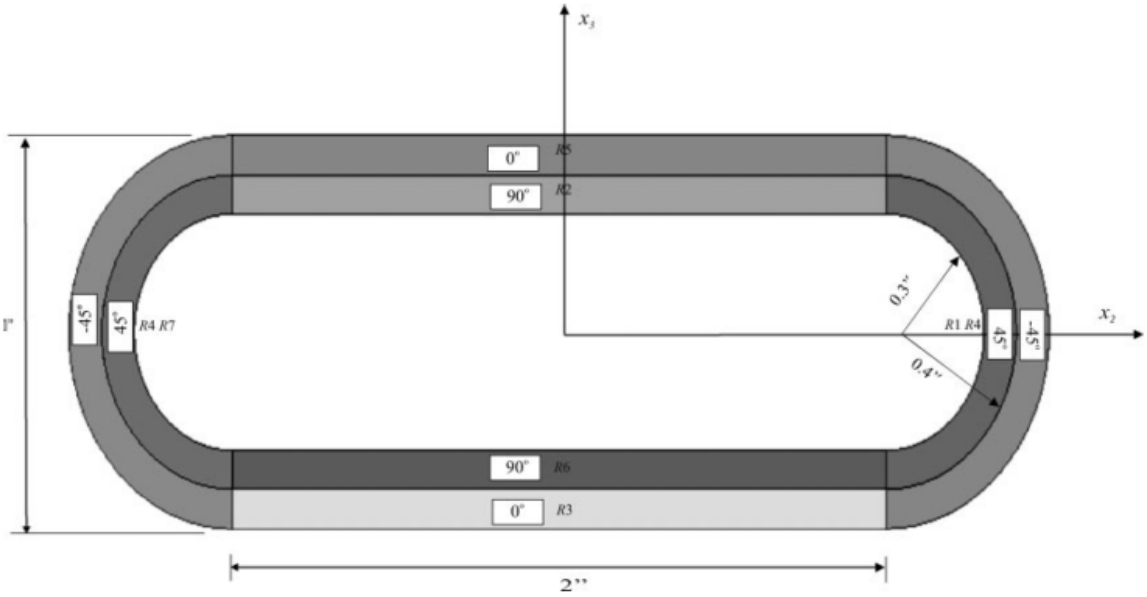

Figure 4.4: Cross section of the pipe [YU2005].

This example has the cross section as shown in Fig. 4.4 [YU2005]. This cross section has two straight walls and two half circular walls. $r = 1.0$ in. and other dimensions are shown in the figure. Each wall has the layup having two layers made from one material. Fiber orientations for each layer are also given in the figure. Material properties

and layups are given in Table 4.4 and Table 4.5. Cross-sectional properties are given in Table 4.6 and compared with the results from [YU2005]. The tiny differences are due to different meshes. Complete input files can be found in `examples\ex_pipe\`, including `pipe.xml`, `baselines.xml`, `materials.xml`, and `layups.xml`.

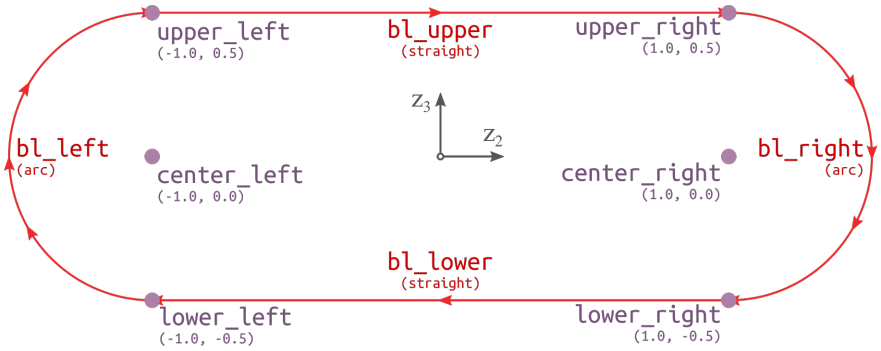

Figure 4.5: *Base point*s and *Base line*s of the pipe cross section.

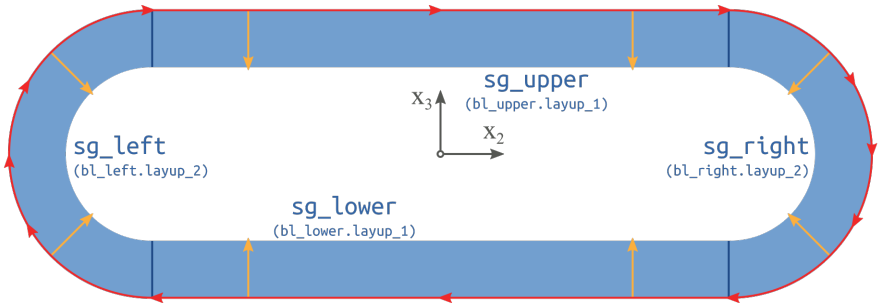

Figure 4.6: *Segment*s of the pipe cross section.

Table 4.4: Material properties

Name	Density	E_1	E_2	E_3	G_{12}	G_{13}	G_{23}	ν_{12}	ν_{13}	ν_{23}
	$\mathrm{lb}\cdot\mathrm{sec}^2/\mathrm{in}^4$	10^6 psi	10^6 psi	10^6 psi	10^6 psi	10^6 psi	10^6 psi			
mat_1	0.057	20.59	1.42	1.42	0.87	0.87	0.87	0.42	0.42	0.42

Table 4.5: Layups

Name	Layer	Material	Ply thickness	Orientation	Number of plies
			in	○	
layup_1	1	mat_1	0.1	0	1
	2	mat_1	0.1	90	1
layup_2	1	mat_1	0.1	-45	1
	2	mat_1	0.1	45	1

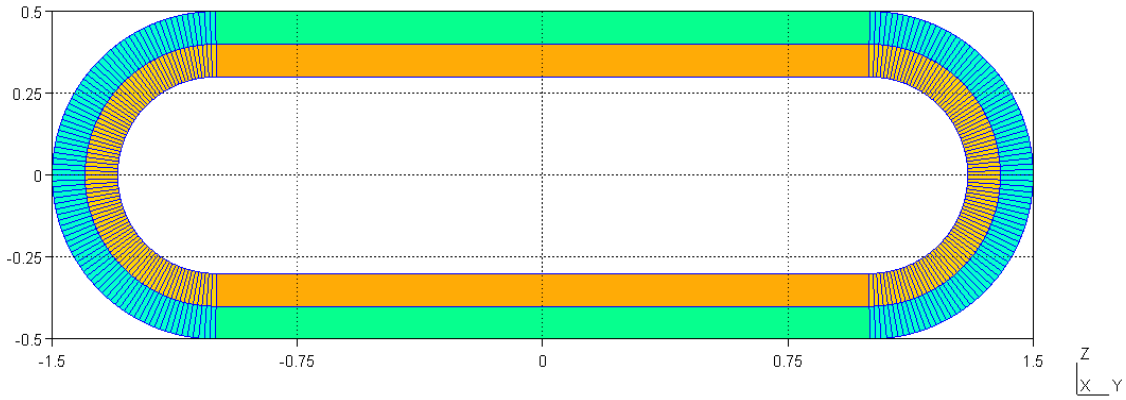

Figure 4.7: Meshed cross section viewed in Gmsh.

4.2.2 Result

Table 4.6: Results

Component	Value	Reference [YU2005]
S_{11} [lb]	1.03892×10^7	1.03890×10^7
S_{22} [lb]	7.85800×10^5	7.84299×10^5
S_{33} [lb]	3.31330×10^5	3.29002×10^5
S_{14} [lb $\cdot$ in]	9.74568×10^4	9.82878×10^4
S_{25} [lb $\cdot$ in]	-8.02785×10^3	-8.18782×10^3
S_{36} [lb $\cdot$ in]	-5.14533×10^4	-5.18541×10^4
S_{44} [lb $\cdot$ in^2]	6.89600×10^5	6.86973×10^5
S_{55} [lb $\cdot$ in^2]	1.88230×10^6	1.88236×10^6
S_{66} [lb $\cdot$ in^2]	5.38985×10^6	5.38972×10^6

4.3 Circular tube

4.3.1 Problem description

This example has a cross section of a simple circular shape with radius $r = 10$ m. This cross section geometry can be defined easily by a center and a radius. Material properties are given in Table 4.7. The layup is defined using the stacking sequence code $[\pm 45_2/0_2/90]_{2s}$. The result is given in Table 4.9. Complete input files can be found in `examples\ex_tube\`, including `tube.xml` and `materials.xml`.

Table 4.7: Material properties

Name	Type	Density	E_1	E_2	E_3	G_{12}	G_{13}	G_{23}	ν_{12}	ν_{13}	ν_{23}
		10^3 kg/m^3	GPa	GPa	GPa	GPa	GPa	GPa			
iso5_4	orthotropic	1.664	10.3	10.3	10.3	8.0	8.0	8.0	0.3	0.3	0.3

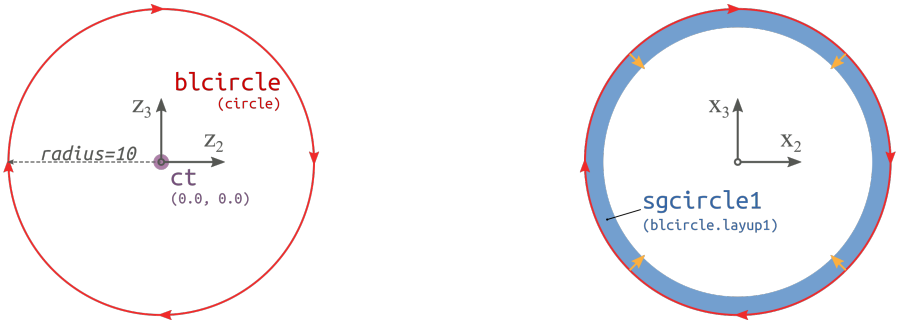

Figure 4.8: *Base point*s, *Base line*s and *Segment*s of the tube cross section.

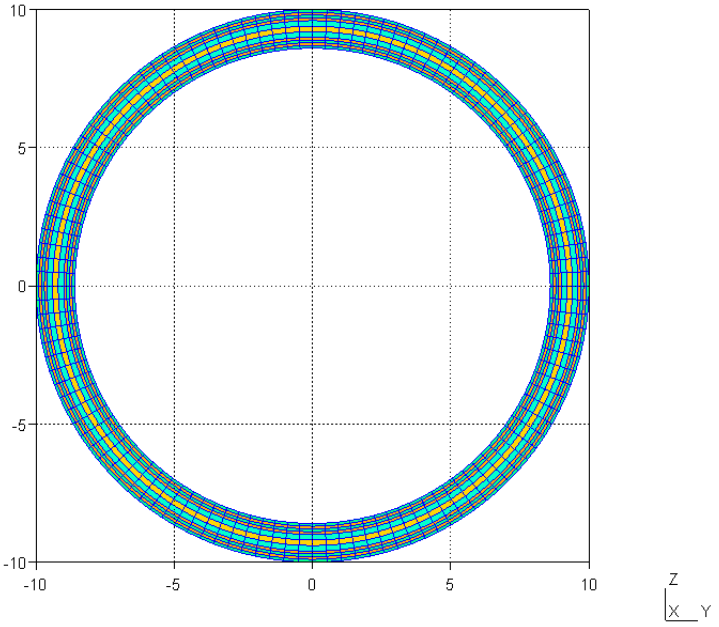

Figure 4.9: Meshed cross section viewed in Gmsh.

Table 4.8: Layups

Name	Material	Stacking sequence
layup1	iso5_4	$[\pm 45_2/0_2/90]_s$

4.3.2 Result

Table 4.9: Results

1.108×10^{12}	-2.677×10^{-3}	-1.050×10^{-4}	-5.795×10^{-5}	-2.099×10^{5}	-1.626×10^{5}
-2.677×10^{-3}	2.352×10^{11}	-1.583×10^{3}	4.781×10^{4}	3.200×10^{-3}	2.063×10^{-2}
-1.050×10^{-4}	-1.583×10^{3}	2.352×10^{11}	4.086×10^{4}	6.546×10^{-4}	1.063×10^{-3}
-5.795×10^{-5}	4.781×10^{4}	4.086×10^{4}	4.043×10^{13}	2.717×10^{-7}	3.229×10^{-8}
-2.099×10^{5}	3.200×10^{-3}	6.546×10^{-4}	2.717×10^{-7}	4.819×10^{13}	-1.399×10^{6}
-1.626×10^{5}	2.063×10^{-2}	1.063×10^{-3}	3.229×10^{-8}	-1.399×10^{6}	4.819×10^{13}

4.4 Channel

4.4.1 Problem description

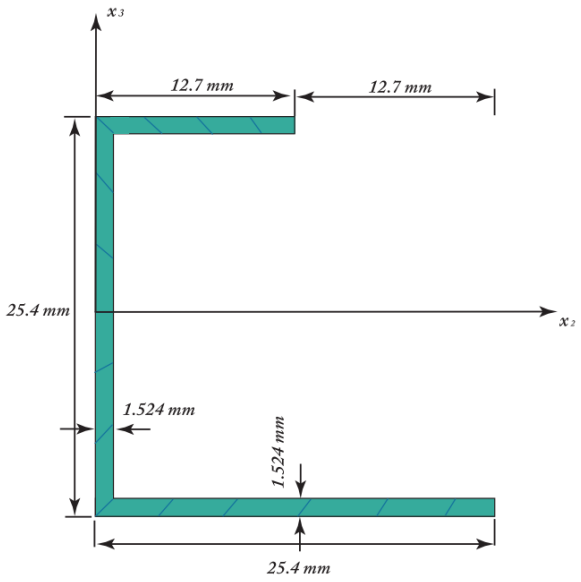

Figure 4.10: Cross section of the pipe [CHEN2010].

This example has a cross section of a highly heterogeneous channel. This cross section geometry can be defined as shown in Fig. 4.10 [CHEN2010]. The isotropic material properties are given in Table 4.10. The layup is defined having a single layer with the thickness 0.001524 m. The result is shown in Table 4.12 and compared with those in [CHEN2010]. Complete input files can be found in `examples\ex_channel\`, including `channel.xml` and `materials.xml`.

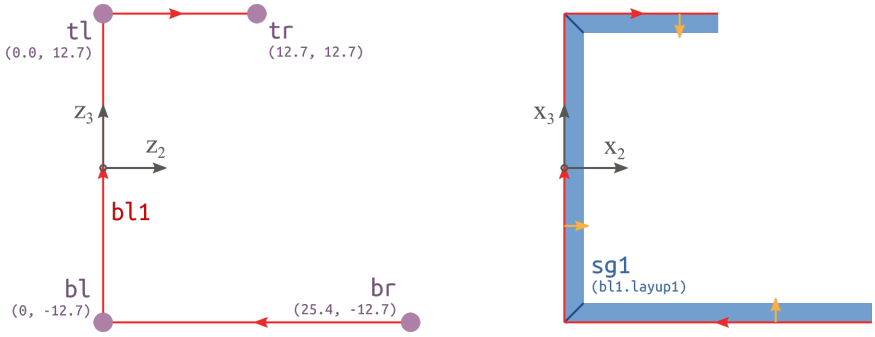

Figure 4.11: *Base point*s, *Base line*s and *Segment*s of the channel cross section.

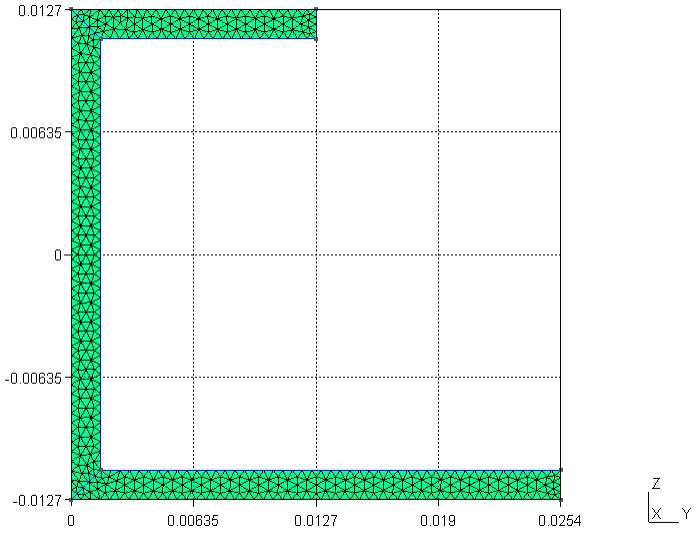

Figure 4.12: Meshed cross section viewed in Gmsh.

Table 4.10: Material properties

Name	Type	Density	E	ν
		$\mathrm{kg/m^3}$	GPa	
mtr1	isotropic	1068.69	206.843	0.49

Table 4.11: Layups

Name	Layer	Material	Ply thickness	Orientation	Number of plies
			m	∘	
layup1	1	mtr1	0.001524	0	1

4.4.2 Result

Table 4.12: Results

1.906×10^7	0.0	0.0	0.0	-4.779×10^4	-1.325×10^5
0.0	2.804×10^6	2.417×10^5	2.128×10^4	0.0	0.0
0.0	2.417×10^5	2.146×10^6	-7.663×10^3	0.0	0.0
0.0	2.128×10^4	-7.663×10^3	2.091×10^2	0.0	0.0
-4.779×10^4	0.0	0.0	0.0	2.011×10^3	9.104×10^2
-1.325×10^5	0.0	0.0	0.0	9.104×10^2	1.946×10^3

Table 4.13: Results from reference [CHEN2010]

1.903×10^7	0.0	0.0	0.0	-4.778×10^4	-1.325×10^5
0.0	2.791×10^6	2.364×10^5	2.122×10^4	0.0	0.0
0.0	2.364×10^5	2.137×10^6	-7.679×10^3	0.0	0.0
0.0	2.122×10^4	-7.679×10^3	2.086×10^2	0.0	0.0
-4.778×10^4	0.0	0.0	0.0	2.010×10^3	9.102×10^2
-1.325×10^5	0.0	0.0	0.0	9.102×10^2	1.944×10^3

4.5 Airfoil (MH-104)

4.5.1 Problem description

This example demonstrates the capability of building a cross section having an airfoil shape, which is commonly seen on wind turbine blades or helicopter rotor blades. This example is also studied in [CHEN2010]. A sketch of a cross section for a typical wind turbine blade is shown in Fig. 4.13. The airfoil is MH 104 (http://m-selig.ae.illinois.edu/ads/coord_database.html#M). In this example, the chord length $CL = 1.9$ m. The origin O is set to the point at 1/4 of the chord. Twist angle θ is 0°. There are two webs, whose right boundaries are at the 20% and 50% location of the chord, respectively. Both low pressure and high pressure surfaces have four segments. The dividing points between segments are listed in Table 4.14. Materials are given in Table 4.15 and layups are given in Table 4.16. A complete 6×6 stiffness matrix is given in Table 4.17. Complete input files can be found in `examples\ex_airfoil\`, including `mh104.xml`, `basepoints.dat`, `baselines.xml`, `materials.xml`, and `layups.xml`.

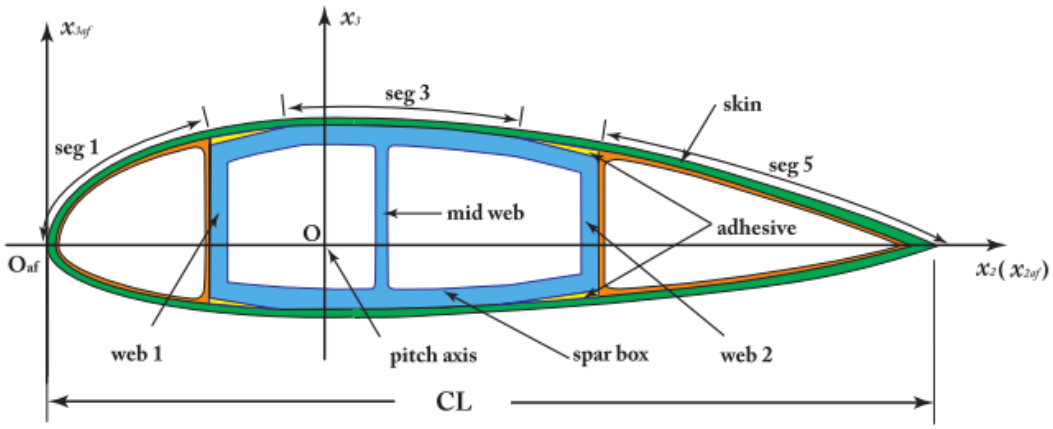

Figure 4.13: Sketch of a cross section for a typical wind turbine blade [CHEN2010].

Table 4.14: Dividing points

Between segments	Low pressure surface	High pressure surface
	(x, y)	(x, y)
1 and 2	(0.004053940, 0.011734800)	(0.006824530, -0.009881650)
2 and 3	(0.114739930, 0.074571970)	(0.126956710, -0.047620490)
3 and 4	(0.536615950, 0.070226120)	(0.542952100, -0.044437080)

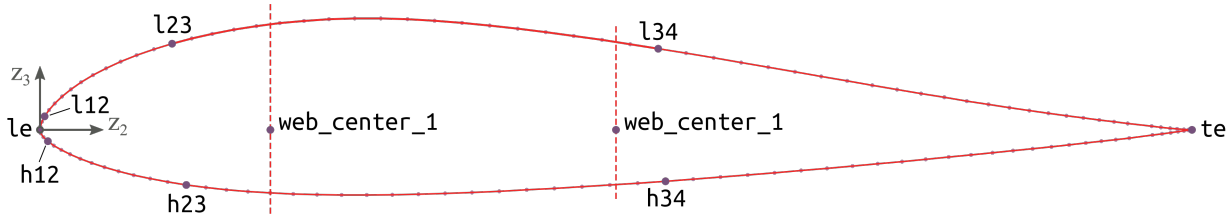

Figure 4.14: *Base point*s of the tube cross section.

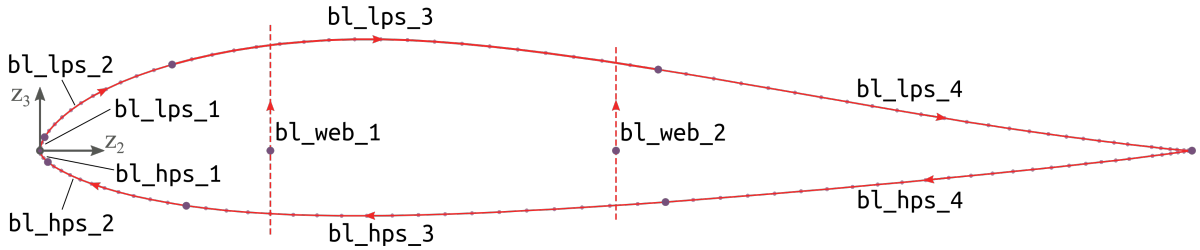

Figure 4.15: *Base line*s of the tube cross section.

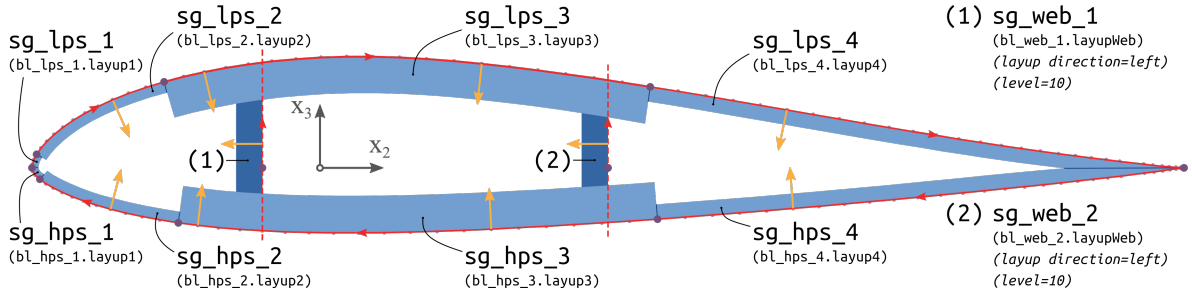

Figure 4.16: *Segment*s of the tube cross section.

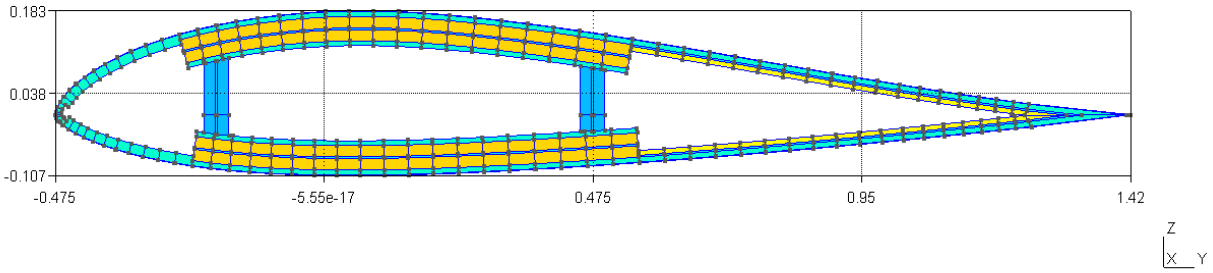

Figure 4.17: Meshed cross section viewed in Gmsh.

Table 4.15: Material properties

Name	Type	Density	E_1	E_2	E_3	G_{12}	G_{13}	G_{23}	ν_{12}	ν_{13}	ν_{23}
		10^3 kg/m^3	GPa	GPa	GPa	GPa	GPa	GPa			
Uni-directional FRP	orthotropic	1.86	37.00	9.00	9.00	4.00	4.00	4.00	0.28	0.28	0.28
Double-bias FRP	orthotropic	1.83	10.30	10.30	10.30	8.00	8.00	8.00	0.30	0.30	0.30
Gelcoat	orthotropic	1.83	1e-8	1e-8	1e-8	1e-9	1e-9	1e-9	0.30	0.30	0.30
Nexus	orthotropic	1.664	10.30	10.30	10.30	8.00	8.00	8.00	0.30	0.30	0.30
Balsa	orthotropic	0.128	0.01	0.01	0.01	2e-4	2e-4	2e-4	0.30	0.30	0.30

Table 4.16: Layups

Name	Layer	Material	Ply thickness	Orientation	Number of plies
			m	◦	
layup_1	1	Gelcoat	0.000381	0	1
	2	Nexus	0.00051	0	1
	3	Double-bias FRP	0.00053	20	18
layup_2	1	Gelcoat	0.000381	0	1
	2	Nexus	0.00051	0	1
	3	Double-bias FRP	0.00053	20	33
layup_3	1	Gelcoat	0.000381	0	1
	2	Nexus	0.00051	0	1
	3	Double-bias FRP	0.00053	20	17
	4	Uni-directional FRP	0.00053	30	38
	5	Balsa	0.003125	0	1
	6	Uni-directional FRP	0.00053	30	37
	7	Double-bias FRP	0.00053	20	16
layup_4	1	Gelcoat	0.000381	0	1
	2	Nexus	0.00051	0	1
	3	Double-bias FRP	0.00053	20	17
	4	Balsa	0.003125	0	1
	5	Double-bias FRP	0.00053	0	16
layup_web	1	Uni-directional FRP	0.00053	0	38
	2	Balsa	0.003125	0	1
	3	Uni-directional FRP	0.00053	0	38

4.5.2 Result

Table 4.17: Effective Timoshenko stiffness matrix

2.395×10^9	1.588×10^6	7.215×10^6	-3.358×10^7	6.993×10^7	-5.556×10^8
1.588×10^6	4.307×10^8	-3.609×10^6	-1.777×10^7	1.507×10^7	2.652×10^5
7.215×10^6	-3.609×10^6	2.828×10^7	8.440×10^5	2.983×10^5	-5.260×10^6
-3.358×10^7	-1.777×10^7	8.440×10^5	2.236×10^7	-2.024×10^6	2.202×10^6
6.993×10^7	1.507×10^7	2.983×10^5	-2.024×10^6	2.144×10^7	-9.137×10^6
-5.556×10^8	2.652×10^5	-5.260×10^6	2.202×10^6	-9.137×10^6	4.823×10^8

Table 4.18: Results from reference [CHEN2010]

2.389×10^9	1.524×10^6	6.734×10^6	-3.382×10^7	-2.627×10^7	-4.736×10^8
1.524×10^6	4.334×10^8	-3.741×10^6	-2.935×10^5	1.527×10^7	3.835×10^5
6.734×10^6	-3.741×10^6	2.743×10^7	-4.592×10^4	-6.869×10^2	-4.742×10^6
-3.382×10^7	-2.935×10^5	-4.592×10^4	2.167×10^7	-6.279×10^4	1.430×10^6
-2.627×10^7	1.527×10^7	-6.869×10^2	-6.279×10^4	1.970×10^7	1.209×10^7
-4.736×10^8	3.835×10^5	-4.742×10^6	1.430×10^6	1.209×10^7	4.406×10^8

Note: The errors between the result and the reference are caused by the difference of modeling of the trailing edge. If reduce the trailing edge skin to a single thin layer, then the difference between the trailing edge shapes is minimized, and the two resulting stiffness matrices are basically the same, as shown in Fig. 4.18.

PreVABS					
2064472125.300	1511727.417	9718692.930	-29031396.199	66725788.529	-248509051.850
1511727.417	502135612.680	-279898.188	-20205923.352	27828620.058	1475369.601
9718692.930	-279898.188	20021647.481	-8727653.954	541726.919	-1738542.290
-29031396.199	-20205923.352	-8727653.954	13509499.286	-2247533.833	4604438.521
66725788.529	27828620.058	541726.919	-2247533.833	21337888.526	-6943938.047
-248509051.850	1475369.601	-1738542.290	4604438.521	-6943938.047	174131287.290
Reference					
2064518914.600	1508938.476	9687610.464	-29015161.717	66726945.698	-248650654.250
1508938.476	502009501.810	-300747.609	-20203039.227	27821546.775	1473462.671
9687610.464	-300747.609	20005103.430	-8714306.591	539559.226	-1733729.206
-29015161.717	-20203039.227	-8714306.591	13500297.561	-2246590.047	4620635.345
66726945.698	27821546.775	539559.226	-2246590.047	21338145.826	-6943939.608
-248650654.250	1473462.671	-1733729.206	4620635.345	-6943939.608	174159526.640
Difference = (PreVABS - Reference) / Reference					
-0.002%	0.185%	0.321%	0.056%	-0.002%	-0.057%
0.185%	0.025%	-6.933%	0.014%	0.025%	0.129%
0.321%	-6.933%	0.083%	0.153%	0.402%	0.278%
0.056%	0.014%	0.153%	0.068%	0.042%	-0.351%
-0.002%	0.025%	0.402%	0.042%	-0.001%	0.000%
-0.057%	0.129%	0.278%	-0.351%	0.000%	-0.016%

Figure 4.18: Comparison of stiffness matrices after modifying the trailing edge.

4.6 Airfoil (Recover)

4.6.1 Problem description

This example continues from the previous one to demonstrate the dehomogenization analysis. It is assumed that a 1D beam analysis has been finished, and data of global deformations and loads have been added to the main input file correctly (*See the recover section*). Suppose that the results in Table 4.19 are used and default values are kept for others. All files generated from the VABS homogenization analysis are kept in the same place as input files. Final visualization is required and the contour plot of one of the stress components is shown in Fig. 4.19. Complete files can be found in `examples\ex_airfoil_r\`, including `mh104.xml`, `basepoints.dat`, `baselines.xml`, `materials.xml`, `layups.xml`, `mh104.sg`, `mh104.sg.ech`, `mh104.sg.K`, `mh104.sg.opt`, `mh104.sg.v0`, `mh104.sg.v1S`, and `mh104.sg.v22`.

Table 4.19: Sectional forces and moments

Quantity	Value
F_1, N	1
F_2, N	2
F_3, N	3
M_1, Nm	4
M_2, Nm	5
M_3, Nm	6

4.6.2 Result

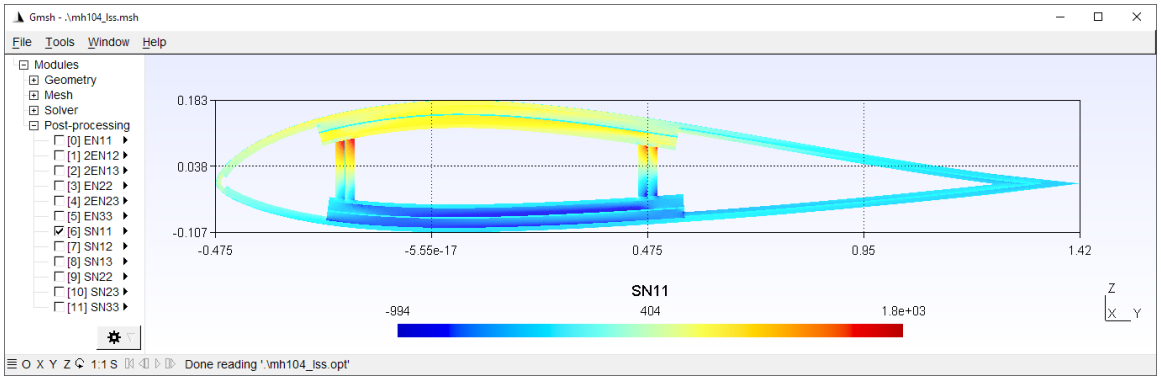

Figure 4.19: Contour plot of recovered nodal stress SN11.

CHAPTER

FIVE

CHANGE LOG

VERSION 1.4

- 1.4.1
  - Fixed an issue in assigning theta1 and theta3 for filling components.
- 1.4.0
  - Added a new command option to run integrated VABS.
  - Added new inputs for the rotor blade specific parameterization.
  - Added new inputs for strength property of 'lamina' type materials.
  - Added new inputs for assigning orientations for filling materials.
  - Added a new input option for recover analysis in VABS. The default is linear beam theory.
  - Added a new input option for setting the tolerance used in geometric computation.
  - Added the capability of failure analysis using VABS.
  - Updated the logging system.
  - Changed the default element type to 'quadratic'.
  - Unified the input syntax of recovery/dehomogenization of VABS/SwfitComp.
  - Fixed the issue of incorrectly parsing numbers with more than one spaces between numbers.
  - Fixed an issue in transforming an arc defined by center, start and radius.

VERSION 1.3

- 1.3.0
  - Added a new capability to create base points using normalized parametric locations on a base line.
  - Added a new capability to assign local mesh size for filling components.

VERSION 1.2

- 1.2.0
  - Added a new capability to create layups from sublayups.
  - Added a new capability to create segments using normalized parametric locations on a base line.

VERSION 1.1

- 1.1.1 (10/28/2020)
  - Fixed the problem of unable to read recover analysis result files on Linux.

  - Fixed a problem when the segment is too short while the laminate is too thick.

- 1.1.0 (10/15/2020)
  - Added a new format for the input file. Now baselines and layups data are merged into the main input file, to reduce the number of input files needed.
  - Added a new material type 'lamina' accepting four numbers for elastic properties.
  - Added default small numbers for elastic properties for isotropic materials.
  - Updated the fill-type component for non-structural mass.

VERSION 1.0 (07/01/2019)

- Added capability to create quadratic triangle elements. In the main input file, under the xml element <*general*>, create a sub element <*element_type*> and set it to *quadratic* or *2*. Default is *linear*.
- Added a new output file **.txt* storing all running messages.
- Changed one of the layup methods tag name from 'explict list' ('el') to 'layer list' ('ll').
- Fixed the crashing issue caused by zero number of layers.
- Fixed the bug that sectional loads were read and written to the distributed loads when doing recovery using VABS.
- Fixed the bug that local lamina data does not overwrite global data.

VERSION 0.6 (07/01/2018)

- Added xml elements in the main input file for various options of VABS/SwiftComp execution.
- Added xml elements in the main input file for failure analysis of SwiftComp.
- Added a <*basepoints*> element in the baseline file. Now for simple shapes, users do not need to use an extra basepoint file, which can still be included for long point list.
- Added a material database file along with the executable. Now PreVABS will look for materials in this file by default.
- Changed Gmsh library to dynamic/shared library.

VERSION 0.5 (01/31/2018)

- Added the capability to create nose mass in an airfoil type cross section.
- Added the post-processing function to visualize the recovered strains and stresses in Gmsh.

VERSION 0.4 (12/04/2017)

- Added the capability to read stacking sequence code.
- Added the parameter to set the number of straight lines to approximate an arc or circle.
- Changed the Gmsh input file name to **.msh*, where * is the cross section name.

VERSION 0.3 (11/27/2017)

- Updated the manual.
- Changed the default behaviour of the command with input file specified only to preparing VABS input (without running VABS).
- Changed the element tag <*origin*> to <*translate*>. Now the two numbers in this element moves base points and base lines, instead of the origin.
- Changed the element tag <*rotation*> to <*rotate*>.

- Changed the 'level' of a segment from an element to an attribute, with default 1.
- Changed the 'layup_direction' of a segment from an element to an attribute of the layup, with default 'right'.
- Changed the material type 'engineering constants' to 'orthotropic'.
- Set the default element type as 'quadratic'.
- Set the default mesh size to the smallest layer thickness.

CHAPTER

SIX

INTRODUCTION TO XML

1. The content of an XML file is composed of elements. Each element is marked by a pair of tags `<tag>...</tag>`.
2. The hierarchical structure of elements is important. At the same level of hierarchy, the arrangement order of elements is not important.
3. The tag names are keywords, which should not be changed.
4. Each element can have multiple attributes, which are `name="value"` pairs. The names are also keywords. The values must be surrounded by double quotes.

CHAPTER

SEVEN

REFERENCES

BIBLIOGRAPHY

[YU2012] Yu, W., Hodges, D. H. and Ho, J. C. (2012) 'Variational asymptotic beam sectional analysis – An updated version', International Journal of Engineering Science, 59, pp. 40–64.

[YU2005] Yu, W. and Hodges, D. H. (2005) 'Generalized Timoshenko Theory of the Variational Asymptotic Beam Sectional Analysis', Journal of the American Helicopter Society, 50(1), pp. 46–55.

[CHEN2010] Chen, H., Yu, W. and Capellaro, M. (2010) 'A critical assessment of computer tools for calculating composite wind turbine blade properties', Wind Energy, 13(6), pp. 497–516.