NASA/TM-20240013073

Filament Wound Composite Analysis Using the NASA Multiscale Analysis Tool (NASMAT) and Finite Element Analysis

Marcus R. Welsh and Kumar C. Jois
Institut für Textiltechnik der RWTH Aachen University, Aachen, Germany

Brett A. Bednarcyk and Trenton M. Ricks
Glenn Research Center, Cleveland, Ohio

November 2024

NASA STI Program Report Series

Since its founding, NASA has been dedicated to the advancement of aeronautics and space science. The NASA scientific and technical information (STI) program plays a key part in helping NASA maintain this important role.

The NASA STI program operates under the auspices of the Agency Chief Information Officer. It collects, organizes, provides for archiving, and disseminates NASA's STI. The NASA STI program provides access to the NTRS Registered and its public interface, the NASA Technical Reports Server, thus providing one of the largest collections of aeronautical and space science STI in the world. Results are published in both non-NASA channels and by NASA in the NASA STI Report Series, which includes the following report types:

- TECHNICAL PUBLICATION.
  Reports of completed research or a major significant phase of research that present the results of NASA programs and include extensive data or theoretical analysis. Includes compilations of significant scientific and technical data and information deemed to be of continuing reference value. NASA counterpart of peer-reviewed formal professional papers but has less stringent limitations on manuscript length and extent of graphic presentations.

- TECHNICAL MEMORANDUM.
  Scientific and technical findings that are preliminary or of specialized interest, e.g., quick release reports, working papers, and bibliographies that contain minimal annotation. Does not contain extensive analysis.

- CONTRACTOR REPORT.
  Scientific and technical findings by NASA-sponsored contractors and grantees.

- CONFERENCE PUBLICATION.
  Collected papers from scientific and technical conferences, symposia, seminars, or other meetings sponsored or cosponsored by NASA.

- SPECIAL PUBLICATION.
  Scientific, technical, or historical information from NASA programs, projects, and missions, often concerned with subjects having substantial public interest.

- TECHNICAL TRANSLATION.
  English-language translations of foreign scientific and technical material pertinent to NASA's mission.

Specialized services also include organizing and publishing research results, distributing specialized research announcements and feeds, providing information desk and personal search support, and enabling data exchange services.

For more information about the NASA STI program, see the following:

- Access the NASA STI program home page at http://www.sti.nasa.gov

NASA/TM-20240013073

Filament Wound Composite Analysis Using the NASA Multiscale Analysis Tool (NASMAT) and Finite Element Analysis

Marcus R. Welsh and Kumar C. Jois
Institut für Textiltechnik der RWTH Aachen University, Aachen, Germany

Brett A. Bednarcyk and Trenton M. Ricks
Glenn Research Center, Cleveland, Ohio

National Aeronautics and
Space Administration

Glenn Research Center
Cleveland, Ohio 44135

November 2024

Acknowledgments

This work was supported by the Cryotank Technology for Exploration Applications (CTE-A) Project within the Space Technology Mission Directorate (STMD) Game Changing Development (GCD) Program. Special thanks to Sandi G. Miller and Derek J. Quade at NASA Glenn, Marc R. Schultz at NASA Langley, and John C. Fikes at NASA Marshall.

Level of Review: This material has been technically reviewed by technical management.

This report is available in electronic form at https://www.sti.nasa.gov/ and https://ntrs.nasa.gov/

NASA STI Program/Mail Stop 050
NASA Langley Research Center
Hampton, VA 23681-2199

Contents

Filament Wound Composite Analysis Using the NASA Multiscale Analysis Tool (NASMAT) and Finite Element Analysis

Marcus R. Welsh and Kumar C. Jois
Institut für Textiltechnik der RWTH Aachen University
52074 Aachen, Germany

Brett A. Bednarcyk and Trenton M. Ricks
National Aeronautics and Space Administration
Glenn Research Center
Cleveland, Ohio 44135

Abstract

Fiber reinforced composite materials, owing to their tailorable thermomechanical and functional properties, allow one to produce a structure that is stronger, stiffer, and lighter than its metal counterpart while performing the same function, yielding a more efficient structure. This not only allows for the improvement of current technologies like aircraft structures, but also enables new technologies like gaseous hydrogen storage for mobility applications, which are otherwise impractical when manufactured using traditional metals due to weight and space restrictions or material embrittlement. However, the use of composites imposes greater design and manufacturing challenges on an engineer, since they are heterogenous, having a distinct structure across multiple length scale, behave generally anisotropically at the structural level and require complex manufacturing and processing methods. Capturing this complex behavior requires detailed numerical simulations, including the modeling of microstructural features like undulations, voids, and fiber alignment. In this paper, multiple repeating unit cells (RUCs), representing filament wound composites, are developed (via a script provided in the Appendix) and analyzed. The refinement of these RUCs is varied, and the analyses are performed using both the Abaqus finite element software and the NASA Multiscale Analysis Tool (NASMAT). A study is undertaken to compare the predicted effective elastic properties of the wound RUC to a laminate representation of the wound RUC, which neglects the undulations. Additionally, two different sets of periodic boundary conditions (PBCs) have been examined. One approximates the real boundary conditions using a standard approach and the other represents the PBCs exactly through the use of an offset. Lastly, a comparison of the local elastic stress fields is made among the models and approaches. Since wound structures are often approximated as laminated structures, it is important to understand the degree to which this assumption is valid, namely by first comparing the elastic constants and local elastic fields. This will provide, on the one hand, information concerning the bulk mechanical behavior and, on the other hand, insights concerning local load distributions and likely damage initiation sites.

Introduction

Interest in alternative fuel sources, particularly hydrogen, has gained interest in recent years due to its potential to replace current fossil fuel sources in mobile applications. In Germany, the national hydrogen strategy (*Nationale Wasserstoffstratagie*) has set goals for research, development, and implementation of transport infrastructure by the end of the decade. A key enabling technology for *mobile* hydrogen storage is the composite overwrapped pressure vessel (COPV) since weight, size and gravimetric efficiency requirements make traditional materials like aluminum or steel impractical. COPVs are designated into five different types as shown in Figure 1.

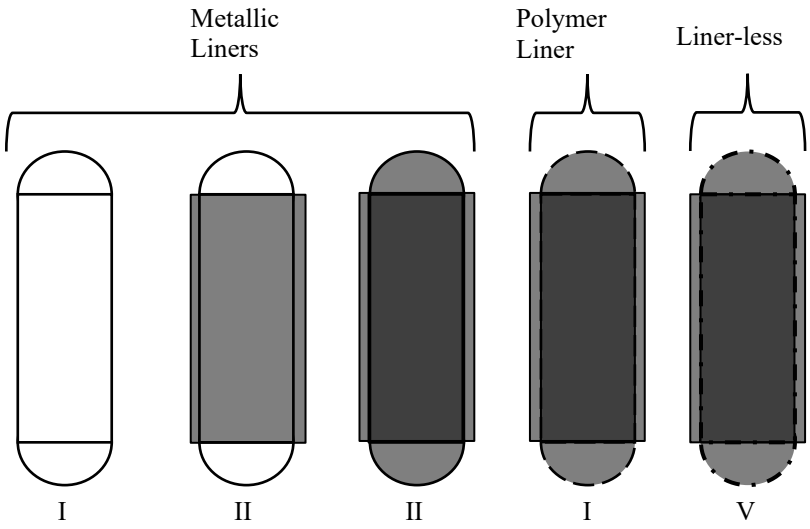

Figure 1.—Pressure vessel designation terminology

Type I is an all-metallic pressure vessel while types II and III have metallic liners with a composite overwrap. The designation type IV refers to vessels that has a polymer liner on which a fiber reinforcement composite architecture will be manufactured; the polymer liner acts as the diffusion barrier while the fiber reinforcement carries the loads. In comparison to types II and III, Type IV is currently the lightest commercially available solution and has garnered most attention / interest. Type V is a liner-less pressure vessel that would be lighter than type IV with a greater gravimetric efficiency, but their production is limited and permeation / leakage through the laminate remains an issue.

Hydrogen can be stored in a gaseous state, a cryo-compressed gaseous state, or as a cryogenic fluid. In this case, the vessel stores gaseous hydrogen at 700 bar (70 MPa) and has a carbon fiber / epoxy composite reinforcement system, which is not only a very stiff material system to prevent excessive deformation, but also very strong as to carry the high mechanical loads. It is also lightweight, making it suitable for mobile applications. To manufacture such a vessel, one can use a variety of processes, but here the towpreg filament winding process is used. Filament winding is a manufacturing process by which a continuous band, composed of multiple yarns or tows, is wound around a mandrel to manufacture the reinforcement architecture (see Figure 2 to Figure 4).

The filament winding process requires the winding head to traverse forwards and backwards – termed the forward stroke and backward stroke – to completely cover the mandrel with material. During the forward stroke the bands are laid with a winding angle, $+\alpha$, while the backward stroke lays the bands at an angle, $-\alpha$, producing a balanced configuration after having covered the entire surface (Figure 3). The solid lines represent the forward stroke ($+\alpha$) and the dashed lines the backward stroke ($-\alpha$). This process is then repeated for the number of winding angles in the layup until the laminate is complete. Filament winding is generally used to manufacturing axisymmetric parts, like tanks and pipes, but it can also be used to manufacture composite parts having non-axisymmetric geometries.

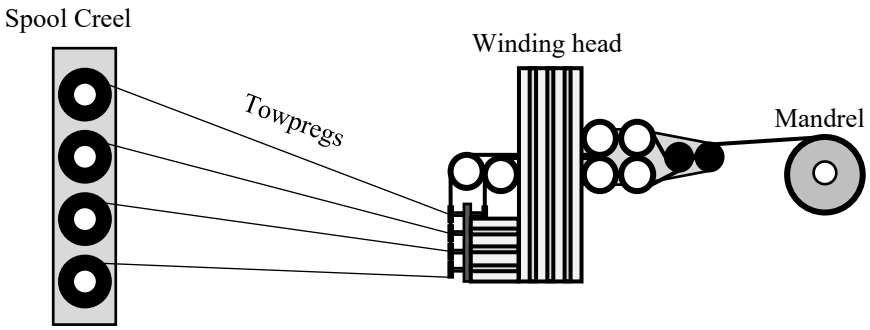

Figure 2.—Filament winding process sketch.

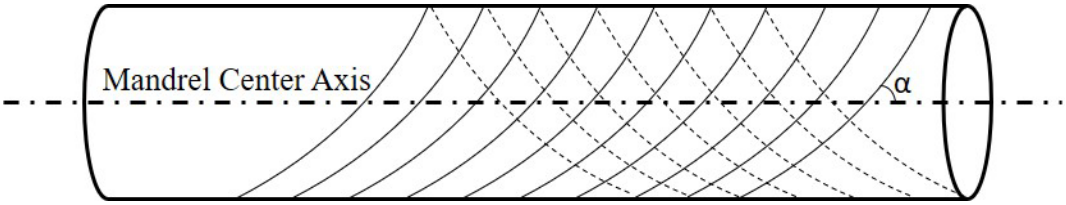

Figure 3.—Winding angle definition on mandrel.

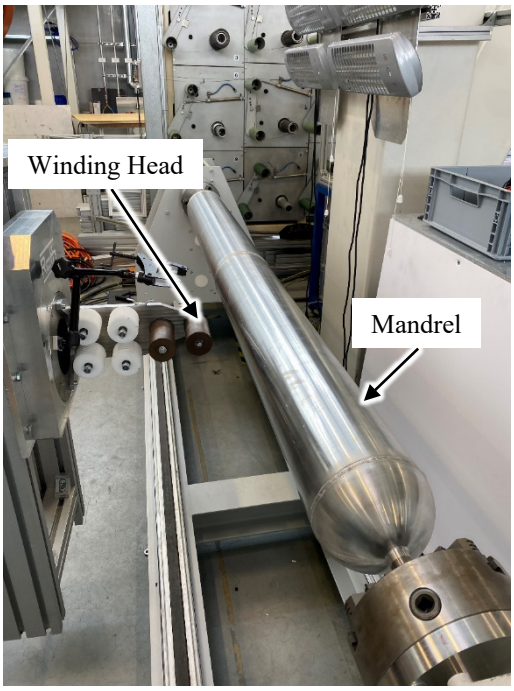

Figure 4.—Filament winding cell at RWTH Aachen University Institut für Textiltechnik (ITA).

Preliminary design and sizing of composite pressure vessels and pipes generally starts with netting theory (Tew, 1995) and classical lamination theory (CLT) (Jones, 1999), where the former provides an initial thickness that can be used in CLT as a starting point during laminate design. After some iteration, an initial laminate design is determined, usually by means of stress analysis with CLT subject to first ply failure criteria. CLT can even be extended, for example, to include damage and degradation of the *material* (reflected in updated entries of the ABD-matrix) (Schürmann, 2007), and/or to obtain a more physically correct *load distribution* (reflected in the load vector by using, for example, the bending theory of shells to capture the combined effects of membrane and flexural loading) (Eschenauer et. al., 1997; Vinson, 1993; Mittelstedt, 2021).

For a more detailed numerical analysis, a finite element model (FEM) can be used to create a 2D axisymmetric model of a pressure vessel or even a full 3D model, increasing progressively the computational expense with increased fidelity. Here, one can accurately capture thick wall effects, g5enerally not captured in lamination theory, as well as delamination (by use of cohesive surfaces), material defects, and damage using user materials and subroutines.

A key microstructural feature that is often neglected during the modeling of wound composite structures is the undulation pattern that emerges due to bands overlapping (Figure 6). Undulation refers to the overlap that occurs during the manufacturing process and can lead to locally reduced stiffness and, critically, stress raisers, which can be damage initiation sites. Capturing the undulating winding pattern with appropriate contact conditions at the global scale quickly becomes computationally intractable and can be extremely time intensive. Thus, to simplify the problem, a *wound ply* with $\pm\alpha$ is typically separated into two *laminated plies*, with one ply of $+\alpha$ and another of $-\alpha$, maintaining balance globally, but neglecting any local effects of the undulation that can lead to damage onset and propagation in the structure (Morozov, E.V., 2006). Obviously, for cyclic loading, stress raisers from the undulations become even more important.

To investigate the effects of undulation in wound composite materials, a python script has been developed that generates a wound undulation pattern within the open-source software TexGen (Brown and Long, 2021) (see Figure 5). The script enables the user to define the yarn (tow) characteristics, the spacing between yarns (tows), the number of yarns (tows) in a band, the spacing between bands, the number of bands in the pattern, and the angle at which the bands are wound, allowing for extensive modeling freedom and parametric analysis. The script correctly places band undulations in the pattern, accurately reflecting the geometry of the wound composite that results from manufacturing. Note that the script is given in the Appendix. The geometry generated by the script can then be analyzed using other software, such as finite element codes.

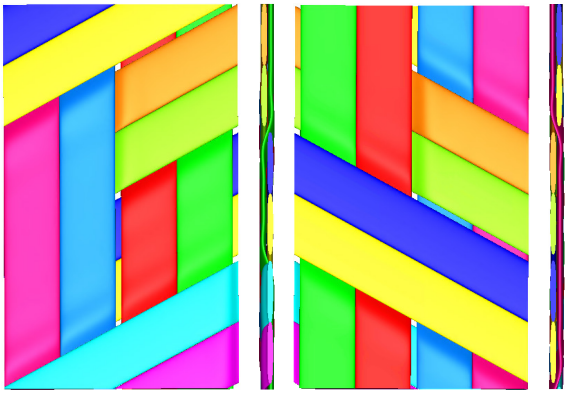

Figure 5.—Wound RUC geometry generated in TexGen.

Herein, this python script has been applied to create several wound repeating unit cells (RUCs) with a layup pattern of $[\pm 30/90_2]$. These RUCs are then analyzed using TexGen in combination with the Abaqus finite element software (Dassault Systemes, 2024), as well as the NASA Multiscale Analysis Tool (NASMAT) (NASA, 2024). The analysis predictions are compared to each other, and also with a laminated RUC having the same layup to assess the impact of including the yarn/band undulations in the composite microstructure. Results include the predicted effective engineering constants of the composites, as well as the predicted local elastic stress fields. Within the finite element results, a comparison between standard and more real offset periodic boundary conditions (PBCs) has also been conducted. The goal of this study is thus to examine, evaluate, and compare quantitatively the following aspects of modeling wound composites: (1) the effect of undulations, (2) the effect of model type (finite element vs. method of cells), and (3) the effect of simplified vs. real PBCs.

The manuscript is organized as follows: First, the python script for generating a wound RUC in TexGen will be explained, followed by a brief discussion about the different periodicity conditions applied during this study. Next, a brief introduction to the 3D High-Fidelity Generalized Method of Cells (HFGMC) homogenization theory used in NASMAT will be provided, after which the predicted effective engineering constants in Abaqus and NASMAT will be discussed. After this, the local elastic stress fields from NASMAT and Abaqus will be discussed, followed by a conclusion and outlook.

Filament Wound RUC Creation

Filament Wound Pattern Generation

In the filament winding process, a band, consisting of multiple yarns (Figure 6), is wound around a rotating mandrel until the surface of the mandrel is completely covered. During this time, the winding head traverses forwards and backwards to deposit the band on the rotating mandrel, ultimately completing one cycle when the winding head returns to its initial position (see Figure 4). While the winding head traverses forwards and backwards, the band will cover previously deposited material, resulting in points of overlap. In order to achieve full coverage of the mandrel (and complete one ply), multiple cycles must be repeated, whereby the winding head again traverses forwards and backwards, depositing more material and creating more points of overlap. Depending on winding angle and mandrel geometry, a surface pattern will emerge, containing a certain number of overlaps (Figure 6).

Figure 7 shows a sample winding pattern with the RUC represented as a black square. The picture on the left side of Figure 7 is the winding pattern in global coordinates, where the horizontal axis is aligned with the center axis of the mandrel. In this configuration, one can see a repeating diamond pattern which is not orthogonal to the global coordinate system. To solve this, the pattern is rotated by the winding angle, α, such that one side of the pattern is parallel with the ordinate. The height of the RUC was then determined by aligning the top left and bottom left corners of the RUC with the top edge of the upper and lower angled bands in the RUC (dashed lines), respectively.

The regions of overlap cause the bands to undulate slightly out of plane, decreasing locally the ply effective stiffness. Additionally, undulation can result in matrix rich regions and air pockets (voids), both of which lead to local stress concentrations and can serve as crack nucleation sites (see Figure 8).

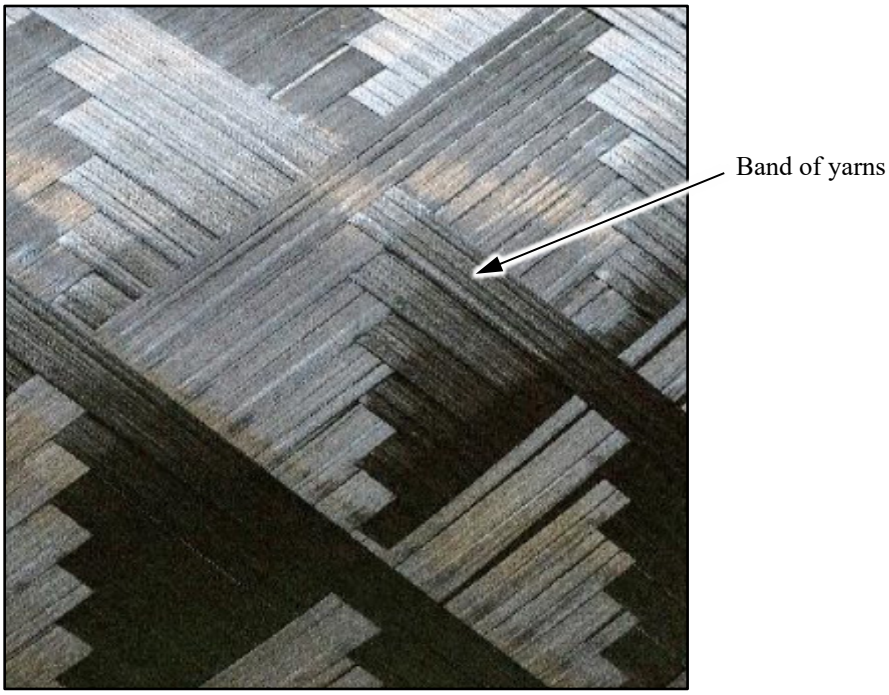

Figure 6.—Filament wound surface pattern.

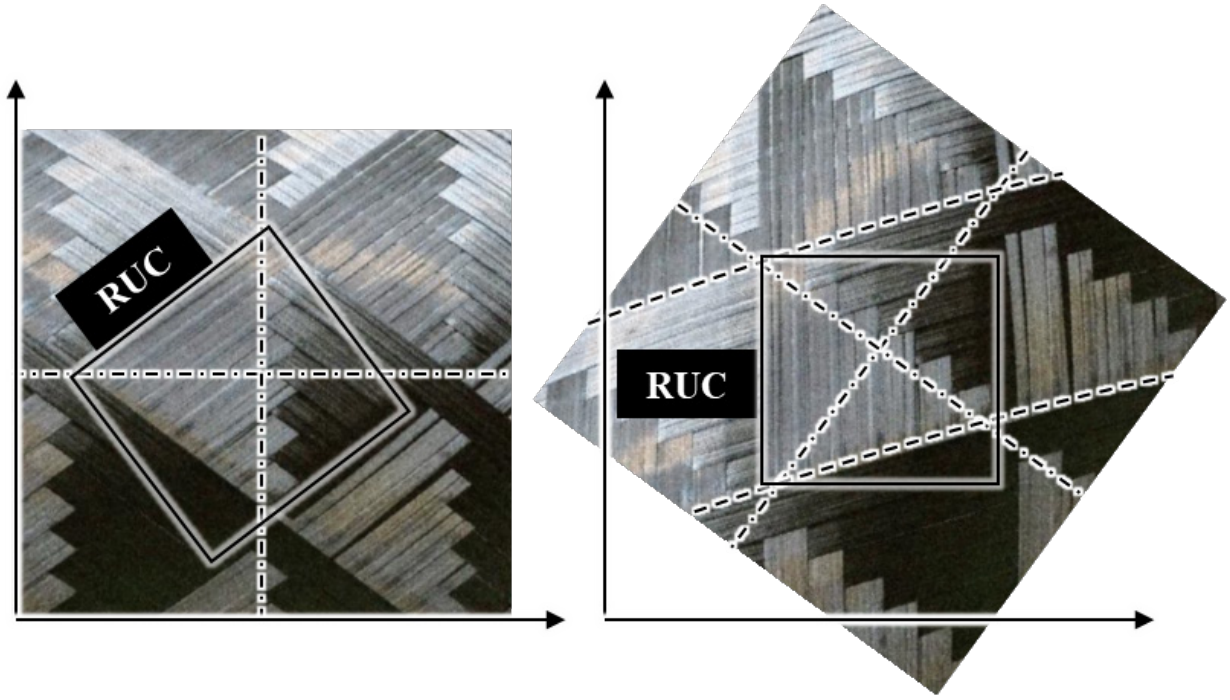

Figure 7.—RUC in global coordinates (left) and local coordinates (right).

Figure 8.—Computer tomography (CT) scan of tube cross section with notable undulation.

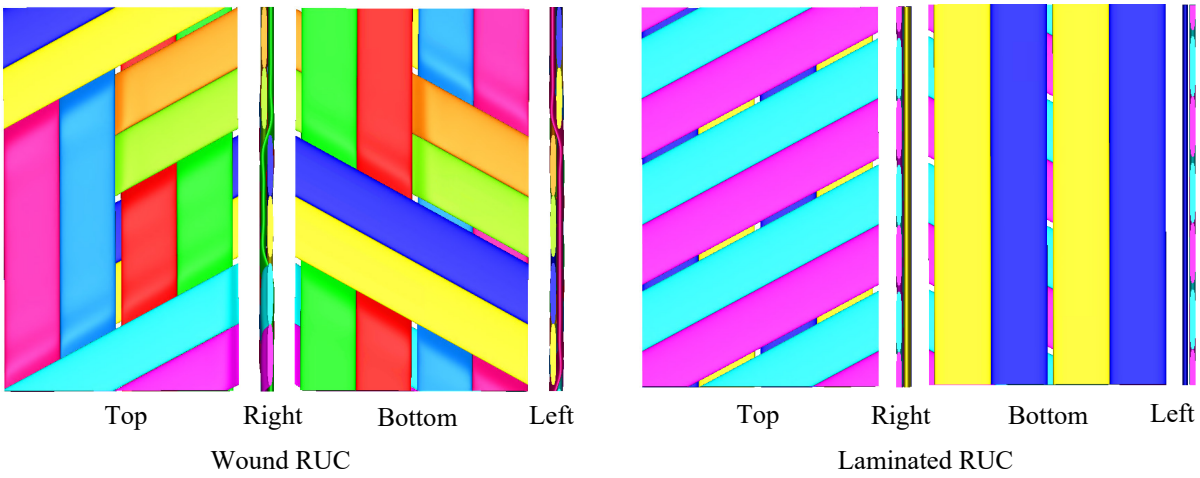

Figure 9.—TexGen [±30] filament wound RUC (left) and laminated RUC (right).

The RUCs in Figure 9 represent a [±30] layup (left) and the laminated equivalent (right). The wound pattern possesses bands undulating through the thickness of the RUC, leading to regions of overlap that are not present in the laminated RUC; the different color yarns in each band simply represent different yarn instances in the TexGen software. Additionally, the regions of overlap result in a macroscopic diamond pattern in the middle of the unit cell, the presence of which will influence the local stress fields upon load application.

Subject to analysis in this manuscript are the RUCs in Figure 10, both possessing hoop layers in addition to the helical 30° layer, resulting in a $[\pm 30/90_2]$ layup. This layup was chosen since tubular specimen at ITA were manufactured and tested according to ASTM D2290 with this same layup, providing experimental results as comparison for future work. The python script currently instantiates a flat RUC, different from the filament wound rings which have some curvature. The influence of curvature is generally neglected if the thickness-to-radius ratio (t/R) is less than 1/15. In this case we manufactured ring specimen having an average thickness of 2.0 mm and a radius of 50 mm, thus t/R is sufficiently thin, having a ratio of 1/25. If the ratio of t/R is greater than 1/15, then curvature must be considered due to the presence of a non-negligible through thickness stress component.

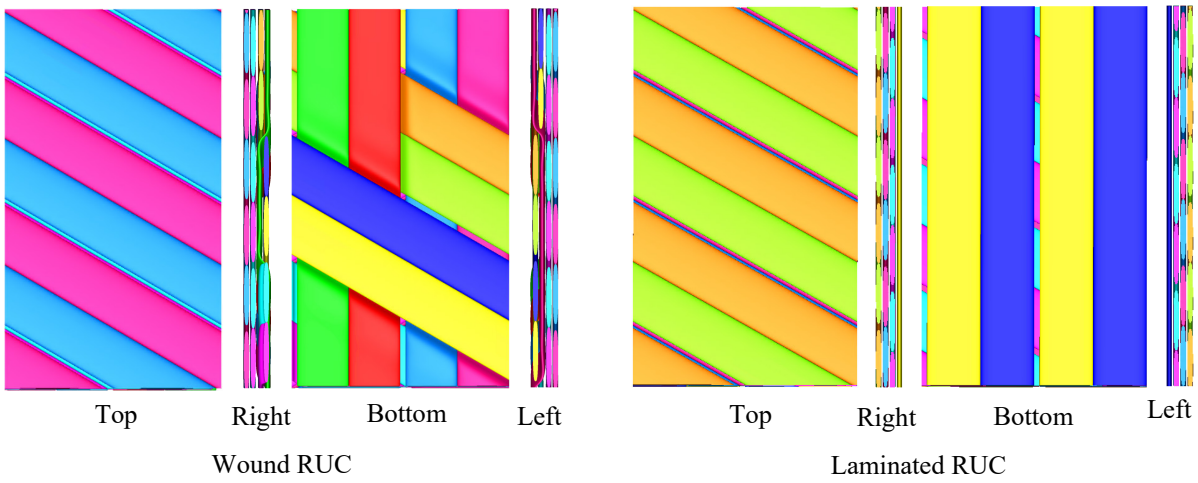

Figure 10.—TexGen [±30/90$_2$] filament wound RUC (left) and laminated RUC (right).

RUC Generation Program Explanation

A primitive class in TexGen is a yarn (also known as a tow), having the following attributes: its cross-section shape and its assigned nodes. The assigned nodes define the path in 3D space along which the cross-section is swept (Figure 11). One can define the yarn cross-section shape by specifying the yarn thickness, yarn width and shape factor (for the *PowerEllipse* cross section in TexGen).

When multiple yarns with the same path are combined, a band is formed (Figure 12). In the python script, a band class is defined by the number of yarns in the band, y_c, and the spacing between adjacent yarns, y_s. During the filament winding process, the number of yarns per band can be adjusted for the desired coverage based on the diameter of the mandrel. Additionally, the spacing between adjacent yarns should ideally be zero, but it is often the case that yarns shift relative to one another during manufacturing, resulting in spaces between yarns, or even yarn overlap, after the band has been deposited on the mandrel. Thus, it is important to have yarn spacing as a variable that can be non-zero; yarn overlap between adjacent yarns has not been considered herein. This may be investigated in future work. Additionally, without the band spaces, the band geometry would interfere or merge at the overlap regions, leading to physically inconsistencies like incorrect material orientation assignments in neighboring bands or artificially low volume fractions in the RUC. To address these points, a processing model would need to be developed to account for tow deformation under specific operating conditions, which is a matter for future work.

Finally, the winding pattern (Figure 13) is defined by the number of bands participating in the pattern, n_{bands}, the spacing between adjacent bands, b_s, the pattern angle, α, and out of plane spacing between bands, $b_{gap,z}$. From this, the bandwidth, b_w, can be formulated and is the sum of all yarn widths and yarn spaces in a band (Figure 12). The assumption is that a single wound layer is generated from a single prescribed winding angle; optionally, hoop layers (90° plies) can be added. The script does not currently consider multiple wound layers, with multiple different winding angles; the script is currently limited to just a single wound layer angle.

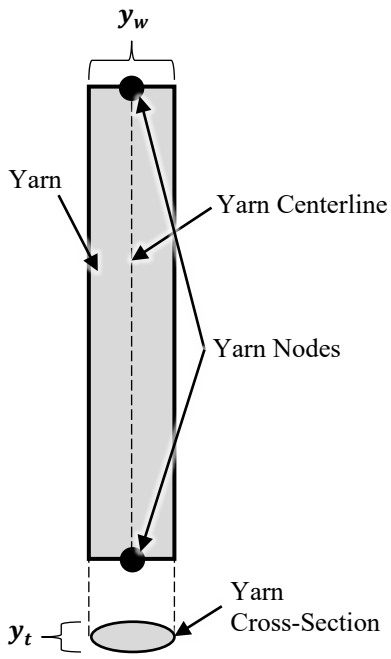

Figure 11.—Yarn definition in python script.

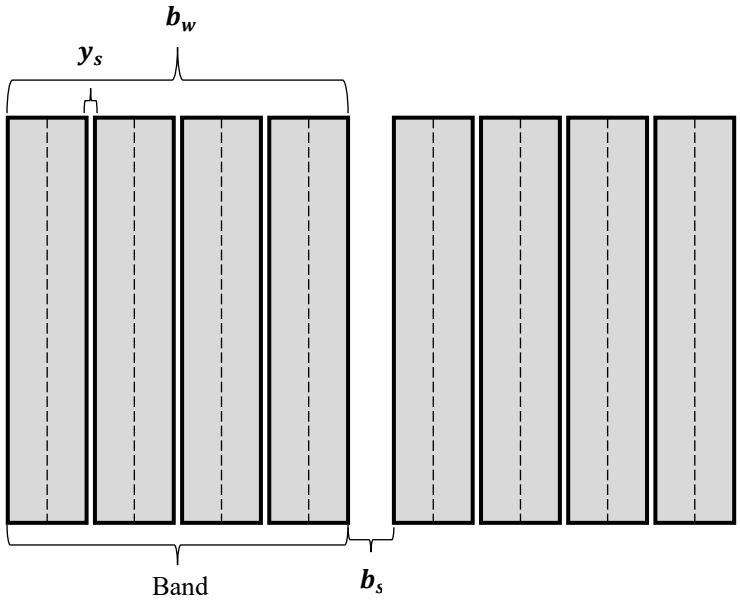

Figure 12.—Band definition in python script.

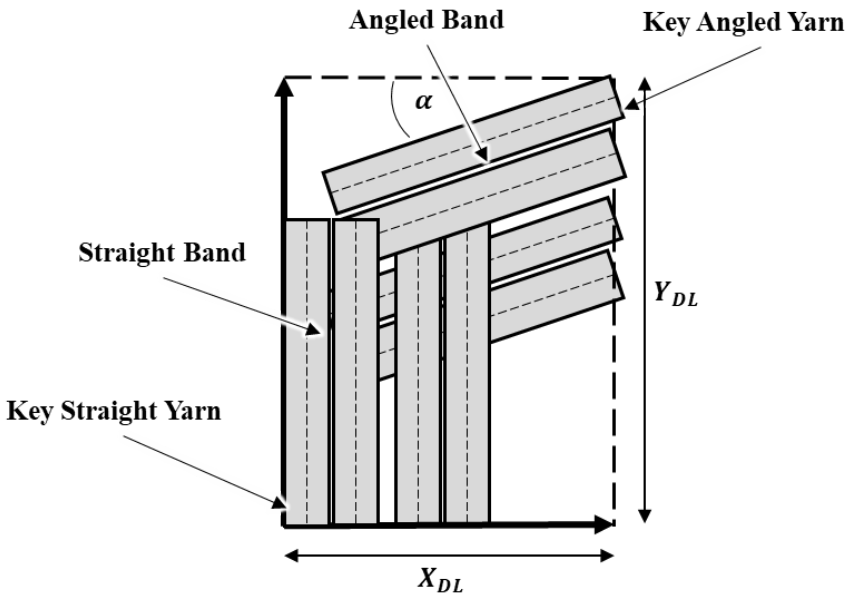

Figure 13.—RUC pattern generation in python script.

The positioning of each band, and thus the position of each yarn in each band, is based on the key yarns, which are the first (leftmost) yarn in the straight bands and the first (topmost) yarn in the angled bands (Figure 13). The initial band positions were chosen, such that the *key straight yarn* lies exactly on the leftmost edge of the domain while the top right corner of the *key angled yarn* coincides with the top right corner of the domain. Furthermore, the RUC domain length in the *x* direction, X_{DL}, is calculated as,

$$X_{DL} = n_{bands}\, b_w + (n_{bands} - 1)\, b_s \tag{1}$$

The RUC domain length in the *y* direction, Y_{DL}, is the same as X_{DL} for $\alpha = 0$. However, if $\alpha \neq 0$, then Y_{DL} is larger by a factor of $1/\cos\alpha$, since a band at angle $\alpha \neq 0$ has a larger projected area onto the *y* axis. That is,

$$Y_{DL} = \frac{X_{DL}}{\cos\alpha} \tag{2}$$

In Figure 14 the initial positioning of the angled bands and angled yarns is illustrated. In the figure, the *0th position* of the first angled band, which contains two yarns, is chosen such that it starts on the right most edge of the domain and with the top most yarn in the band coincident with the top right corner of the domain, given by the following equations:

$$y_{0,b,a,i} = Y_{DL} - i_b \cdot \frac{b_w + b_s}{\cos\alpha} - \frac{y_w}{2\cos\alpha} \tag{3}$$

$$x_{0,b,a,i} = X_{DL} \tag{4}$$

Where the subscript terminology is as follows: the first index indicates the beginning point (0) or end point (1) of a segment, but *init* is used in a select case (see below). Index 2 indicates whether it is a yarn (*y*) or a band (*b*). Index three indicates if the segment is straight (*s*) or angled (*a*), and the last index, *i*, indicates the yarn number (if index 2 is *y*) or the band number (if index 2 is *b*). Also, when index 2 is *y*,

then the *x*, *y* positions in the following equations refer to yarn nodal positions which can be accessed in TexGen. However, when index 2 is *b*, then the *x*, *y* positions in the following equations refer to band "nodal" positions, but are not associated with nodes accessible in TexGen; they only act as offsets from which the yarn nodes are positioned. Furthermore, i_b is the band index, which begins at zero and increments over the number of bands in the pattern. Thus, when $i_b = 0$, the middle term in *y* coordinate calculation becomes zero and the *y* position is simply the domain length in the *y* direction minus half of the projected area of the key angled yarn, positioning it coincident with the top right point of the domain.

In Figure 14 there are points residing outside of the domain, referred to as the *initial points* ensuring clean cuts when the textile is trimmed to the dimensions of the domain, forming the RUC. The initial *x* position of i^{th} yarn in the angled band is chosen as,

$$x_{init,y,a,i} = 1.25 \cdot X_{DL} \tag{5}$$

ensuring that the initial *x* coordinate lies outside the domain. It is recommended to stay above a factor of 1.25 so that the yarns are completely outside the domain before they are trimmed to the domain size. The initial *y* coordinate is given by,

$$y_{init,y,a,i} = y_{0,y,a,i} + \left(x_{init,y,a,i} - x_{0,y,a,i}\right) \cdot \tan\alpha \tag{6}$$

ensuring an angle α is maintained over the distance $\left(x_{init,y,a,i} - x_{0,y,a,i}\right)$.

Figure 15 shows the next segment to be created by the script. Here, the yarn end positions $(x_{1,y,a,i}, y_{1,y,a,i})$ need to be determined. These coordinates are meant to position the ends of the angled yarns at known points in the domain relative to the straight yarns.

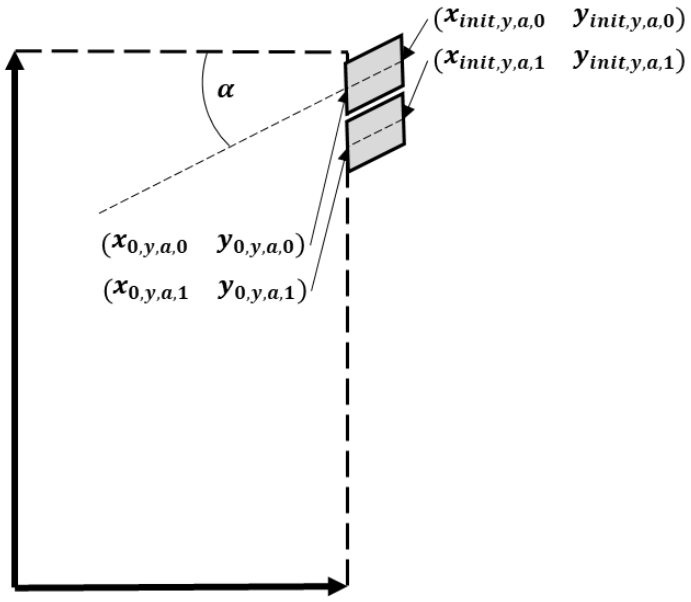

Figure 14.—Angle band initial positioning.

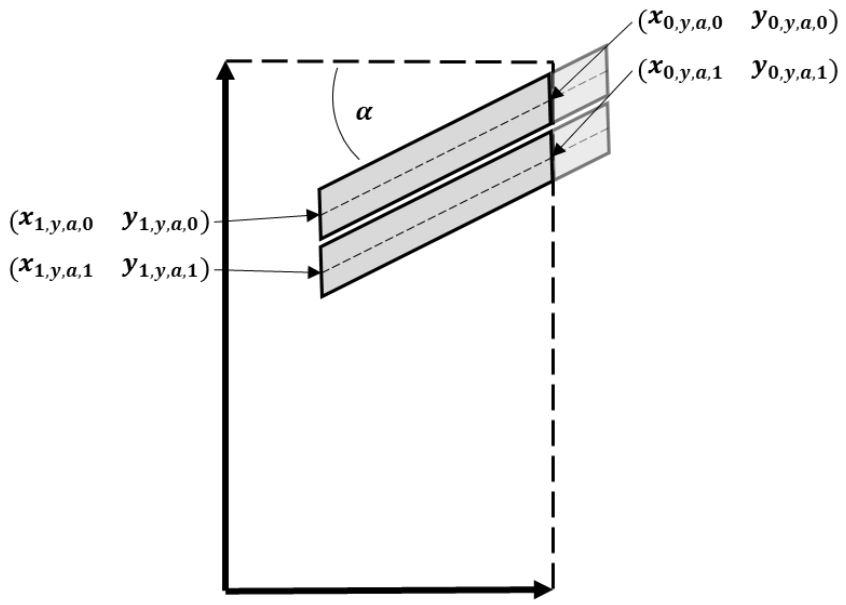

Figure 15.—Angled band end positioning.

The equations for the end positions of the angled bands are,

$$y_{1,b,a,i} = y_{0,b,a,i} - \left(x_{0,b,s,i} - x_{0,b,a,i}\right) \cdot \tan\alpha \tag{7}$$

$$x_{1,b,a,i} = x_{0,b,s,i} \tag{8}$$

Here the *y* coordinate at the end of the angled band, $y_{1,b,a,i}$, is chosen such that the center angled band maintains an angle α over a distance $\left(x_{0,b,s,i} - x_{0,b,a,i}\right)$, where $x_{0,b,s,i}$ is the *x* coordinate at the center of the i^{th} straight band. The center of the i^{th} straight band was chosen for convenience (see Figure 16). The band end position *y* coordinate, Equation (7), is later used to position the *y* coordinate of the straight yarn, $y_{1,s,y,i}$.

To position the end coordinates of each angled yarn, the 0^{th} position coordinates of the angled yarns are used in the following equations,

$$x_{0,y,a,i} = x_{0,b,a,i} \tag{9}$$

$$y_{1,y,a,i} = y_{0,y,a,i} - \text{abs}\left(x_{0,y,a,i} - x_{0,s,b,i} - \frac{b_w}{2} - \frac{3\,b_s}{2}\right)\tan\alpha \tag{10}$$

where the 0^{th} position *x* coordinate in an angled yarn, $x_{0,y,a,i}$, cooresponds with the 0^{th} position of the angled band, $x_{0,b,a,i}$, and the angled yarn end position *y* coordinate, $y_{1,y,a,i}$, is calculated such that an angle α is maintained over a distance $(x_{0,y,a,i} - x_{0,s,b,i} - \frac{b_w}{2} - \frac{3\,b_s}{2})$.

Now that the angled band and yarn positions have been determined, the straight bands and yarns positions can be established. Figure 16 shows the desired position of the straight bands and angled bands relative to each other.

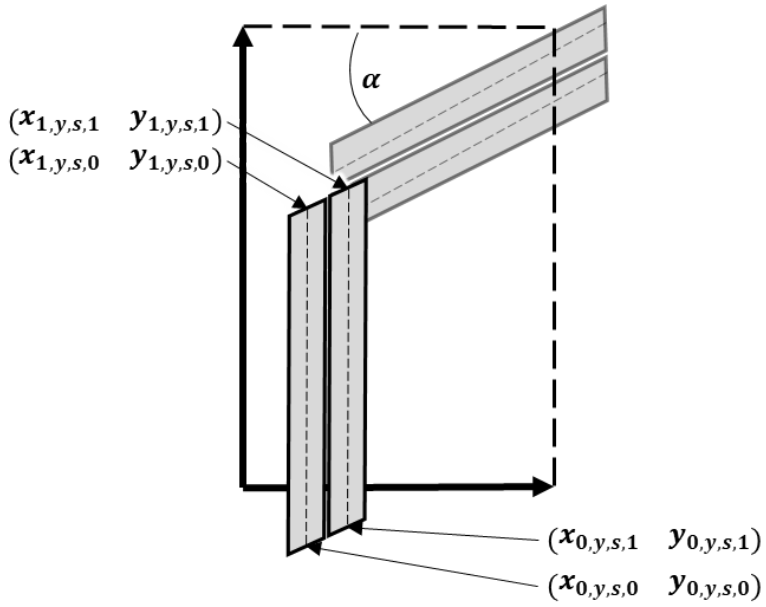

Figure 16.—Positioning of straight yarns.

Using the following equations, the straight yarn positions can be determined,

$$y_{1,y,s,i} = y_{1,b,a,i} - (y_c - 1) \cdot \left(\frac{y_w \tan\alpha}{2} + \frac{y_w}{2\cos\alpha} + \frac{y_s \tan\alpha}{2}\right) + i_y \cdot (y_w + y_s) \cdot \tan\alpha - \frac{y_c\,(y_w + b_s)}{2\cos\alpha} \tag{11}$$

$$x_{0,b,s,i} = (i_b + 0.5) \cdot b_w + i_b b_s \tag{12}$$

$$x_{0,y,s,i} = x_{0,b,s,i} + \left(i_y - \frac{(y_c - 1)}{2}\right) \cdot (y_w + y_s) \tag{13}$$

where the angled band end position *y* coordinate, $y_{1,y,s,i}$, is used for initial positioning. From this position, the second term and the fourth term on the right hand side of Equation (11) provide a constant offset to bring the center of the straight band into alignment with the center of the angled band. The third term on the right hand side of Equation (11) with the yarn index, i_y, adjusts the *y* position of each yarn in the i^{th} straight band such that they lie exactly along the centerline of the angled band, demonstrated in Figure 16. To establish the correct *x* coordinate for the straight bands, the bands are offset from the leftmost edge of the domain where $x = 0$. For the first band, $i_b = 0$, reducing the 0^{th} straight band *x* coordinate to an offset $\frac{b_w}{2}$. This positions the band such that the left edge of the first yarn in the first straight band – the *key straight yarn* – lies on the leftmost edge of the domain. Once the first band and its yarns are positioned, the subsequent bands will be laid at an appropriate distance to maintain the defined band spacing, b_s, between bands (see Figure 17).

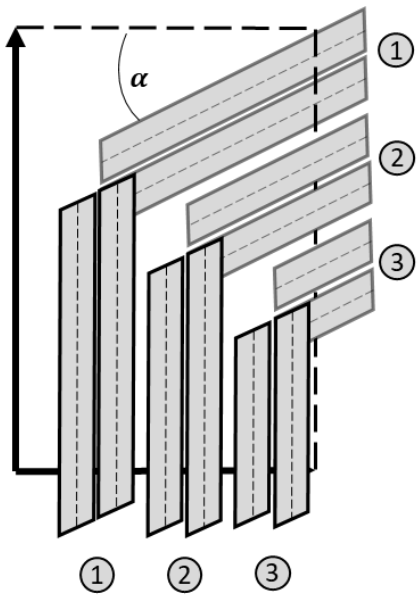

Figure 17.—Pattern generation continued.

Once the ends of each bands are in known positions *relative* to one another (i.e., the ends of the straight yarns are aligned along the centerlines of the angled bands and vice versa), the pattern can be easily generated in terms of band and yarn dimensions. For example, if one would like to advance the ends of the straight yarns in Figure 17 to the topmost edge of the angled bands, one simply adds half the bandwidth, $b_w/2$, to the current y positions. From this point, each straight band must undulate under an angled band and proceed straight to the top edge of the domain.

To ensure periodicity, the angled bands are added to the top and bottom of the domain (Figure 18). Here, the band that enters at the bottom right side of the domain (point 0) exits at the bottom center (point 1) and continues its trajectory from the top center (point 1) and exits at the top left corner (point 2). This pattern continues until the angled bands returns to the starting point (where point 5 maps to point 0). If one were to stack repeats of the RUC vertically, geometric continuity would also be preserved. This pattern is referred to as *simple periodicity* (see also Figure 21) since it emerges by the coupling of corresponding nodes lying directly opposite of one another on opposed faces of the domain (left face with right face, top face with bottom face). This type of periodicity does *not* actually emerge during the winding process. Rather, it is an approximation that enables use of standard, opposite face periodicity conditions. The impact of this approximation on the Abaqus predictions will be addressed in the Results section; NASMAT models currently use only the standard periodicity, since the staggered periodicity (discussed next) is not available at this time. Note that, if the bands and yarns were not all the same size, shape, and material, the simple periodicity conditions would result in discontinuities at the periodic boundaries, resulting in a poor approximation.

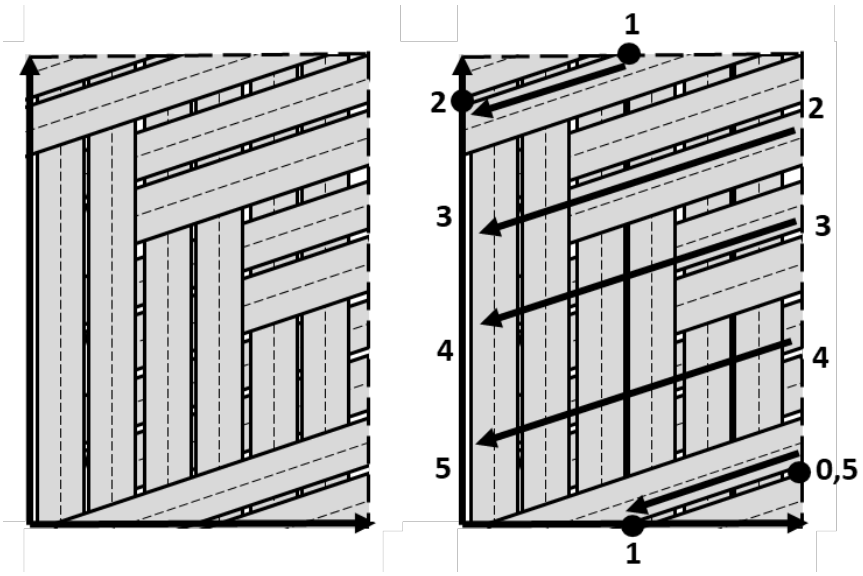

Figure 18.—Completed filament wound pattern with simple periodic bounds.

In contrast to the simple periodicity described above, the actual, staggered periodicity conditions are depicted in Figure 19. These conditions have been termed *real periodicity*. Here, the periodicity conditions are staggered and continuous only along the individual bands. To see this, one can trace one's finger along the arrows in Figure 19 from one point to the next. Starting at 0 and going to 1, one traverses from the bottom right to the bottom middle. Point 1 at the bottom continues to point 1 at the top of the domain, which is connected to point 2. From here, point 2 returns the band to the original starting point 0. This means that a load applied to this band travels only along this band (but of course transfers to adjacent bands via shear through the matrix). In comparison, the simple periodicity in Figure 18 connects each of the angled bands with each other, acting effectively as one band. As such, a load applied to one angled band is transferred through all angled bands in the domain, as if it is continuous. This can also be illustrated using set notation:

Simple Periodicity: $(0\ 1\ 2\ 3\ 4\ 5)$

Real Periodicity: $(0\ 1\ 2)(3\ 4)(5\ 6)(7\ 8)$

where each subset represents a continuous path of the angle bands. Simple periodicity represents a cyclic permutation across all nodes of the angled bands, again showing the connectivity across all angled bands in the domain. Real periodicity, on the other hand, permutes nodes only lying on the same band and hence decomposes the domain into a number of subsets equal to the actual number of bands in the RUC domain arising from the winding pattern. The consequences of simple periodicity and real periodicity will be addressed in Numerical Analysis of Filament Wound and Laminated RUCs Section.

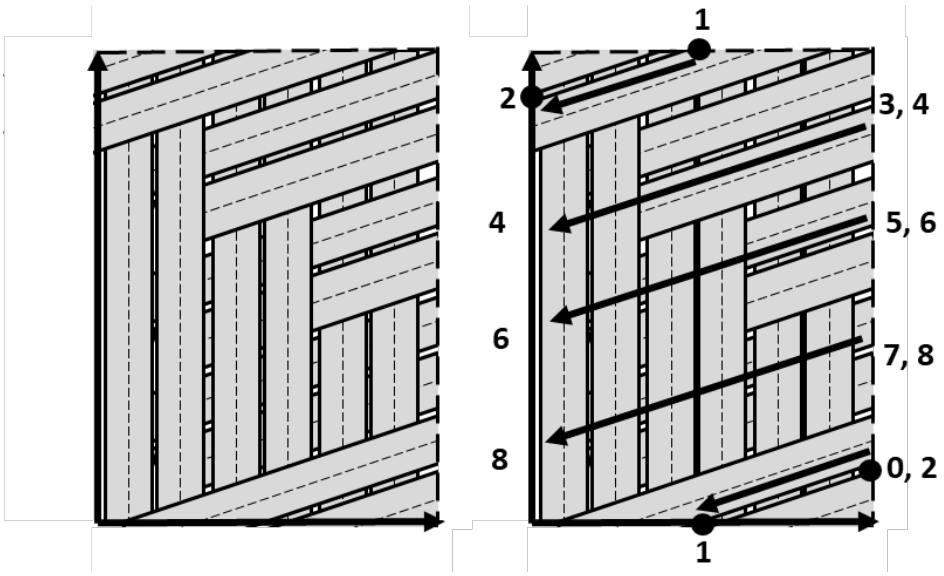

Figure 19.—Completed filament wound pattern with real periodic bounds.

Numerical Analysis of Filament Wound and Laminated RUCs

To assess the performance of the wound RUC, it is compared against a laminated RUC. Here, the laminated RUC represents the ideal case where no undulation is present (see Figure 9). The effect of the winding pattern is often neglected during modeling. To begin this section, there is a brief introduction to the 3D HFGMC homogenization theory used in NASMAT. Following this, a brief section on the periodicity conditions and elastic constants will come. Lastly, an analysis and discussion of the results generated in NASMAT and Abaqus will follow. Specifically, the elastic moduli and local elastic fields will be compared, not only in the tows, but also in the matrix. Furthermore, the two different types of periodicity conditions are investigated: simple periodicity and real periodicity. Both NASMAT and Abaqus can handle the simple periodic case. However, NASMAT currently cannot represent the real periodic conditions, thus a direct comparison between NASMAT and Abaqus is not possible for this case.

Homogenization Method Used in NASMAT

Homogenization theory is a technique whereby a medium is treated as homogeneous at a higher, macroscopic length scale even though the medium is heterogeneous at a lower, microscopic length scale (Suquet, 1987; Aboudi et al., 2013; Oller, 2014). Scale separation is assumed such that the macroscopic behavior will not be affected by the details of the microstructure, and the fields at the microscale can then be treated as perturbations of the macroscopic, global fields. Homogenization theory is very useful for the design and analysis of composite materials and structures as it enables effective, homogenized properties to be calculated, and these can then be used at the global scale as if the material is a standard, homogeneous continuum. Of course, the key assumption of scale separation is never fully valid and thus should be thought of as an engineering approximation that is good in some situations and very rough in others. For example, a standard carbon/epoxy composite tow contains thousands of individual carbon filaments barely visible to the naked eye. In contrast, the weave/braid pattern in a textile composite is on the mm length scale and easily observed in composite parts. Yet both of these heterogeneities are

commonly treated via homogenization theory. Proximity to boundaries and discontinuities, damage and inelasticity, and myriad other factors affect the validity of homogenization theory's scale separation assumption. It remains, however, a very useful engineering approximation, particularly for composite materials.

The conjugate to homogenization is localization (also sometimes referred to as "dehomogenization"). Homogenization provides effective properties at a higher scale based on lower length scale constituent properties and their arrangement. Localization determines the local fields at a lower length scale based on known macroscale fields. For example, known applied strain components on a composite test coupon can be localized to determine the local strains in the fiber and matrix constituents. Many micromechanics theories/methods are capable of both homogenization and localization.

A number of leading micromechanics homogenization/localization theories have been implemented in the NASA Multiscale Analysis Tool (NASMAT) software, which is maintained and released by NASA (NASA, 2024). These theories include Mori-Tanaka, the Generalized Method of Cells (GMC), and the High-Fidelity Generalized Method of Cells (HFGMC) (Aboudi et al., 2021). Unique to NASMAT is the ability for the micromechanics models to call each other or themselves recursively to capture microstructural geometries at any number of length scales. In addition, multiple nonlinear damage and viscoplastic constitutive models are available for the constituent materials, and NASMAT can link with Abaqus and other finite element codes to enable micromechanics analysis at the integration points in a structural finite element model (Pineda et al., 2021). Multiscale modeling of other physics, such as thermal/electrical conductivity, diffusion, and magnetic permeability (Bednarcyk et al., 2017) can also be conducted with NASMAT.

Herein, the HFGMC micromechanics theory within NASMAT has been applied to predict the effective properties of wound composites. HFGMC considers the composite material to be periodic and analyzes a repeating unit cell (RUC) composed of an arbitrary number of parallelepiped sub-volumes (called subcells). This geometric representation is shown in Figure 20. The derivation of the HFGMC theory is given by Aboudi et al. (2013, 2021). To briefly summarize, it is based on ensuring continuity of surface-averaged tractions and displacements between interior subcells and similar periodicity conditions at the RUC boundaries. Since HFGMC makes use of a quadratic displacement field, it provides good approximations of the composite local fields and effective properties but can be computationally demanding as there are many unknown variables (and thus a large system of equations to solve) compared to lower-fidelity theories (such as GMC).

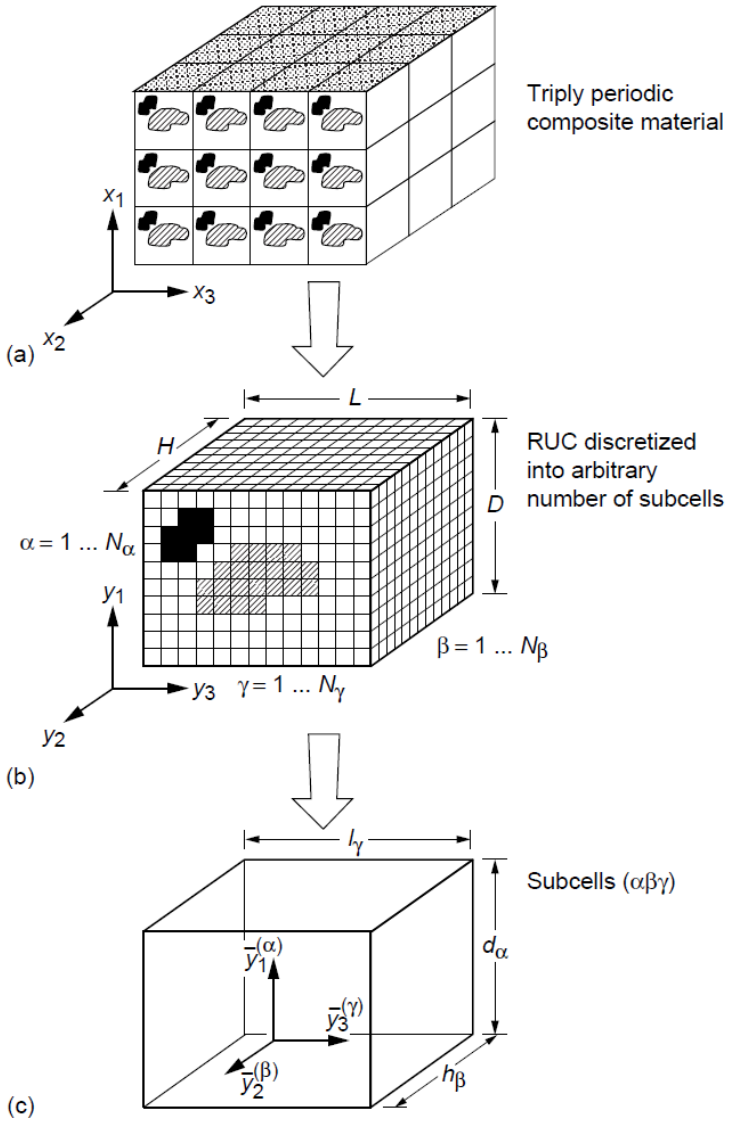

Figure 20.—(a) A multiphase composite with triply-periodic microstructures defined with respect to global coordinates (x_1, x_2, x_3). (b) The repeating unit cell (RUC) is represented with respect to local coordinates (y_1, y_2, y_3). It is divided into N_α by N_β by N_γ subcells, in the y_1, y_2, and y_3 directions, respectively. (c) A characteristic subcell ($\alpha\beta\gamma$) with local coordinates $(\bar{y}_1^{(\alpha)}, \bar{y}_2^{(\beta)}, \bar{y}_3^{(\gamma)})$ whose origin is located at its center.

Periodicity Conditioned Prescribed in NASMAT and Abaqus

Once the textile pattern is generated in TexGen, it is possible to export a voxel file to Abaqus. Here, the domain is discretized in the x, y, and z directions by fixed user inputs, and periodic boundary conditions (PBCs) are automatically applied.

With Abaqus, it was possible to assess both the simple periodic case (Figure 21) and the real periodic case (Figure 22). As previously discussed, simple periodicity does not exactly reflect the conditions in a wound RUC due to an inappropriately imposed continuity across all angled yarns in the domain.

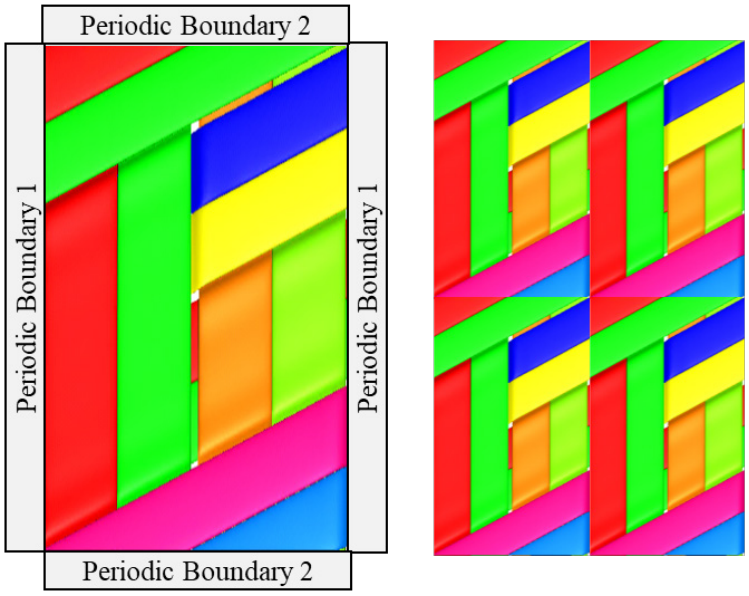

Figure 21.—Simplified periodic boundary conditions (left) and resultant pattern (right).

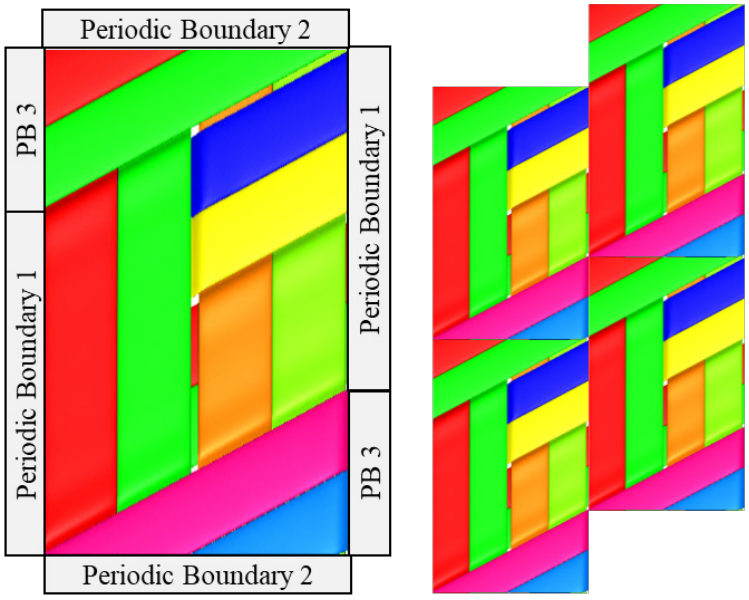

Figure 22.—Real periodic boundary conditions (left) and resultant pattern (right).

As a means of comparison, the elastic constants from each RUC were calculated using a linear perturbation step in Abaqus, whereby a unit load is applied at a master node to which the PBCs are coupled. Upon application of each unit load, the RUC experiences a generalized displacement. To extract the displacement from each loading frame in the Abaqus ODB, the following python script was used:

```
import numpy as np
odb = session.odbs['filename']
S = np.zeros([6,6])
```

```
for i in range(6):
  for j in range(6):
   S[i,j]+=odb.steps['Isothermallinearperturbationstep'].frames[i+1].field\
Outputs['U'].values[j].data[0]
```

where S is the compliance matrix. Thus, for each load type (3 normal loads and 3 shear loads) the subsequent displacements (3 normal displacements and 3 transverse displacements) were extracted. The boundary conditions were formulated such that the displacements are equal to the compliance matrix. This returned a 6 x 6 compliance matrix with entries generally non-zero due to anisotropy. From the compliance matrix, the engineering constants could be calculated using the following equations:

$$E_{11} = \frac{1}{s_{11}} \qquad E_{22} = \frac{1}{s_{22}} \qquad G_{12} = \frac{1}{s_{66}} \qquad v_{12} = -\frac{s_{12}}{s_{11}}$$

Additionally, the yarn volume fraction for each model was determined using the length of the set containing all elements and the length of the set containing just matrix elements:

```
odb = session.odbs['filename']
part = odb.rootAssembly.instances['PART-1-1']
ALLELS = len(part.elementSets['ALLELEMENTS'].elements)
MATELS = len(part.elementSets['MATRIX'].elements)
vf = (ALLELS - MATELS) / ALLELS
```

Since a voxelated geometry was used, all elements (C3D8R) have the exact same dimensions, allowing for a straightforward yarn volume fraction calculation. The yarn volume fraction showed a slight dependence on voxel size, primarily with a coarse mesh. However, using finer meshes (smaller voxel sizes) allowed the yarn volume fraction to converge to a single value (Figure 23). Table 1 and Table 2 contain the properties assigned to the tow and the matrix, respectively.

Figure 23 shows the volume fraction of the tows within the composite RUC as a function of the number of voxels in the RUC. The Abaqus and NASMAT RUCs are identical, thus the volume fractions are the same for both the simplified periodicity and laminate cases. The goal was to keep the tow volume fraction as close as possible for all models and levels of discretization, but this is not possible as the discretization changes. Figure 23 shows that, for a 6000 (6k) voxelization, there is a significant difference between the periodic models and the laminate model, but for all four more refined RUCs, the discrepancy is small. There is still, however, some discrepancy in the tow volume fractions as a function of RUC voxel refinement, with this discrepancy decreasing as the refinement is increased (as expected). In addition to the Abaqus and NASMAT voxelated models, predictions have made using classical lamination theory (CLT) with a tow volume fraction of 58.9%. This value is plotted in Figure 23 for comparison with the voxelated RUC models, although this is a single prediction (not a function of number of voxels) based on a $[90/30/–30/30]_s$ laminate. The MATLAB code provided by Aboudi et al. (2021) was used for the CLT calculations wherein the ply properties were determined from an HFGMC RUC consisting of the tow material and matrix material, as depicted in Figure 24.

TABLE 1.—CONSTITUTIVE MATERIAL PROPERTIES OF TOWPREG WITH 65% FIBER VOLUME FRACTION (T700/EPOXY)

E_{11}, MPa	E_{22}, MPa	E_{33}, MPa	ν_{12}, -	ν_{13}, -	ν_{23}, -	G_{12}, MPa	G_{13}, MPa	G_{23}, MPa
143970	7450	7450	0.27	0.0676	0.0676	5600	3700	3700

TABLE 2.—CONSTITUTIVE MATERIAL PROPERTIES OF EPOXY MATRIX

E, MPa	ν, -
3300	0.35

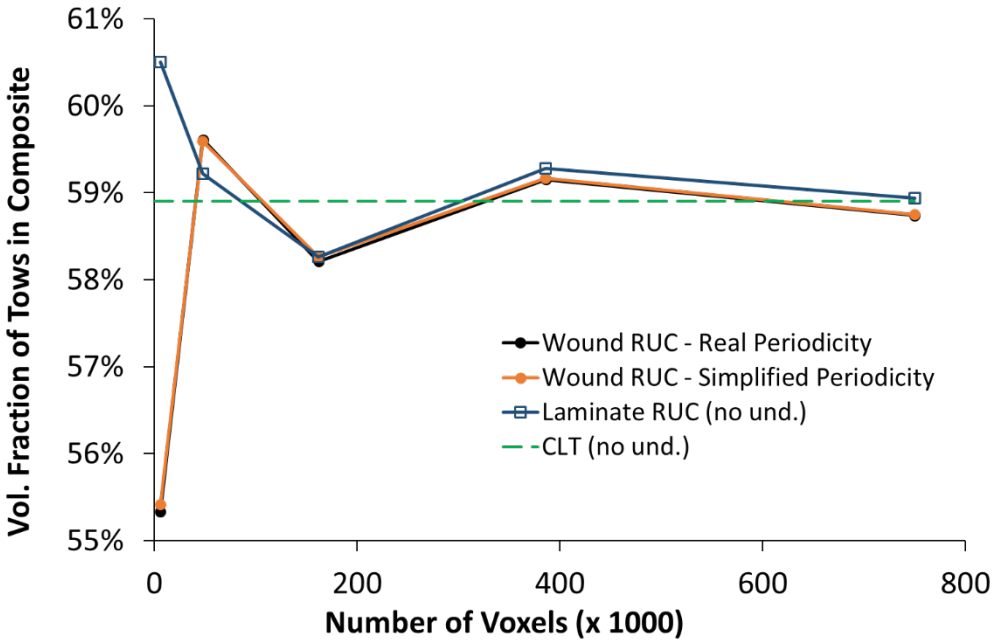

Figure 23.—Volume fraction of the tows within the composite for each of the models as a function of RUC voxel refinement. For the simplified periodicity and laminate cases, the Abaqus model and the NASMAT model have identical tow volume fractions. The classical lamination theory (CLT) prediction is not based on voxelization; a single value is plotted for comparison.

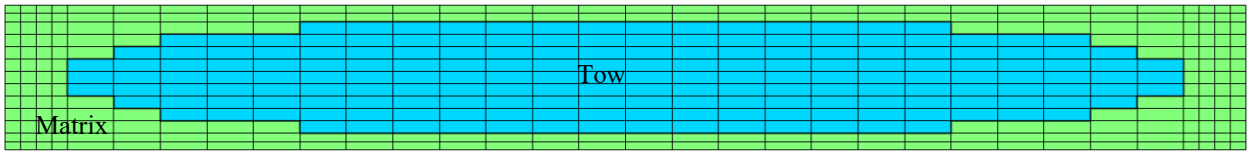

Figure 24.—HFGMC RUC used to calculate the ply properties for use in the CLT prediction of the composite properties.

Results and Discussion

Predictions of Effective Properties

In order to compare the different numerical methods and periodicity conditions, multiple models were created that were both analyzed in Abaqus or NASMAT (Table 3). Here, one can see that there are 5 different models, according to their periodicity conditions. Additionally, a CLT model was created and compared against those in NASMAT and Abaqus, but it is not listed in Table 3. As discussed above, five different refinement levels, ranging from 6000 (6k) voxels to 750,000 (750k) voxels, were created and used to determine when the elastic moduli converged. The criterion for convergence was set to 5%, whereby the elastic constants at the higher refinement level was compared with the ones below. Not only must the difference in predicted moduli be below 5%, but it must be below this value for at least two consecutive refinement levels to be considered converged. In this case, the 2 highest refinement levels (384k, 750k) all showed converged behavior, having changes in elastic moduli below 5%.

TABLE 3.—MODELS ORGANIZED ACCORDING TO THEIR DIFFERENT PERIODICITY CONDITIONS

Model	Simplified Periodicity	Real Periodicity
NASMAT Wound RUC	x	
Abaqus Wound RUC	x	x
NASMAT Laminated RUC	x	
Abaqus Wound RUC	x	

For the Abaqus RUC models, the wound RUC was simulated with both simplified periodicity and real periodicity while the laminated RUC required only simplified periodicity; since the laminated RUC has no undulations, simplified periodicity and real periodicity give identical results. For the NASMAT RUCs, only simplified periodicity conditions could be used since there is no way, currently, to stagger the periodicity conditions.

Figure 25 compares the predicted in-plane Young's moduli of the composite. In all cases, greater E_{11} is predicted (compared to E_{22}) because the RUC band pattern (and CLT layup) is 75% ±30° and only 25% 90° (see Figure 25). Within the Abaqus results (solid lines), it is clear that the presence of undulation has a noticeable effect. For example, in the case of the 750k voxel RUC, neglecting the undulation results in an E_{11} and E_{22} that are 6.2 and 6.7% higher (respectively) than the prediction with real periodicity. The differences in the Abaqus results for real vs. simplified periodicity are relatively small, with E_{11} and E_{22} exhibiting only a 1.1 and 2.5% difference (respectively) for the 750k voxel case. Although this difference is small, it may become significant if damage occurs. Due to the different periodicity conditions, the loads are carried differently in the domain of both RUCs, affecting how damage progresses within the domain (see RUC Generation Program Explanation, specifically the section concerning the periodicity conditions).

The NASMAT results for the laminate cases (with no undulation) in Figure 25 agree very well with the corresponding Abaqus results. NASMAT's E_{11} predictions are within 2.5% for discretization levels at or above 162k, while for E_{22}, the NASMAT and Abaqus predictions are nearly identical when neglecting the undulation. For the simple periodicity case, which includes the undulations, the NASMAT E_{11} predictions are 1.5 to 2.9% lower than Abaqus for discretization levels at or above 162k with the discrepancy decreasing for increasing refinement. The NASMAT to Abaqus agreement is even better in the E_{22} predictions, with NASMAT predicting 0.8 to 2.1% lower values (again, at or above the or above 162k discretization). The main difference between finite element models, like Abaqus, and the HFGMC model within NASMAT is the method used to enforce continuity and periodicity. Finite element models do this in point-wise manner at the nodes, whereas HFGMC does this via surface averages over the sub-volume faces. The results in Figure 25 indicate that the impact of this difference decreases as each method converges based on its discretization. Figure 25 indicates that a refinement of 162k voxels is reasonably converged compared to the results for the more refined models. A slight anomaly is present in the data is 386k E_{11} prediction by Abaqus with the real periodicity, which is a bit higher than expected.

As mentioned, a prediction of the in-plane moduli was also made using CLT, wherein the ply properties were determined via an HFGMC model whose RUC is shown in Figure 24. This RUC provides a close representation of the tow architecture used in the more refined Abaqus and NASMAT models, and the RUC reflects the tow volume fraction of the more refined models quite well (see Figure 23). The Abaqus and NASMAT laminate predictions (neglecting the undulations) are generally within 3 to 4% of the CLT results, which is evidence that the models are implemented correctly.

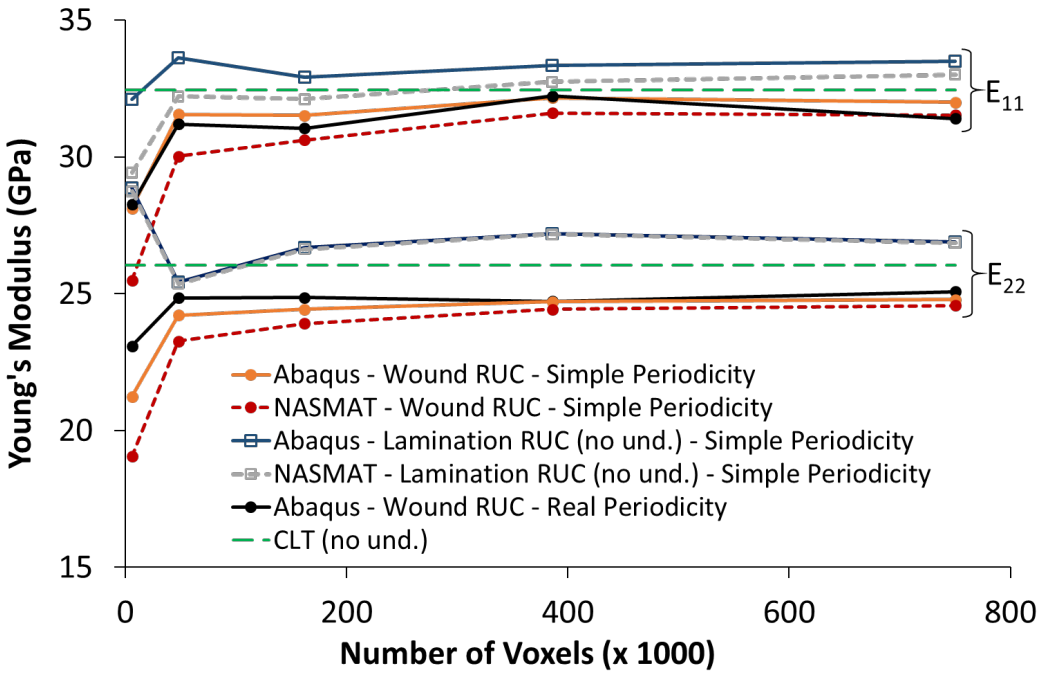

Figure 25.—Predicted effective in-plane Young's Moduli, E_{11} and E_{22}, of the composite as a function of RUC voxel refinement. The classical lamination theory (CLT) prediction is not based on voxelization; a single value is plotted for comparison.

Figure 26 presents the in-plane shear modulus, G_{12}, predictions of the Abaqus, NASMAT, and CLT models. The trends and differences among the predictions are similar to those observed in the E_{11} predictions in Figure 25, with the exception of the effect of RUC discretization. The number of voxels in the RUC appears to have a greater impact on the G_{12} predictions compared to both the E_{11} and E_{22} predictions, with a greater discrepancy between the coarser models and their more refined counterparts.

Figure 27 compares the predicted in-plane Poisson's ratio, ν_{12}, among the models. The Abaqus and NASMAT models without undulation predict a lower ν_{12} compared to the cases with undulation. Interestingly, the CLT prediction is somewhat higher compared to the Abaqus and NASMAT predictions, and it actually is closer to the Abaqus and NASMAT predictions that include undulation. This is likely coincidental as there is often significant variability in micromechanics prediction of matrix dominated Poisson ratio's (c.f., Aboudi et al., 2021), as they are based on small induced transverse strains. A key distinction of the CLT model is that the yarns and matrix are homogenized to obtain effective ply properties, whereas the Abaqus and NASMAT models retain separate yarn and matrix phases at the RUC scale. Figure 25 to Figure 27 indicate that this difference impacts the effective ν_{12} prediction to a greater extent compared to the moduli predictions.

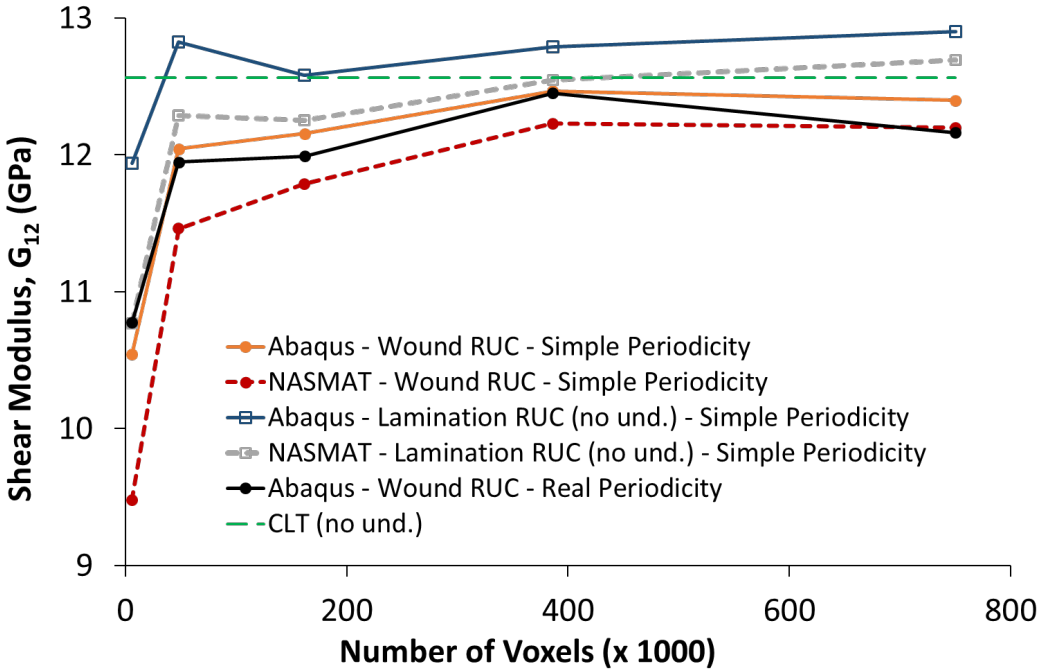

Figure 26.—Predicted effective in-plane shear modulus, G_{12}, of the composite as a function of RUC voxel refinement. The classical lamination theory (CLT) prediction is not based on voxelization; a single value is plotted for comparison.

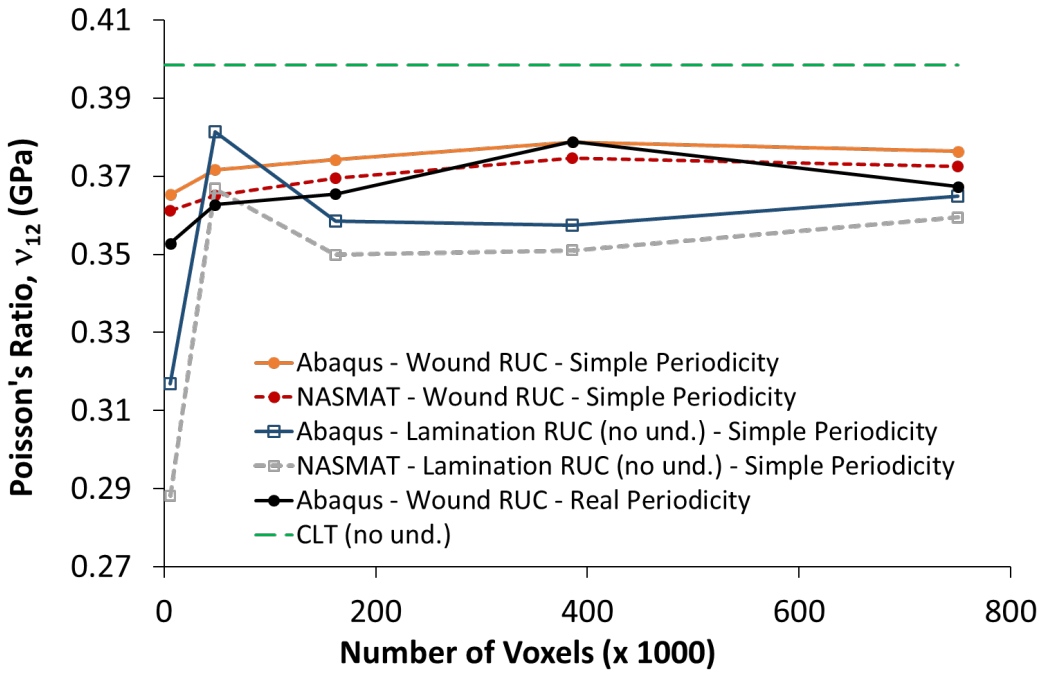

Figure 27.—Predicted effective in-plane Poisson's ratio, σ_{12}, of the composite as a function of RUC voxel refinement. The classical lamination theory (CLT) prediction is not based on voxelization; a single value is plotted for comparison.

Predictions of Local Stress Fields

Figure 28 to Figure 30 show the 750k voxel Abaqus predictions for local stress fields in the yarns, with the matrix between the yarns removed. These figures show the local in-plane normal and shear (σ_{11}, σ_{22}, and σ_{12}) fields, respectively, in response to a corresponding applied unit global stress component (with all other global stress components kept at 0). Three Abaqus cases are compared: real periodicity, simple periodicity, and laminate representation (with no yarn undulation). The results are generally quite similar, with the differences between real and simple periodicity confined to the undulation regions. The laminate representation, which has no yarn undulation, appears to approximate the yarn stresses away from the undulations quite well.

Similarly, Figure 31 and Figure 32 show the 750k voxel Abaqus predictions for the in-plane normal local stress fields in the matrix, with the yarns removed. Again, each local component shown in each figure is in response to an applied corresponding unit global stress component. Stress concentrations arising between the yarns are clear on the top surface of the RUC. The real and simple periodicity predictions are again quite similar, with the real periodicity exhibiting some greater stress concentrations for the σ_{11} field (c.f., Figure 31 top edge on faces normal to the x_1-direction). In the σ_{22} field (Figure 32), it appears that the simple periodicity predicts higher concentrations at some points. The matrix stresses for the laminate representation of the composite are quite different from the models that include undulation. There are still stress concentrations between the yarns, but the concentrations in the undulation regions are (obviously) absent. Note that the matrix stress magnitudes are much lower than the yarn stress magnitudes (due to the lower stiffness of the matrix, see Table 1 and Table 2). Given that the effective properties are directly related to the volume averages of the stress components, the differences observed in the matrix stress distributions contribute less to the composite effective properties than do the higher stresses in the yarns. This is one reason the effective properties predicted by the laminate representation are in reasonably good agreement with the other models despite the significant discrepancies in the local matrix stress fields. The more approximate nature of the laminate representation matrix stress fields also suggests that, if progressive damage were considered in the models, the laminate predictions would also be more approximate compared to the models that include the undulation.

Figure 33 to Figure 37 compare the in-plane local stress field predictions of NASMAT with Abaqus for the 750k voxel model with simple periodicity. Again, each local stress component shown is in response to the corresponding applied global stress component, and the stress fields in the yarns and the stress fields in the matrix are shown separately. The NASMAT local stress field predictions are in good agreement with the Abaqus predictions, with NASMAT appearing to predict slightly higher stress concentrations in the tows and Abaqus predicting slightly higher concentrations in the matrix. The good correspondence between NASMAT and Abaqus in terms of the local field predictions corresponds with the good agreement observed in their predicted effective properties.

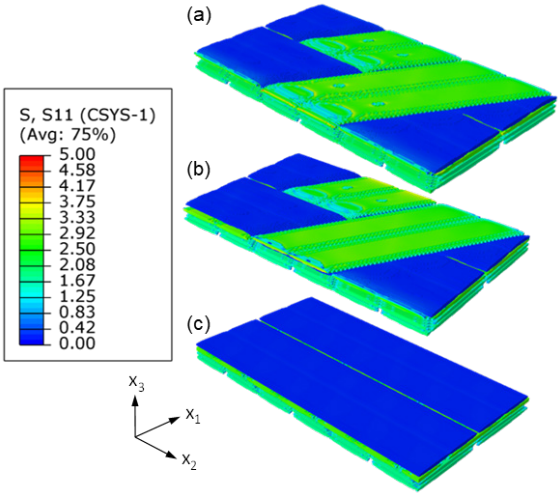

Figure 28.—Abaqus σ_{11} local stress field predictions (in MPa) for the yarns when the RUC is subjected to a global unit σ_{11} stress (with all other global stress components equal to zero). (a) Real periodicity. (b) Simple periodicity. (c) Laminate (no undulation)

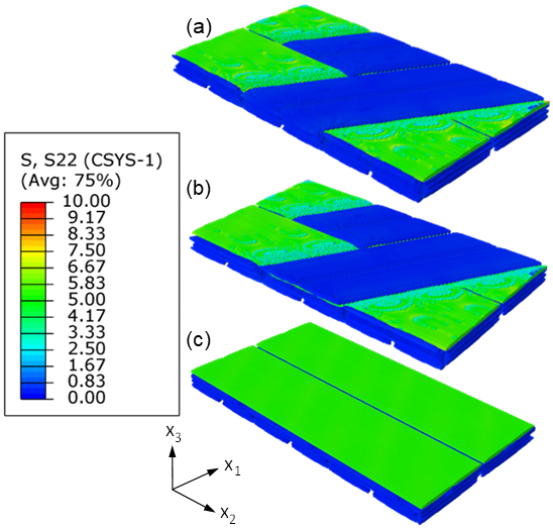

Figure 29.—Abaqus σ_{22} local stress field predictions (in MPa) for the yarns when the RUC is subjected to a global unit σ_{22} stress (with all other global stress components equal to zero). (a) Real periodicity. (b) Simple periodicity. (c) Laminate (no undulation).

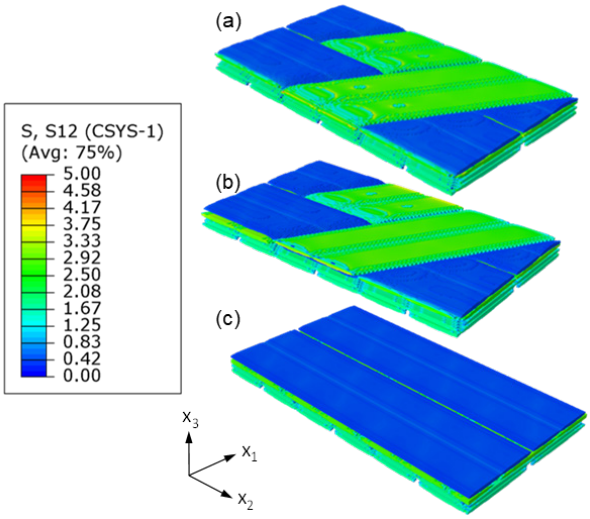

Figure 30.—Abaqus σ_{12} local in-plane shear stress field predictions (in MPa) for the yarns when the RUC is subjected to a global unit σ_{12} in-plane shear stress (with all other global stress components equal to zero). (a) Real periodicity. (b) Simple periodicity. (c) Laminate (no undulation).

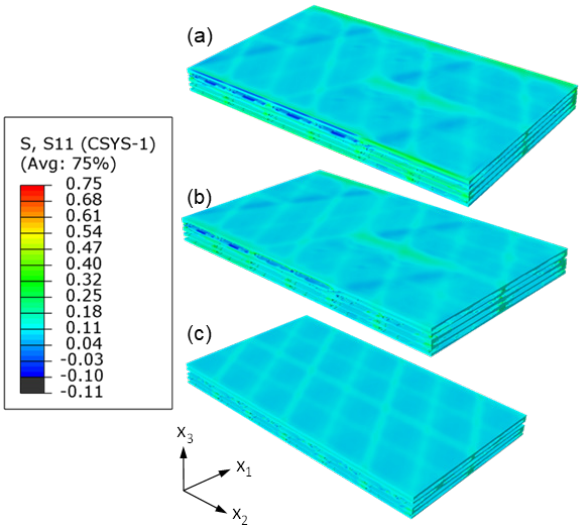

Figure 31.—Abaqus σ_{11} local in-plane shear stress field predictions (in MPa) for the matrix when the RUC is subjected to a global unit σ_{11} stress (with all other global stress components equal to zero). (a) Real periodicity. (b) Simple periodicity. (c) Laminate (no undulation).

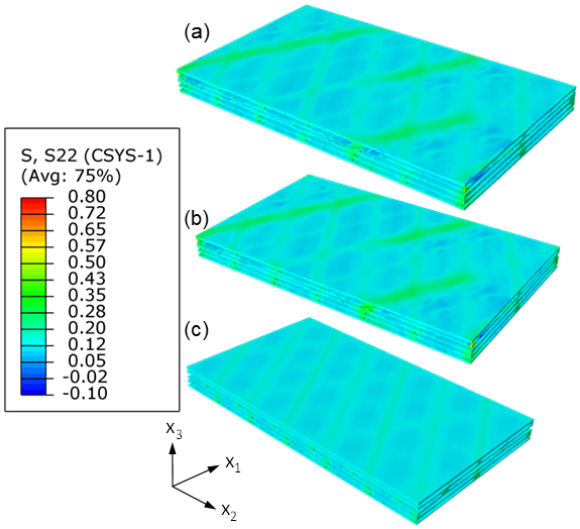

Figure 32.—Abaqus σ_{22} local stress field predictions (in MPa) for the matrix when the RUC is subjected to a global unit σ_{22} stress (with all other global stress components equal to zero). (a) Real periodicity. (b) Simple periodicity. (c) Laminate (no undulation).

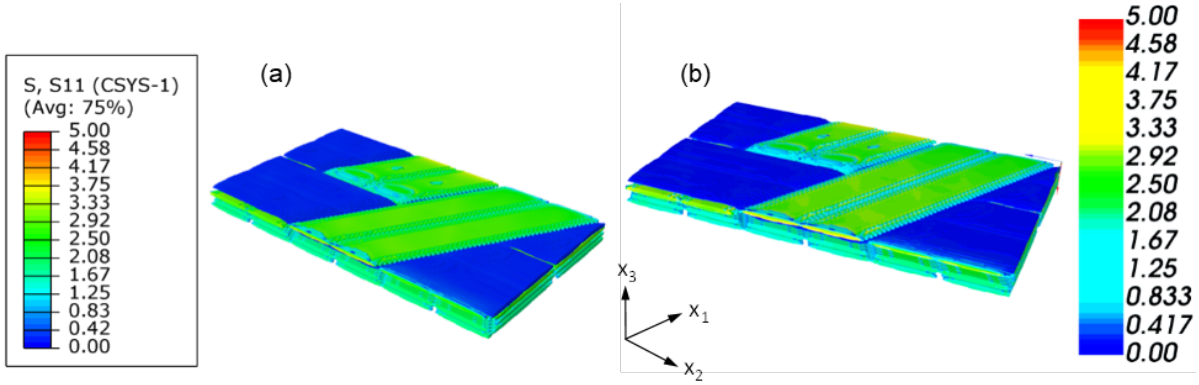

Figure 33.—Comparison of (a) Abaqus and (b) NASMAT σ_{11} local stress field predictions (in MPa) for the yarns when the RUC is subjected to a global unit σ_{11} stress (with all other global stress components equal to zero).

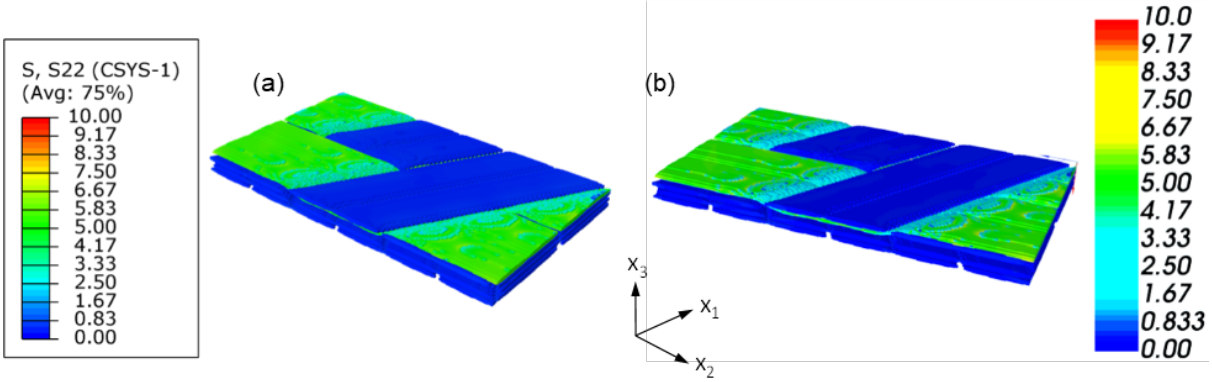

Figure 34.—Comparison of (a) Abaqus and (b) NASMAT σ_{22} local stress field predictions (in MPa) for the yarns when the RUC is subjected to a global unit σ_{22} stress (with all other global stress components equal to zero).

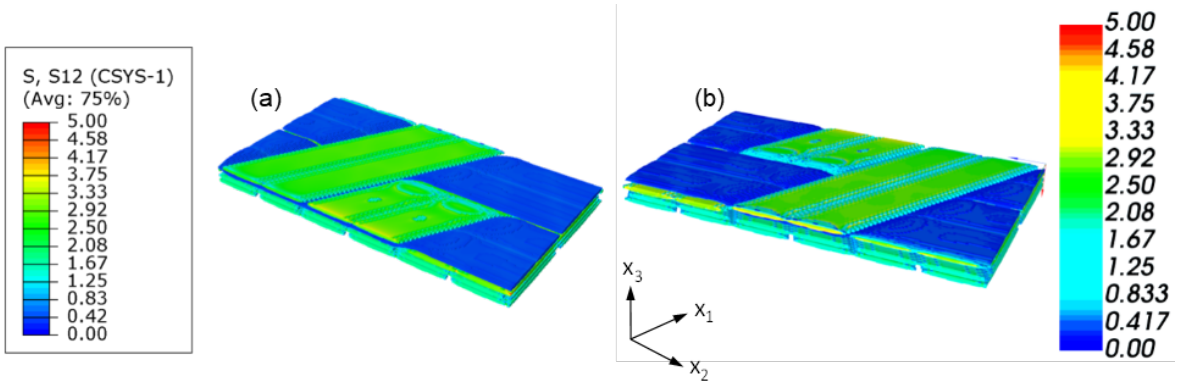

Figure 35.—Comparison of (a) Abaqus and (b) NASMAT σ_{12} local in-plane shear stress field predictions (in MPa) for the yarns when the RUC is subjected to a global unit σ_{12} in-plane shear stress (with all other global stress components equal to zero).

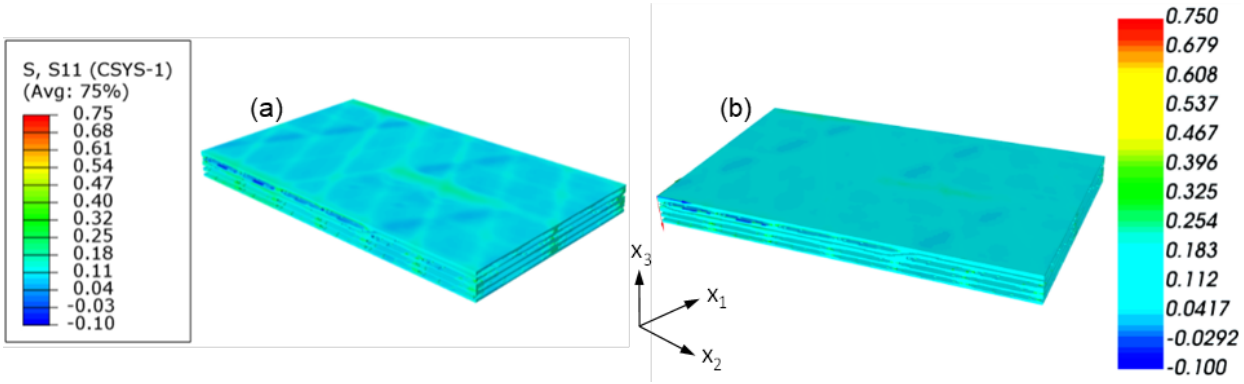

Figure 36.—Comparison of (a) Abaqus and (b) NASMAT σ_{11} local stress field predictions (in MPa) for the matrix when the RUC is subjected to a global unit σ_{11} stress (with all other global stress components equal to zero).

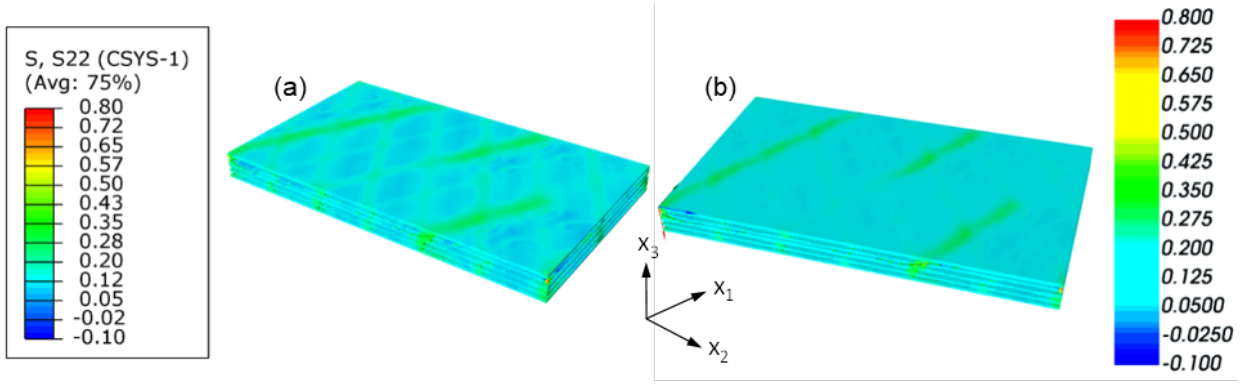

Figure 37.—Comparison of (a) Abaqus and (b) NASMAT σ_{22} local stress field predictions (in MPa) for the matrix when the RUC is subjected to a global unit σ_{22} stress (with all other global stress components equal to zero).

Conclusion

The subject of this paper was the elastic analysis of RUCs representing a wound composite represented as a laminate compared to a more accurate representation accounting for undulations of the yarns. In particular, a script (given in the Appendix) for generating wound RUCs was presented and a study was undertaken to examine, evaluate, and compare quantitatively the following aspects of modeling wound composites: (1) the effect of undulations, (2) the effect of model type (Abaqus vs. NASMAT), and (3) the effect of simplified vs. more realistic PBCs.

Initially, the RUCs were compared against each other using their effective engineering constants. To this end, a study was undertaken to determine the geometric refinement (voxel count) at which the engineering constants would converge as well as the corresponding tow volume fraction at each voxel count. Considering five refinements, ranging from 6,000 (6k) to 750,000 (750k) voxels, it was shown that at a voxel count of 384k the models converged for all elastic moduli. The RUC with the highest values of E_{11}, E_{22}, and G_{12} was the laminated RUC in both Abaqus and NASMAT, while the RUC with real PBCs had very similar engineering constants to the RUC with simple PBCs in Abaqus. The RUC with the lowest values of E_{11}, E_{22}, and G_{12} was the wound RUC with simple PBCs in NASMAT. For verification, the results from Abaqus and NASMAT were also compared against effective engineering constants calculated using CLT (with effective ply properties determined using HFGMC). CLT predicted values of E_{11}, E_{22}, and G_{12} that landed in the middle of the range of other results, however, CLT predicted a higher value of Poisson's ratio than the other methods.

In addition, predicted elastic stress fields were compared for the 750k voxel models. Specifically, the laminated RUC, the wound RUC with simple PBCs, and the wound RUC with real PBCs simulated in Abaqus were compared, while the NASMAT and Abaqus predictions were compared for the wound RUC with simple PBCs. The Abaqus real and simple PBC local stress field predictions were in good agreement, as were those of NASMAT and Abaqus. The biggest observed discrepancy in the local stress fields was in the matrix between the laminate representation and the representations that included the undulations. Compared to the laminate RUC, the wound RUC undulation regions have stress levels 1.5 to 2.0 times greater in magnitude for both σ_{11} (Figure 31) and σ_{22} (Figure 32). This would be expected to affect nonlinear model predictions (e.g., progressive damage), but the effective properties presented herein are only minorly affected because of the lower stiffness of the matrix compared to the yarns.

Based on this study, the following conclusions can be made:

1. The effective engineering constants of a laminated RUC will generally be somewhat higher than those of a wound RUC, regardless of the boundary conditions.
2. Simulating the wound RUC with both simple PBCs and real PBCs revealed that the differences in effective engineering constants are quite small. Additionally, the elastic stress fields between the two models were very similar. Thus, using simple PBCs instead of the real PBCs to simulate a wound composite should yield a good approximation of the effective engineering constants and local elastic stress fields. This simplification may be less effective in the presence of nonlinearities such as local constituent damage.
3. Additionally, the predicted effective engineering constants and the local elastic stress fields between both Abaqus and NASMAT were in good agreement (in terms of both stress field magnitude and distribution), even when comparing real and standard periodicity against each other. However, the difference between both programs becomes apparent when comparing run times for the 750k voxel models. In Abaqus it was possible to leverage both the GPU (Nvidia RTX 4080 16Gb) and CPU (AMD Threadripper Pro 5995X 64C/128T) during computation while NASMAT could only access

the CPU. For the sake of brevity, the solver times for the wound RUCs will only be compared, since the same trend also holds true for the laminated RUCs. In Abaqus, using C3D8R elements, the total solver time using 1x logical processor (thread) and 1x GPU was 2257s and 1297s for real periodicity and standard periodicity, respectively. Additionally, the solver time for the same models in Abaqus using 64x logical processors (threads) for real periodicity was 706s and for standard periodicity it was 301s. In NASMAT, the total solver time using 3D HFGMC for standard periodicity using 64x logical processors (threads) was 3362s, a more than 10x difference in runtime when compared against Abaqus. Based on the homogenization theory used in NASMAT, the runtimes can change drastically. Multi-step homogenization can also be used to reduce the size of the global problem, leading to improved run times.

4. CLT provides a prediction of the effective engineering constants that fits the middle of the data; however, the prediction of Poisson's ratio was highest of all methods, thus appearing to be overpredicted.
5. Based on the elastic stress fields in the matrix, it seems that the highest stressed regions occur near points of undulation, indicating that damage would likely initiate in these regions. When compared to the laminate RUC, the wound RUC undulation regions have stress levels 1.5 to 2.0 times greater in magnitude for both σ_{11} (Figure 31) and σ_{22} (Figure 32). These are neglected by the laminate representation, and they also represent the regions of the largest discrepancies among the models.
6. All analyses in Abaqus leveraged the GPU (Nvidia RTX 4080 16Gb) during computation while the NASMAT models used strictly the CPU (AMD Threadripper Pro 5995X 64C/128T).

Furthermore, the study conducted herein will be extended to the following topics:

1. Thermal Loading: Mechanical loading in the elastic regime has been thoroughly investigated in the manuscript, particularly the resulting in plane elastic stress fields. A simple extension of this model to include thermal loading in the elastic regime can be done, wherein a unit uniform temperature change would be applied, resulting in local stress fields due to the mismatch in properties between the yarns and the matrix.
2. Multistep Homogenization: In the present study, the one step 3D HFGMC homogenization technique was implemented, effectively modeling a RUC at a single length scale. However, it is possible to perform a multistep homogenization, reducing the size of the global problem (in terms of systems of equations), leveraging NASMAT's recursive modeling ability. In this case, a RUC consisting of fiber and matrix constituents is homogenized and represents effective yarn properties. Next, these yarn unit cells are stacked / oriented appropriately to represent locally the layup and are subsequently homogenized. Finally, these stacks are placed accordingly as to reconstruct the desired global RUC and homogenized using 2D homogenization at the global scale, smearing the properties effectively into a single plane at the highest length scale.

Appendix—RUC Generation Code

```
from TexGen.Core import *
import math

##############################################################################
###
#-- Classes
##############################################################################
###
class BandDefinition:
  """Class for defining yarn characteristics"""

  def __init__(self, yarn_width: float, yarn_thickness: float, yarn_count:
float, yarn_spacing: float = 0, shape_factor: float = 1, shape: str =
"PowerEllipse") -> None:
    # -- Exceptions
    if yarn_width <= 0:
      raise ValueError("Yarn width must be finite and positive")
    if yarn_thickness <= 0:
      raise ValueError("Yarn thickness must be finite and positive")
    if yarn_count == 1:
      print("Yarn count is equal to 1, yarn spacing has been set to 0")
      yarn_spacing = 0

    # -- Band Attributes
    self.yarn_width = yarn_width
    self.yarn_thickness = yarn_thickness
    self.yarn_count = yarn_count
    self.yarn_spacing = yarn_spacing
    self.shape = shape
    self.shape_factor = shape_factor
    self.band_width = self.yarn_width * self.yarn_count + (self.yarn_count -
1) * self.yarn_spacing

  def yarn_shape(self):
    "Method for defining the yarn cross section shape"
    if self.shape == "Lenticular":
      return CSectionLenticular(self.yarn_width, self.yarn_thickness)
    elif self.shape == "Ellipse":
      return CSectionEllipse(self.yarn_width, self.yarn_thickness)
    elif self.shape == "PowerEllipse":
      return CSectionPowerEllipse(self.yarn_width, self.yarn_thickness,
self.shape_factor)


class WoundPattern:
  """Class defining the wound pattern to the generated pattern"""

  def __init__(self, band_straight: object, band_angled: object, band_count:
int, band_space: float, angle: float, band_gap_z: float = 0, render_hoop:
bool = False) -> None:
    # -- Exceptions
    if band_count <= 0:
      raise ValueError("A postive, integer number of yarns must be defined")
    if angle < 0 or angle > 70:
      raise ValueError("Yarn angle must be between 0 degrees and 70 degrees")
```

```
    if band_space < 0:
      raise ValueError("Yarn spacing must be greater than or equal to 0")

    # -- Primitive Pattern Attributes
    self.band_straight = band_straight
    self.band_angled = band_angled
    self.band_gap_z = band_gap_z
    self.band_space = band_space
    self.band_count = band_count
    self.render_hoop = render_hoop

    # -- Derived Local Pattern Attributes
    self._ANGLE = angle * math.pi / 180
    self._VB_OFFSET = (self.band_angled.band_width / 2) /
math.cos(self._ANGLE)
    self._VB_OFFSET_GAP = self.band_space / (math.cos(self._ANGLE))
    self._XB_DOMAIN_LENGTH = self.band_count * self.band_straight.band_width
+\ (self.band_count - 1) * self.band_space
    self._YB_DOMAIN_LENGTH = self._XB_DOMAIN_LENGTH * (1 +
math.sin(self._ANGLE))

    # -- Private TexGen Band Lists
    self._band_list_straight =
self.__generate_Bandlist(band=self.band_straight)
    self._band_list_angled = self.__generate_Bandlist(band=self.band_angled)
    self.extra_band_list = self.__generate_Bandlist(band=band_angled)
    self.extra_band_list_lower = self.__generate_Bandlist(band=band_angled)
    self.hoop_band_list = self.__generate_Bandlist(band=band_angled)
    self.extra_band_list.pop(), self.extra_band_list_lower.pop()

    self._Textile = CTextile()
    self.bottom_y_domain = self._YB_DOMAIN_LENGTH - self.band_count *
self.band_angled.band_width / math.cos(self._ANGLE) - (self.band_count) *
self.band_space / math.cos(self._ANGLE) - self._XB_DOMAIN_LENGTH *
math.tan(self._ANGLE)
    self.top_y_domain = 0.5 * self.band_space / math.cos(self._ANGLE)
    # + (self.band_angled.yarn_width + 2 * self.band_space) / (2 *
math.cos(self._ANGLE))

  def __generate_Bandlist(self, band: object) -> list[list]:
    "Returns a tuple containing tuples of yarn objects for each yarn in each
band"
    return [[CYarn() for _ in range(band.yarn_count)] for _ in
range(self.band_count)]

  def __get_diamond_pattern(self) -> None:
    "Private method for calculating straight sections inside WoundPattern
class"

    inc = self.band_straight.band_width / 4 # -- mm
    for ib, band in enumerate(self._band_list_straight):
      # -- starting center coordinate for each angled band
      x_ab0 = self._XB_DOMAIN_LENGTH
      y_ab0 = self._YB_DOMAIN_LENGTH - ib * (self.band_angled.band_width +
self.band_space) / math.cos(self._ANGLE) - 0.5 * self.band_angled.yarn_width
/ math.cos(self._ANGLE)
```

```
      y_ab0_hoop = self._YB_DOMAIN_LENGTH - ib * (self.band_angled.band_width
+ self.band_space) / math.cos(self._ANGLE) - 0.5 *
self.band_angled.yarn_width / math.cos(self._ANGLE)
      z_ab0 = 0 - self.band_gap_z / 2 if ib == len(self._band_list_straight)
- 1 else self.band_straight.yarn_thickness + self.band_gap_z / 2
      # -- starting center coordinate for each straight band
      x_sb0 = (ib + .5) * self.band_straight.band_width + ib *
self.band_space
      # - 0.5 * self.band_space
      y_sb0 = 0
      z_sb0 = self.band_straight.yarn_thickness - self.band_gap_z / 2
      # -- end coordinate of each angled band
      x_ab1 = x_sb0
      y_ab1 = y_ab0 - abs(x_ab1 - x_ab0) * math.tan(self._ANGLE)
      z_ab1 = 0 - self.band_gap_z / 2 if ib == len(self._band_list_straight)
- 1 else self.band_straight.yarn_thickness + self.band_gap_z / 2

      for iy, yarn in enumerate(band):
        # -- adjusts the y coordinate in each band based on angle
        seq_yarn_y_adjust = (self.band_straight.yarn_width +
self.band_straight.yarn_spacing) * math.tan(self._ANGLE)

        # -- starting coordinate for each angled yarn relative to center
starting coordinate of angled band ib
        x_ay0 = x_ab0
        y_ay0 = y_ab0 - iy * (self.band_straight.yarn_width +
self.band_straight.yarn_spacing) / math.cos(self._ANGLE)
        z_ay0 = z_ab0
        # -- starting coordinate for each straight yarn relative to center of
straight band ib
        x_sy0 = x_sb0 + (iy - 0.5 * (self.band_straight.yarn_count - 1)) *
(self.band_straight.yarn_width + self.band_straight.yarn_spacing)
        y_sy0 = y_sb0 - self.band_straight.yarn_width + iy *
seq_yarn_y_adjust
        z_sy0 = self.band_straight.yarn_thickness - self.band_gap_z / 2
        # -- end coordinate for each angled yarn relative to center end
coordinate of angled band ib
        x_ay1 = x_sb0 + 0.5 * self.band_straight.band_width +
1.5*self.band_space
        y_ay1 = y_ay0 - abs(x_ay0 - x_sb0 - 0.5 *
self.band_straight.band_width - 1.5*self.band_space)* math.tan(self._ANGLE)
        z_ay1 = z_ab1
        # -- end coordinate for each straight yarn relative to center of
straight band ib
        x_sy1 = x_sb0 + (iy - 0.5 * (self.band_straight.yarn_count - 1)) *
(self.band_straight.yarn_width + self.band_straight.yarn_spacing)
        y_sy1 = y_ab1 - (self.band_straight.yarn_count / 2 - 0.5) *
self.band_straight.yarn_width * math.tan(self._ANGLE) -
(self.band_straight.yarn_count / 2 - 0.5) * self.band_straight.yarn_width /
math.cos(self._ANGLE) - ((self.band_straight.yarn_count - 1) / 2) *
self.band_straight.yarn_spacing * math.tan(self._ANGLE) + iy *
seq_yarn_y_adjust - \
        ((self.band_straight.yarn_count / 2) * self.band_straight.yarn_width
/ math.cos(self._ANGLE) + 0.5 * self.band_space / math.cos(self._ANGLE))
        # y_sy1 = (y_ab1 - (self.band_straight.yarn_count / 2 - 0.5) *
self.band_straight.yarn_width * math.tan(self._ANGLE) -
(self.band_straight.yarn_count / 2 - 0.5) * self.band_straight.yarn_width /
```

```
math.cos(self._ANGLE) - ((self.band_straight.yarn_count - 1) / 2) *
self.band_straight.yarn_spacing * math.tan(self._ANGLE) -
((self.band_straight.yarn_count - 1) / 2)* self.band_angled.yarn_spacing /
math.cos(self._ANGLE)) + iy * seq_yarn_y_adjust - \
        # ((self.band_straight.yarn_count / 2) *
self.band_straight.yarn_width / math.cos(self._ANGLE) +
((self.band_straight.yarn_count - 1) / 2) * self.band_angled.yarn_spacing /
math.cos(self._ANGLE) + 0.5 * self.band_space / math.cos(self._ANGLE))
        z_sy1 = z_sb0

        # -- Points for straight yarns
        # point_list_straight_bot_0 = [(x_sy0, -0.5 * self._YB_DOMAIN_LENGTH,
z_sy0 - self.band_straight.yarn_thickness), (x_sy0, y_sy1 - (self.band_count
- ib - 1) * ( self.band_angled.band_width + self.band_space) /
math.cos(self._ANGLE) - self.band_space / math.cos(self._ANGLE), z_sy0 -
self.band_straight.yarn_thickness)]
        # point_list_straight_top_0 = [(x_sy0, y_sy1 - (self.band_count - ib
- 1) * ( self.band_angled.band_width + self.band_space) /
math.cos(self._ANGLE) + 1 * self.band_space / math.cos(self._ANGLE),
self.band_angled.yarn_thickness + self.band_gap_z / 2), (x_sy0, y_sy1 +
self.band_angled.band_width / math.cos(self._ANGLE),
self.band_angled.yarn_thickness + self.band_gap_z / 2)]
        # point_list_straight_bot_1 = [(x_sy0, y_sy1 +
self.band_angled.band_width / math.cos(self._ANGLE) + 2*self.band_space /
math.cos(self._ANGLE), z_sy0 - self.band_straight.yarn_thickness), (x_sy0,
1.25*self._YB_DOMAIN_LENGTH, z_sy0 - self.band_straight.yarn_thickness)]
        point_list_straight_bot_0 = [(x_sy0, -0.5 * self._YB_DOMAIN_LENGTH,
z_sy0 - self.band_straight.yarn_thickness), (x_sy0, y_sy1 - (self.band_count
- ib - 1) * ( self.band_angled.band_width + self.band_space) /
math.cos(self._ANGLE) - 2 * self.band_space / math.cos(self._ANGLE), z_sy0 -
self.band_straight.yarn_thickness)]
        point_list_straight_top_0 = [(x_sy0, y_sy1 - (self.band_count - ib -
1) * ( self.band_angled.band_width + self.band_space) / math.cos(self._ANGLE)
+ 2 * self.band_space / math.cos(self._ANGLE),
self.band_angled.yarn_thickness + self.band_gap_z / 2), (x_sy0, y_sy1 +
self.band_angled.band_width / math.cos(self._ANGLE) - 1 * self.band_space /
math.cos(self._ANGLE), self.band_angled.yarn_thickness + self.band_gap_z /
2)]
        point_list_straight_bot_1 = [(x_sy0, y_sy1 +
self.band_angled.band_width / math.cos(self._ANGLE) + 3*self.band_space /
math.cos(self._ANGLE), z_sy0 - self.band_straight.yarn_thickness), (x_sy0,
1.25*self._YB_DOMAIN_LENGTH, z_sy0 - self.band_straight.yarn_thickness)]
        # -- Points for angled yarns
        # point_list_angled_0 = [(1.25 * self._XB_DOMAIN_LENGTH, y_ay0 +
abs(.25 * self._XB_DOMAIN_LENGTH) * math.tan(self._ANGLE), z_ay1 -
self.band_gap_z/2), (x_ay1, y_ay1, z_ay1 - self.band_gap_z/2)]
        # point_list_angled_1 = [(x_ay1 - 2 * self.band_space, y_ay1 - abs(2
* self.band_space) * math.tan(self._ANGLE), z_ay1 - self.band_gap_z/2 -
self.band_angled.yarn_thickness), (-0.25 * self._XB_DOMAIN_LENGTH, y_ay1 -
abs(0.25 * self._XB_DOMAIN_LENGTH + (x_ay1 - 2 * self.band_space)) *
math.tan(self._ANGLE) - abs(2 * self.band_space) * math.tan(self._ANGLE),
z_ay1 - self.band_gap_z/2 - self.band_angled.yarn_thickness)]
        # point_list_angled_11 = [(1.25 * self._XB_DOMAIN_LENGTH, y_ay0 +
abs(.25 * self._XB_DOMAIN_LENGTH) * math.tan(self._ANGLE), z_ay1 +
self.band_angled.yarn_thickness / 2 - self.band_gap_z / 2), (-.25 *
self._XB_DOMAIN_LENGTH, y_ay0 - abs(1.25 * self._XB_DOMAIN_LENGTH) *
```

```
math.tan(self._ANGLE), z_ay1 + self.band_angled.yarn_thickness / 2 -
self.band_gap_z / 2)]

        # point_list_angled_lower_0 = [(1.25 * self._XB_DOMAIN_LENGTH, y_ay0
+ abs(.25 * self._XB_DOMAIN_LENGTH) * math.tan(self._ANGLE), z_ay1 -
self.band_gap_z / 2 - self.band_angled.yarn_thickness),
(self._XB_DOMAIN_LENGTH + self.band_space, y_ay0 + abs(self.band_space) *
math.tan(self._ANGLE), z_ay1 - self.band_gap_z / 2 -
self.band_angled.yarn_thickness)]
        point_list_angled_0 = [(1.25*self._XB_DOMAIN_LENGTH -
self.band_space, y_ay0 + abs(0.25 * self._XB_DOMAIN_LENGTH) *
math.tan(self._ANGLE), z_ay1 - self.band_gap_z/2), (x_ay1, y_ay1, z_ay1 -
self.band_gap_z/2)]
        # point_list_angled_0 = [(self._XB_DOMAIN_LENGTH - self.band_space,
y_ay0 - abs(self.band_space) * math.tan(self._ANGLE), z_ay1 -
self.band_gap_z/2), (x_ay1, y_ay1, z_ay1 - self.band_gap_z/2)]
        point_list_angled_1 = [(x_ay1 - 3 * self.band_space, y_ay1 - abs(3 *
self.band_space) * math.tan(self._ANGLE), z_ay1 - self.band_gap_z/2 -
self.band_angled.yarn_thickness), (-0.25 * self._XB_DOMAIN_LENGTH, y_ay1 -
abs(0.25 * self._XB_DOMAIN_LENGTH + (x_ay1 - 1 * self.band_space)) *
math.tan(self._ANGLE) - abs(1 * self.band_space) * math.tan(self._ANGLE),
z_ay1 - self.band_gap_z/2 - self.band_angled.yarn_thickness)]

        point_list_angled_lower_11_0 = [(1.25 * self._XB_DOMAIN_LENGTH, y_ay0
+ abs(.25 * self._XB_DOMAIN_LENGTH) * math.tan(self._ANGLE), z_ay1 +
0*self.band_gap_z / 2 + 0*self.band_angled.yarn_thickness),
(self._XB_DOMAIN_LENGTH + self.band_space, y_ay0 + abs(self.band_space) *
math.tan(self._ANGLE), z_ay1 + 0*self.band_gap_z / 2 +
0*self.band_angled.yarn_thickness)]
        point_list_angled_11 = [(1.25 * self._XB_DOMAIN_LENGTH, y_ay0 +
abs(.25 * self._XB_DOMAIN_LENGTH) * math.tan(self._ANGLE), z_ay1 +
self.band_gap_z/2), (-.25 * self._XB_DOMAIN_LENGTH, y_ay0 - abs(1.25 *
self._XB_DOMAIN_LENGTH) * math.tan(self._ANGLE), z_ay1 + self.band_gap_z/2)]
        point_list_angled_11 = [(self._XB_DOMAIN_LENGTH - self.band_space,
y_ay0 - abs(self.band_space) * math.tan(self._ANGLE), z_ay1 +
self.band_gap_z/2), (-.25 * self._XB_DOMAIN_LENGTH, y_ay0 - abs(1.25 *
self._XB_DOMAIN_LENGTH) * math.tan(self._ANGLE), z_ay1 + self.band_gap_z/2)]


        # point_list_angled_e_upper = [(1.25 * self._XB_DOMAIN_LENGTH, y_ay0
+ abs(.25 * self._XB_DOMAIN_LENGTH) * math.tan(self._ANGLE) + (2*ib + 1) *
(self.band_angled.band_width + self.band_space) / math.cos(self._ANGLE),
z_ay1 - self.band_gap_z / 2), (-.25 * self._XB_DOMAIN_LENGTH, y_ay0 -
abs(1.25 * self._XB_DOMAIN_LENGTH) * math.tan(self._ANGLE) + (2*ib + 1) *
(self.band_angled.band_width + self.band_space) / math.cos(self._ANGLE),
z_ay1 - self.band_gap_z / 2)]
        # point_list_angled_e_lower_0 = [(1.25 * self._XB_DOMAIN_LENGTH,
y_ay0 + abs(.25 * self._XB_DOMAIN_LENGTH) * math.tan(self._ANGLE) -
(self.band_count) * (self.band_angled.band_width + self.band_space) /
math.cos(self._ANGLE), z_ay1 - self.band_gap_z / 2 -
self.band_angled.yarn_thickness), (self._XB_DOMAIN_LENGTH + self.band_space,
y_ay0 + abs(self.band_space) * math.tan(self._ANGLE) - (self.band_count) *
(self.band_angled.band_width + self.band_space) / math.cos(self._ANGLE),
z_ay1 - self.band_gap_z / 2 - self.band_angled.yarn_thickness)]
        # point_list_angled_e_lower_1 = [(self._XB_DOMAIN_LENGTH, y_ay0 +
abs(0 * self._XB_DOMAIN_LENGTH) * math.tan(self._ANGLE) - (self.band_count) *
(self.band_angled.band_width + self.band_space) / math.cos(self._ANGLE),
```

```
z_ay1 - self.band_gap_z / 2), (-.25 * self._XB_DOMAIN_LENGTH, y_ay0 -
abs(1.25 * self._XB_DOMAIN_LENGTH) * math.tan(self._ANGLE) -
(self.band_count) * (self.band_angled.band_width + self.band_space) /
math.cos(self._ANGLE), z_ay1 - self.band_gap_z / 2)]
        point_list_angled_e_upper = [(1.25 * self._XB_DOMAIN_LENGTH, y_ay0 +
abs(.25 * self._XB_DOMAIN_LENGTH) * math.tan(self._ANGLE) + (2*ib + 1) *
(self.band_angled.band_width + self.band_space) / math.cos(self._ANGLE),
z_ay1 - self.band_gap_z / 2), (-.25 * self._XB_DOMAIN_LENGTH, y_ay0 -
abs(1.25 * self._XB_DOMAIN_LENGTH) * math.tan(self._ANGLE) + (2*ib + 1) *
(self.band_angled.band_width + self.band_space) / math.cos(self._ANGLE),
z_ay1 - self.band_gap_z / 2)]
        # point_list_angled_e_lower_0 = [(1.25 * self._XB_DOMAIN_LENGTH,
y_ay0 + abs(.25 * self._XB_DOMAIN_LENGTH) * math.tan(self._ANGLE) -
(self.band_count) * (self.band_angled.band_width + self.band_space) /
math.cos(self._ANGLE), z_ay1 - self.band_gap_z / 2 -
0*self.band_angled.yarn_thickness), (self._XB_DOMAIN_LENGTH +
self.band_space, y_ay0 + abs(self.band_space) * math.tan(self._ANGLE) -
(self.band_count) * (self.band_angled.band_width + self.band_space) /
math.cos(self._ANGLE), z_ay1 - self.band_gap_z / 2 -
0*self.band_angled.yarn_thickness)]
        point_list_angled_e_lower_0 = [(1.25 * self._XB_DOMAIN_LENGTH, y_ay0
+ abs(.25 * self._XB_DOMAIN_LENGTH) * math.tan(self._ANGLE) -
(self.band_count) * (self.band_angled.band_width + self.band_space) /
math.cos(self._ANGLE), z_ay1 - self.band_gap_z / 2 -
self.band_angled.yarn_thickness), (self._XB_DOMAIN_LENGTH + self.band_space,
y_ay0 + abs(self.band_space) * math.tan(self._ANGLE) - (self.band_count) *
(self.band_angled.band_width + self.band_space) / math.cos(self._ANGLE),
z_ay1 - self.band_gap_z / 2 - self.band_angled.yarn_thickness)]
        point_list_angled_e_lower_1 = [(self._XB_DOMAIN_LENGTH -
self.band_space, y_ay0 - abs(0 * self._XB_DOMAIN_LENGTH - self.band_space) *
math.tan(self._ANGLE) - (self.band_count) * (self.band_angled.band_width +
self.band_space) / math.cos(self._ANGLE), z_ay1 - self.band_gap_z / 2), (-.25
* self._XB_DOMAIN_LENGTH, y_ay0 - abs(1.25 * self._XB_DOMAIN_LENGTH -
self.band_space) * math.tan(self._ANGLE) - (self.band_count) *
(self.band_angled.band_width + self.band_space) / math.cos(self._ANGLE),
z_ay1 - self.band_gap_z / 2)]
        # -- Points for hoop layers
        point_list_hoop_bot = [(1.25 * self._XB_DOMAIN_LENGTH, y_ab0_hoop -
iy * (self.band_straight.yarn_width + self.band_straight.yarn_spacing) /
math.cos(self._ANGLE) - abs(.25 * self._XB_DOMAIN_LENGTH) *
math.tan(self._ANGLE), z_ay1 + 2.5 * self.band_angled.yarn_thickness / 2), (-
.25 * self._XB_DOMAIN_LENGTH, y_ab0_hoop - iy *
(self.band_straight.yarn_width + self.band_straight.yarn_spacing) /
math.cos(self._ANGLE) + abs(1.25 * self._XB_DOMAIN_LENGTH) *
math.tan(self._ANGLE), z_ay1 + 2.5 * self.band_angled.yarn_thickness / 2)]
        point_list_hoop_top = [(1.25 * self._XB_DOMAIN_LENGTH, y_ab0_hoop -
iy * (self.band_straight.yarn_width + self.band_straight.yarn_spacing) /
math.cos(self._ANGLE) - abs(.25 * self._XB_DOMAIN_LENGTH) *
math.tan(self._ANGLE) - 0.5*(self.band_angled.band_width + self.band_space) /
math.cos(self._ANGLE), z_ay1 + 7 * self.band_angled.yarn_thickness / 2 +
self.band_gap_z), (-.25 * self._XB_DOMAIN_LENGTH, y_ab0_hoop - iy *
(self.band_straight.yarn_width + self.band_straight.yarn_spacing) /
math.cos(self._ANGLE) + abs(1.25 * self._XB_DOMAIN_LENGTH) *
math.tan(self._ANGLE) - 0.5*(self.band_angled.band_width + self.band_space) /
math.cos(self._ANGLE), z_ay1 + 7 * self.band_angled.yarn_thickness / 2 +
self.band_gap_z)]
```

```
        # -- Yarn Generation
        if ib < self.band_count - 1:
          create_yarn_segment(coordinate_list=point_list_straight_bot_0,
yarn=yarn, yarn_shape=self.band_straight.yarn_shape(), increment=inc,
rotation=self._ANGLE)
          create_yarn_segment(coordinate_list=point_list_straight_top_0,
yarn=yarn, yarn_shape=self.band_straight.yarn_shape(), increment=inc,
rotation=self._ANGLE)
          create_yarn_segment(coordinate_list=point_list_straight_bot_1,
yarn=yarn, yarn_shape=self.band_straight.yarn_shape(), increment=inc,
rotation=self._ANGLE)

          # -- new
          # create_yarn_segment(coordinate_list=point_list_angled_lower_0,
yarn=self._band_list_angled[ib][iy],
yarn_shape=self.band_angled.yarn_shape(), increment=inc, rotation=-
self._ANGLE)

          create_yarn_segment(coordinate_list=point_list_angled_0,
yarn=self._band_list_angled[ib][iy],
yarn_shape=self.band_angled.yarn_shape(), increment=inc, rotation=-
self._ANGLE)
          create_yarn_segment(coordinate_list=point_list_angled_1,
yarn=self._band_list_angled[ib][iy],
yarn_shape=self.band_angled.yarn_shape(), increment=inc, rotation=-
self._ANGLE)
          create_yarn_segment(coordinate_list=point_list_angled_e_upper,
yarn=self.extra_band_list[ib][iy], yarn_shape=self.band_angled.yarn_shape(),
increment=inc, rotation=-self._ANGLE)
          create_yarn_segment(coordinate_list=point_list_angled_e_lower_0,
yarn=self.extra_band_list_lower[ib][iy],
yarn_shape=self.band_angled.yarn_shape(), increment=inc, rotation=-
self._ANGLE)
          create_yarn_segment(coordinate_list=point_list_angled_e_lower_1,
yarn=self.extra_band_list_lower[ib][iy],
yarn_shape=self.band_angled.yarn_shape(), increment=inc, rotation=-
self._ANGLE)
          self.__instantiate_yarn(yarn_list=self.extra_band_list[ib])
          self.__instantiate_yarn(yarn_list=self.extra_band_list_lower[ib])

        else:
          create_yarn_segment(coordinate_list=point_list_straight_bot_0,
yarn=yarn, yarn_shape=self.band_straight.yarn_shape(), increment=inc,
rotation=self._ANGLE)
          create_yarn_segment(coordinate_list=point_list_straight_top_0,
yarn=yarn, yarn_shape=self.band_straight.yarn_shape(), increment=inc,
rotation=self._ANGLE)
          create_yarn_segment(coordinate_list=point_list_straight_bot_1,
yarn=yarn, yarn_shape=self.band_straight.yarn_shape(), increment=inc,
rotation=self._ANGLE)

          create_yarn_segment(coordinate_list=point_list_angled_lower_11_0,
yarn=self._band_list_angled[ib][iy],
yarn_shape=self.band_angled.yarn_shape(), increment=inc, rotation=-
self._ANGLE)
          create_yarn_segment(coordinate_list=point_list_angled_11,
yarn=self._band_list_angled[ib][iy],
```

```
yarn_shape=self.band_angled.yarn_shape(), increment=inc, rotation=-
self._ANGLE)

        if self.render_hoop is True:
          # -- Adds Hoop Layers
          if ib == 0:
            create_yarn_segment(coordinate_list=point_list_hoop_bot,
yarn=self.hoop_band_list[0][iy], yarn_shape=self.band_angled.yarn_shape(),
increment=inc, rotation=self._ANGLE)
            repeat_vector_bot = XYZ(0, (self.band_space +
self.band_angled.band_width) / math.cos(self._ANGLE), 0)
            self.hoop_band_list[ib][iy].SetRepeats([repeat_vector_bot])

          elif ib == self.band_count-1:
            create_yarn_segment(coordinate_list=point_list_hoop_top,
yarn=self.hoop_band_list[ib][iy], yarn_shape=self.band_angled.yarn_shape(),
increment=inc, rotation=self._ANGLE)
            repeat_vector_top = XYZ(0, (self.band_space +
self.band_angled.band_width) / math.cos(self._ANGLE), 0)
            self.hoop_band_list[ib][iy].SetRepeats([repeat_vector_top])

      # -- Interpolates over nodes to create yarn
      self.__instantiate_yarn(yarn_list=band)
      self.__instantiate_yarn(yarn_list=self._band_list_angled[ib])
    self.__instantiate_yarn(yarn_list=self.hoop_band_list[0])
    self.__instantiate_yarn(yarn_list=self.hoop_band_list[-1])

  def __get_lamination_pattern(self) -> None:

    inc = self.band_straight.band_width / 4 # -- mm
    for ib, band in enumerate(self._band_list_straight):
      # -- starting center coordinate for each angled band
      x_ab0 = 1.25 * self._XB_DOMAIN_LENGTH
      y_ab0 = self._YB_DOMAIN_LENGTH - ib * (self.band_angled.band_width +
self.band_space) / math.cos(self._ANGLE) - 0.5 * self.band_angled.yarn_width
/ math.cos(self._ANGLE)
      y_ab0_hoop = self._YB_DOMAIN_LENGTH - ib * (self.band_angled.band_width
+ self.band_space) / math.cos(self._ANGLE) - 0.5 *
self.band_angled.yarn_width / math.cos(self._ANGLE)
      z_ab0 = self.band_straight.yarn_thickness + self.band_gap_z / 2
      # -- starting center coordinate for each straight band
      x_sb0 = (ib + .5) * self.band_straight.band_width + ib *
self.band_space
      # - 0.5 * self.band_space
      y_sb0 = 0
      z_sb0 = self.band_straight.yarn_thickness - self.band_gap_z / 2
      # -- end coordinate of each angled band
      x_ab1 = -0.25 * self._XB_DOMAIN_LENGTH
      y_ab1 = y_ab0 - abs(x_ab1 - x_ab0) * math.tan(self._ANGLE)
      z_ab1 = self.band_straight.yarn_thickness + self.band_gap_z / 2

      for iy, yarn in enumerate(band):
        # -- adjusts the y coordinate in each band based on angle
        seq_yarn_y_adjust = (self.band_straight.yarn_width +
self.band_straight.yarn_spacing) * math.tan(self._ANGLE)
```

```
        # -- starting coordinate for each angled yarn relative to center
starting coordinate of angled band ib
        x_ay0 = x_ab0
        y_ay0 = y_ab0 - iy * (self.band_straight.yarn_width +
self.band_straight.yarn_spacing) / math.cos(self._ANGLE)
        z_ay0 = z_ab0
        # -- starting coordinate for each straight yarn relative to center of
straight band ib
        x_sy0 = x_sb0 + (iy - 0.5 * (self.band_straight.yarn_count - 1)) *
(self.band_straight.yarn_width + self.band_straight.yarn_spacing)
        y_sy0 = - 0.25 * self._YB_DOMAIN_LENGTH
        z_sy0 = self.band_straight.yarn_thickness - self.band_gap_z / 2
        # -- end coordinate for each angled yarn relative to center end
coordinate of angled band ib
        x_ay1 = x_ab1
        # y_ay1 = y_ab0 - iy * (self.band_straight.yarn_width +
self.band_straight.yarn_spacing) / math.cos(self._ANGLE)
        y_ay1 = y_ay0 - abs(x_ay1 - x_ay0)* math.tan(self._ANGLE)
        z_ay1 = z_ab1
        # -- end coordinate for each straight yarn relative to center of
straight band ib
        x_sy1 = x_sb0 + (iy - 0.5 * (self.band_straight.yarn_count - 1)) *
(self.band_straight.yarn_width + self.band_straight.yarn_spacing)
        y_sy1 = 1.25 * self._YB_DOMAIN_LENGTH
        z_sy1 = z_sb0

        point_straight = [(x_sy0, y_sy0, z_sy0 -
self.band_straight.yarn_thickness / 2 - self.band_gap_z / 2), (x_sy1, y_sy1,
z_sy1 - self.band_straight.yarn_thickness / 2 - self.band_gap_z / 2)]
        point_angled = [(x_ay0, y_ay0, z_ay0), (x_ay1, y_ay1, z_ay1)]
        point_list_hoop_bot = [(1.25 * self._XB_DOMAIN_LENGTH, y_ab0_hoop -
iy * (self.band_straight.yarn_width + self.band_straight.yarn_spacing) /
math.cos(self._ANGLE) - abs(.25 * self._XB_DOMAIN_LENGTH) *
math.tan(self._ANGLE), z_ay1 + 2.5 * self.band_angled.yarn_thickness / 2), (-
.25 * self._XB_DOMAIN_LENGTH, y_ab0_hoop - iy *
(self.band_straight.yarn_width + self.band_straight.yarn_spacing) /
math.cos(self._ANGLE) + abs(1.25 * self._XB_DOMAIN_LENGTH) *
math.tan(self._ANGLE), z_ay1 + 2.5 * self.band_angled.yarn_thickness / 2)]
        point_list_hoop_top = [(1.25 * self._XB_DOMAIN_LENGTH, y_ab0_hoop -
iy * (self.band_straight.yarn_width + self.band_straight.yarn_spacing) /
math.cos(self._ANGLE) - abs(.25 * self._XB_DOMAIN_LENGTH) *
math.tan(self._ANGLE) - 0.5*(self.band_angled.band_width + self.band_space) /
math.cos(self._ANGLE), z_ay1 + 4.5 * self.band_angled.yarn_thickness / 2 +
self.band_gap_z / 2), (-.25 * self._XB_DOMAIN_LENGTH, y_ab0_hoop - iy *
(self.band_straight.yarn_width + self.band_straight.yarn_spacing) /
math.cos(self._ANGLE) + abs(1.25 * self._XB_DOMAIN_LENGTH) *
math.tan(self._ANGLE) - 0.5*(self.band_angled.band_width + self.band_space) /
math.cos(self._ANGLE), z_ay1 + 4.5 * self.band_angled.yarn_thickness / 2 +
self.band_gap_z / 2)]

        if ib == 0:
          create_yarn_segment(coordinate_list=point_straight, yarn=yarn,
yarn_shape=self.band_straight.yarn_shape(), increment=inc,
rotation=self._ANGLE)
          create_yarn_segment(coordinate_list=point_angled,
yarn=self._band_list_angled[ib][iy],
```

```
yarn_shape=self.band_angled.yarn_shape(), increment=inc, rotation=-
self._ANGLE)
          create_yarn_segment(coordinate_list=point_list_hoop_bot,
yarn=self.hoop_band_list[0][iy], yarn_shape=self.band_angled.yarn_shape(),
increment=inc, rotation=self._ANGLE)

          repeat_vector_straight = XYZ(self.band_straight.band_width +
self.band_space, 0, 0)
          repeat_vector_angled = XYZ(0, (self.band_straight.band_width +
self.band_space) / math.cos(self._ANGLE), 0)
          repeat_vector_bot = XYZ(0, (self.band_space +
self.band_angled.band_width) / math.cos(self._ANGLE), 0)

          self.hoop_band_list[ib][iy].SetRepeats([repeat_vector_bot])
          yarn.SetRepeats([repeat_vector_straight])
          self._band_list_angled[ib][iy].SetRepeats([repeat_vector_angled])

          # self.__instantiate_yarn(yarn_list=band)
          self.__instantiate_yarn(yarn_list=self._band_list_angled[ib])

        # elif ib == self.band_count-1:
        #   create_yarn_segment(coordinate_list=point_list_hoop_top,
yarn=self.hoop_band_list[ib][iy], yarn_shape=self.band_angled.yarn_shape(),
increment=inc, rotation=self._ANGLE)
        #   repeat_vector_top = XYZ(0, (self.band_space +
self.band_angled.band_width) / math.cos(self._ANGLE), 0)
        #   self.hoop_band_list[ib][iy].SetRepeats([repeat_vector_top])

    # self.__instantiate_yarn(yarn_list=self.hoop_band_list[0])
    # self.__instantiate_yarn(yarn_list=self.hoop_band_list[-1])

  def __instantiate_yarn(self, yarn_list) -> None:

    for yarn in yarn_list:
      # Set the interpolation function
      yarn.AssignInterpolation(CInterpolationCubic(True, True, True))
      # Set the resolution of the surface mesh created
      yarn.SetResolution(500, 50)
      # Add yarn to textile in TexGen
      self._Textile.AddYarn(yarn)

  def generate_wound_pattern(self) -> None:
    # -- Assign straigh sections to pattern
    self.__get_diamond_pattern()
    # self.__get_lamination_pattern()


    # Create a domain and assign it to the textile
    self._Textile.AssignDomain(CDomainPlanes(XYZ(0, self.bottom_y_domain, -1
* YARN_THICK), XYZ(self._XB_DOMAIN_LENGTH + self.band_space,
self._YB_DOMAIN_LENGTH + self.top_y_domain, 2*self.band_gap_z + 4 *
YARN_THICK)))
    # 4.5 * YARN_THICK
    # Add the textile
    AddTextile("Wound_Pattern_{}-Deg_{}-Bands".format(round(self._ANGLE * 180
/ math.pi), self.band_count), self._Textile)
```

```
#############################################################################
###
#-- Utility Functions
#############################################################################
###

def create_yarn_segment(coordinate_list: list, yarn: object, yarn_shape:
object, increment: float, rotation: float = 0) -> None:

  for i in range(len(coordinate_list)-1):
    assign_nodes(yarn=yarn, yarn_shape=yarn_shape,
coordinate_vector_0=coordinate_list[i],
coordinate_vector_1=coordinate_list[i+1], increment=increment,
cs_rotation=rotation)

def assign_nodes(yarn: object, yarn_shape: object, coordinate_vector_0:
tuple, coordinate_vector_1: tuple, increment: float, cs_rotation: float = 0,
) -> None:
  """Assigns a specified number of nodes to a specified yarn"""
  x0, y0, z0 = coordinate_vector_0

  dist = get_distance(coordinate_vector_0=coordinate_vector_0,
coordinate_vector_1=coordinate_vector_1)
  n_nodes = math.ceil(dist / increment)
  dx, dy, dz = vector_subtact(coordinate_vector_0=coordinate_vector_0,
coordinate_vector_1=coordinate_vector_1)
  inc_x, inc_y, inc_z = dx / n_nodes, dy / n_nodes, dz / n_nodes
  shape = CSectionPowerEllipse(YARN_WIDTH / math.cos(cs_rotation),
YARN_THICK, SHAPE_FACTOR)

  for j in range(n_nodes+1):
    node = CNode(XYZ(x0 + j*inc_x, y0 + j*inc_y, z0 + j*inc_z))
    node.SetAngle(cs_rotation)
    yarn.AddNode(node)

  yarn.AssignSection(CYarnSectionConstant(shape))

def vector_subtact(coordinate_vector_0: tuple, coordinate_vector_1: tuple) ->
tuple:
  "Performs element-wise subtraction on two vectors of length 3"
  if len(coordinate_vector_0) != len(coordinate_vector_1):
    raise ValueError("Vector 1 and vector 2 are not equal in length: ({} vs
{})".format(len(coordinate_vector_0), len(coordinate_vector_1)))

  x0, y0, z0 = coordinate_vector_0
  x, y, z = coordinate_vector_1

  return x - x0, y - y0, z - z0

def get_distance(coordinate_vector_0: tuple, coordinate_vector_1: tuple) ->
float:
  "Returns the distance between 2 points in 3D space"
  if len(coordinate_vector_0) != len(coordinate_vector_1):
    raise ValueError("Vector 1 and vector 2 are not equal in length: ({} vs
{})".format(len(coordinate_vector_0), len(coordinate_vector_1)))
```

```
  x0, y0, z0 = coordinate_vector_0
  x1, y1, z1 = coordinate_vector_1

  return math.sqrt((x1 - x0)**2 + (y1 - y0)**2 + (z1 - z0)**2)

################################################################################
###
# -- User Defined Inputs
################################################################################
###

# -- Band Definition Constants
YARN_WIDTH: float = 4.0
YARN_THICK: float = 0.4
YARN_COUNT: int = 2
YARN_SPACE: float = 0.0
YARN_SHAPE: str = "PowerEllipse"
SHAPE_FACTOR: float = 0.6

# -- Pattern Definition Constants
BAND_ANGLE: float = 0
BAND_COUNT: int = 2
BAND_SPACE: float = .4
BAND_GAP_Z: float = 0.1

################################################################################
###
# -- End of User Defined Inputs
################################################################################
###

# -- Object Instantiation
BandStraight = BandDefinition(yarn_width=YARN_WIDTH,
yarn_thickness=YARN_THICK, shape=YARN_SHAPE,
               shape_factor=SHAPE_FACTOR, yarn_spacing=YARN_SPACE,
yarn_count=YARN_COUNT)
BandAngled = BandDefinition(yarn_width=YARN_WIDTH, yarn_thickness=YARN_THICK,
shape=YARN_SHAPE,
              shape_factor=SHAPE_FACTOR, yarn_spacing=YARN_SPACE,
yarn_count=YARN_COUNT)
Pattern = WoundPattern(band_straight=BandStraight, band_angled=BandAngled,
angle=BAND_ANGLE,
            band_gap_z=BAND_GAP_Z, band_count=BAND_COUNT,
band_space=BAND_SPACE, render_hoop=False)

# -- Create Wound Pattern
Pattern.generate_wound_pattern()
```

References

Aboudi, J., Arnold, S.M., and Bednarcyk, B.A. (2013): Micromechanics of Composite Materials: A Generalized Multiscale Analysis Approach. Elsevier, Oxford, UK.

Aboudi, J., Arnold, S.M., Bednarcyk, B.A.: Practical micromechanics of composite materials. Butterworth-Heinemann, Oxford, United Kingdom (2021).

Bednarcyk, B.A., Aboudi, J., Arnold, S.M.: Micromechanics of composite materials governed by vector constitutive laws. International Journal of Solids and Structures 110-111, 137–51 (2017). https://doi.org/10.1016/j.ijsolstr.2017.01.033.

Brown, L.P. and Long, A.C. (2021) "Modelling the geometry of textile reinforcements for composites: TexGen", Chapter 8 in "Composite reinforcements for optimum performance (Second Edition)", ed. P Boisse, Woodhead Publishing Ltd, 2021, ISBN: 978-0-12-819005-0. https://doi.org/10.1016/B978-0-12-819005-0.00008-3

Dassault Systemes. Abaqus (2024) [Computer Software]

Eschenauer, H., Olhoff, N., Schnell, W. (1997) Applied Structural Mechanics – Fundamentals of Elasticity, Load-Bearing Structures, Structural Optimization, Springer-Verlag.

Jones, R.M. (1999) Mechanics of Composite Materials, Taylor & Francis.

Mittelstedt, C. (2022) Flächentragwerke – Scheiben, Platten, Schalen, Geschichtete Strukturen, Springer Verlag.

Morozov, E.V. (2006) The effect of filament-winding mosaic patterns on the strength of thin-walled composite shells, Composite Structures 76, 123-129.

NASA (2024). NASA Multiscale Analysis Tool (NASMAT): LEW-20244-1; https://software.nasa.gov/software/LEW-20244-1, last accessed July 18, 2024.

Oller, S. (2014) Numerical Simulation of Mechanical Behaviour of Composite Materials. Springer, International Center for Numerical Methods in Engineering, Barcelona, Spain.

Pineda, E.J., Bednarcyk, B.A., Ricks, T.M., Arnold, S.M., Henson, G. (2021) Efficient Multiscale Recursive Micromechanics of Composites for Engineering Applications. International Journal for Multiscale Computational Engineering 19(4) https://doi.org10.1615/IntJMultCompEng. 2021039732.

Schürmann, H. (2007) Konstruieren mit Faser-Kunststoff-Verbunden, Springer-Verlag.

Suquet P.M. (1987) Elements of homogenization for inelastic solid mechanics. Homogenization Techniques for Composite Media. Ed. E. Sanchez-Palencia and A. Zaoui. Spring-Verlag, Berlin.

Talreja, R. (2024) Failure Analysis of Composite Materials with Manufacturing Defects, CRC Press.

Tew, B.W. (1995) Preliminary Design of Tubular Composite Structures Using Netting Theory and Composite Degradation Factors, ASME Journal of Pressure Vessel Technology.

Vinson, J.R. (1993) The Behavior of Shells Composed of Isotropic and Composite Materials, Springer Science.