A simple MATLAB draping code for fiber-reinforced composites with application to optimization of manufacturing process parameters

Krogh, Christian; Bak, Brian L.V.; Lindgaard, Esben; Olesen, Asbjørn M.; Hermansen, Sebastian M.; Broberg, Peter H.; Kepler, Jørgen A.; Lund, Erik; Jakobsen, Johnny

Published in:
Structural and Multidisciplinary Optimization

DOI (link to publication from Publisher):
10.1007/s00158-021-02925-z

Publication date:
2021

Document Version
Accepted author manuscript, peer reviewed version

A simple MATLAB draping code for fiber-reinforced composites with application to optimization of manufacturing process parameters

Christian Krogh, Brian L. V. Bak, Esben Lindgaard, Asbjørn M. Olesen, Sebastian M. Hermansen, Peter H. Broberg, Jørgen A. Kepler, Erik Lund & Johnny Jakobsen

Structural and Multidisciplinary Optimization manuscript No.
(will be inserted by the editor)

A simple MATLAB draping code for fiber-reinforced composites with application to optimization of manufacturing process parameters

Christian Krogh · Brian L.V. Bak · Esben Lindgaard · Asbjørn M. Olesen · Sebastian M. Hermansen · Peter H. Broberg · Jørgen A. Kepler · Erik Lund · Johnny Jakobsen

Received: date / Accepted: date

Abstract This paper presents a simple code written in MATLAB for simulating and optimizing the draping of a composite material fabric onto a mold. Being simple and kinematically based, the algorithm can be used to predict the final fiber orientations after a fabric has adapted to a double-curved mold surface. These fiber orientations will in turn govern the mechanical properties of the composite part and a draping analysis is thus advantageous in connection with a structural analysis as well as manufacturing. The code is intended for educational purposes and can be found in the appendix of the paper and on the repository at doi.org/10.5281/zenodo.4316860 along with a Python implementation. After a description of the code, various extensions are introduced, including a framework for optimization of the draping parameters.

Keywords Composites · Draping · Kinematic analysis · Manufacturing optimization

1 Introduction

This paper is intended for engineering education, specifically in courses covering analysis, optimization and manufacturing of laminated composite structures. The main focus is on draping, and students and newcomers to the field can get acquainted through the MATLAB code described in the paper.

Laminated composites offer excellent mechanical properties through tailoring of the layup. Structural designers specify the fiber orientations and ply stacking sequences, i.e. the instructions for manufacturing of the composite part. In the manufacturing step, the plies - essentially pieces of fabric - are *draped* on the mold surface either by hand (Fig. 1) or by means of some semi- or fully automatic process. For complex parts, the former is still widely used in the composite industry.

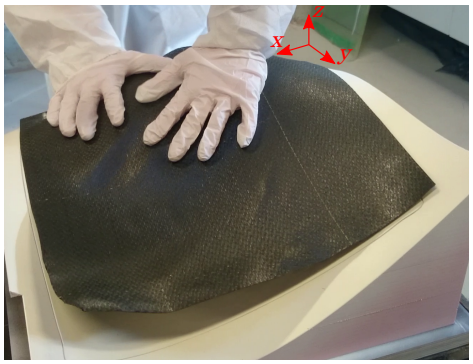

Fig. 1 Manual draping (hand layup) of carbon fiber weave onto double curved mold.

If the composite part possesses double curvature, an initially flat ply must deform in-plane in order to conform to such a double-curved surface. This is a fundamental theorem in differential geometry - consider also map projections from the spherical Earth to a flat paper, which can not be achieved without distortions (Callens and Zadpoor, 2018). The preferred way of in-plane deformation is generally that of the lowest resistance. For bi-axial fabrics the lowest resistance is mainly associated with *shearing* (Cao et al., 2008). Shearing entails that the tows of the fabric rotate but remain more or less undeformed. The concept of fabric shearing is illustrated in Fig. 2 for a woven fabric which is also primarily considered in this paper (In section 4.4, the applicability to non-crimp fabrics (NCF) is discussed). It should be noted that there exist a limit

C. Krogh
Department of Materials and Production, Aalborg University, Fibigerstraede 16, 9220 Aalborg, Denmark.
E-mail: ck@mp.aau.dk

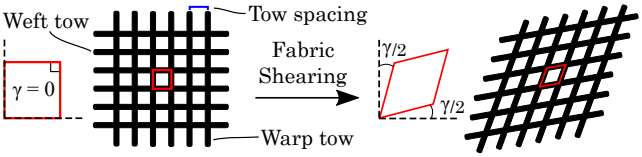

Fig. 2 Shearing of woven fabric: the tows rotate at their cross-over points in the weave. The shear angle is denoted γ.

on the amount of shear that can be achieved with a fabric, which is denoted the *locking angle*. Shearing beyond this angle is likely to result in wrinkling of the fabric, which will introduce defects in the final composite component. Other deformations than shearing also occur during draping and in general, the behavior of fabrics is very complex.

The rotation of the fiber tows makes it beneficial that the draping process is taken into account in the design phase of the composite part. That is, the specification of fiber orientations does not come with complete freedom, but is linked to the fabric architecture and mold curvature. At worst, a specified composite design might not even be manufacturable due to excessive shear. The accuracy of the structural analysis can hence be increased by modeling the actual fiber orientations compared to e.g. projected or idealized fiber orientations, which is used in many cases. The deviation from idealized fiber orientations will be exemplified in the result section of the paper.

To predict the final fiber orientations on the mold surface and thereby in the composite part, various draping models have been developed. Typically, a distinction is made between *mechanical* models and *kinematic* models. Mechanical models are commonly accomplished using dynamic nonlinear finite element analysis (FEA). Experimental data can feed into advanced material models and thereby a particular draping process can be simulated (Boisse et al., 2010; Dangora et al., 2015; Harrison, 2016; Krogh et al., 2019). The list of physical phenomena that can be accounted for with such models is long and so are the required CPU times.

Kinematic models, on the other hand, approximate the behavior of fabrics with basis in the high modulus of structural fibers and a fabric's limited resistance to shearing. Thus, it is assumed that the fabric *shear stiffness is zero* while the *fiber extensional stiffness is infinite*. These assumptions enable the modeling of the fabric as a grid of hinged or *pin-jointed* cells on the mold surface as proposed by Mack and Taylor (1956). They considered simple analytical surfaces, but the method has subsequently been developed numerically for arbitrary surfaces (Van West et al., 1990; Long, 1994). Other researchers have benchmarked the kinematic model predictions against experimental data (Laroche and Vu-Khanh, 1994; Wang et al., 1999). Commercial implementations are also available, e.g. FiberSim (Siemens Industry Software Inc, 2020), Composites Modeler for Abaqus/CAE (Dassault Systèmes, 2020) or Ansys ACP (Ansys Inc., 2020). Kinematic models will predict the draping pattern on the mold under simplified conditions but can not take the actual draping process into account. The advantage of kinematic models is their low CPU times, which greatly facilitate the use of optimization techniques.

In several works, a kinematic draping algorithm has been combined with optimization. Skordos et al. (2006) applied a commercial kinematic draping algorithm coupled with a genetic algorithm (GA) to simplify the manufacturing of a composite pilot helmet. The objective was to minimize the shear angles of the fabric on the mold by altering the initial conditions of the draping. Kaufmann et al. (2010) considered cost/weight optimization of a composite part and included a kinematic draping algorithm in their framework. The objectives considered were fiber angle deviation (difference between actual and nominal fiber orientation of a ply), the magnitude of the computed shear angles, and material waste. Kussmaul et al. (2019) studied optimization of patched laminates and incorporated a kinematic draping algorithm for predicting the draped pattern of the patches. A dedicated mechanical model enabled the incorporation of stiffness and strength criteria.

This paper describes a simple kinematic draping algorithm implemented in MATLAB, which is readily extended for optimization. The rest of the paper is organized as follows: Section 2 elaborates the algorithm behind kinematic draping. Section 3 describes the implementation in MATLAB and Section 4 presents various extensions, including optimization. Lastly, the paper concludes with a summary in Section 5. The code implementation is provided in Appendices A and B.

2 The kinematic draping algorithm

The starting point of the algorithm is the definition of the grid of cells, which represents the fabric as sketched in Fig. 3. Here, the cell edges represent fiber tows and the cell vertices, i.e. grid nodes, represent their cross-over points. The cell edge lengths, i.e. distance between nodes, is the discretization, d. Notice, though, that d does not have to be equal to the tow spacing. As previously mentioned, the fiber tows are assumed to be inextensible and the fabric shear stiffness is assumed to be zero. In addition, the tow bending stiffness is assumed to be zero and also tow slipping is neglected.

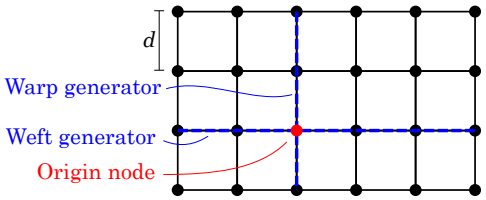

Fig. 3 The fabric modeled as a grid of pin-jointed cells. The red node is the chosen origin node through which the blue dashed generators pass.

These assumptions entail that the discretization distance must always be obeyed while the rotation at the nodes is free. Such a grid of pin-jointed cells work like an unconstrained mechanism. Forcing the nodes of the grid to be on the mold surface introduces some constraints, but to produce a unique solution, some initial conditions of the draping are required.

The first constraint is that an origin point on the mold surface and a corresponding origin node from the grid are coincident. The physical meaning of the origin point on the mold is the fabric's first contact point during draping. Next, the specification of an initial draping direction at the origin will serve to orient the fabric on the mold. To specify the remaining constraints, the traditional approach is to compute initial paths of the two fiber directions (warp and weft) - so-called *generators* which form a cross (Fig. 3). The generators must intersect at the origin point and follow the initial draping direction. By using the discretization d, the created generators can be populated with nodes on the mold. These nodes, representing the two initial fiber paths, will constrain the remaining draping pattern and thus produce a unique solution.

A simple approach to the creation of the generators is to use curves that arise from intersecting the mold surface with selected planes passing through the origin point. This approach is known as the *planar method* and is sketched in Fig. 4. The planar method has been proven to work well for rotationally symmetric shapes such as a hemisphere, but it fails to give realistic predictions for arbitrary mold shapes (Wang et al., 1999). In this case, better predictions are obtained by specifying the generators as *geodesic lines*.

2.1 Geodesic lines

A geodesic line can be considered as a generalization of a straight line on a curved surface. Mathematically, it is a curve whose component of curvature tangential to the surface (geodesic curvature) is zero. For a unit-speed curve, this is equivalent to the component of tangential acceleration being equal to zero (Pressley, 2010). Given

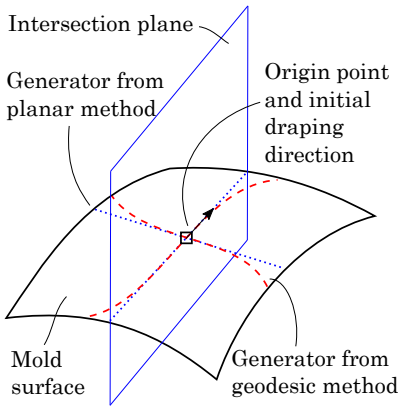

Fig. 4 Creation of generators as constraints for the kinematic draping algorithm.

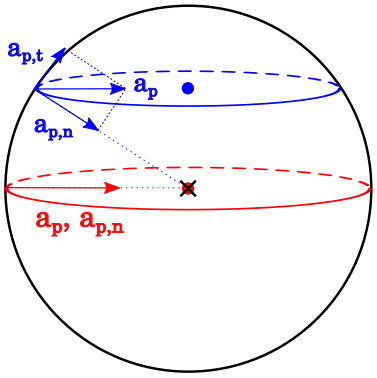

Fig. 5 Different circular curves on a sphere. The lower, red curve, i.e. equator, is a geodesic while the upper blue is not because the component of acceleration tangential to the surface, $\mathbf{a_{p,t}}$ is nonzero.

any two points on a double curved surface, the shortest surface conformant path between them is a geodesic line. For these reasons, geodesics are also considered as a more natural course of a fiber tow on a double-curved mold surface, see Fig. 4.

Consider for instance the surface of a sphere as sketched in Fig. 5. Circular curves on the surface can be produced by intersecting the sphere with a horizontal plane. From the study of particles undergoing uniform circular motion, it is known that the acceleration of the particle, $\mathbf{a_p}$, points to the center of the circle. Now, in order for the acceleration to only have a component normal to the sphere ($\mathbf{a_{p,n}}$), the acceleration vector must also point to the center of the sphere. This condition can be achieved by letting the intersecting plane pass through the center of the sphere (lower, red curve in Fig. 5). Such a circle on the sphere is called a *great circle*, which is thus a geodesic. The previous discussion also explains the initial success of the planar method on rotationally symmetric surfaces. Nonetheless, a geodesic line will in general produce a more realistic course for a generator.

2.2 Details of the presented algorithm

For the algorithm presented in this paper, a simplification is made with regard to the surface representation. The mold is modeled as a surface on the form $z = F(x, y)$, i.e. with a unique z coordinate for each x and y coordinate. In this way, molds with vertical sections can not be treated, but the chosen surface representation greatly simplifies the modeling.

The presented algorithm is also restricted to rectangular plies, such that the generators completely span the ply. This may be modified using parts of the presented theory but at the expense of a more complicated implementation.

Geodesic lines on arbitrary mold surfaces can be quite complex and are typically obtained by integration of a differential equation (Ramgulam and Potluri, 2007). In this paper, the generators are created in a more simple and intuitive manner. Instead of lines, cells are placed to span the ply. Emanating from the origin point, the cells are created - one by one - by minimizing the sum of shear angles in each cell. The cells will lie in a plane tangential to the surface. Minimizing the cell shear is equivalent to minimizing the distortion from right angles in the tangential plane. In this way, the generator cells will form a straight path tangential to the surface, and thereby minimize the tangential curvature, thus meeting the conditions of a geodesic. A validation of the method is presented in Section 4.

The draping algorithm follows the general description in the beginning of the section and can be divided into three steps which are also sketched in Fig. 6:

1. Placing the first pair of nodes based on the origin point and the initial draping direction.
2. Creating the generator cells, i.e. initial fiber paths.
3. Placing the remaining cells constrained by the generator cells.

To ensure that the nodes are always on the mold surface, only the x and y coordinates are specified such that the mold z coordinate is queried from F directly.

In Step 1, the origin node is placed on the origin point on the mold. Next, a second node is placed along the initial draping direction a distance d apart from the origin (Fig. 6 a)). This placement is achieved by solution of an equation in one variable using numerical optimization techniques.

In Step 2, the generator cells are created one by one. For each generator cell, two vertices are known while the remaining two vertices are unknown. This situation is sketched in Fig. 6 b) with the cell with three dashed edges: Vertices $\mathbf{V_1}$ and $\mathbf{V_2}$ are known from the previous cell and vertices $\mathbf{V_3}$ and $\mathbf{V_4}$ must be placed such that the sum of shear angles in the cell is minimized while

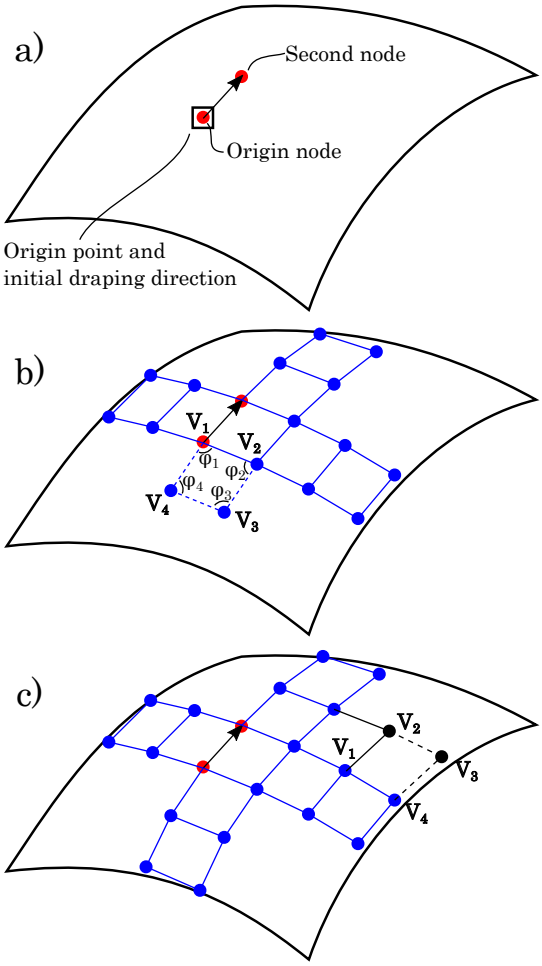

Fig. 6 The three steps in the draping algorithm: a) Step 1, red nodes, b) Step 2, blue nodes, and c) Step 3, black nodes.

the dashed cell edges have lengths d. The locations of the unknown vertices are defined relative to the known vertices (using only x and y coordinates):

$$\mathbf{V_3} = \mathbf{V_2} + \{x_1, y_1, F(x_1, y_1)\} \tag{1}$$
$$\mathbf{V_4} = \mathbf{V_1} + \{x_2, y_2, F(x_2, y_2)\} \tag{2}$$

The angles at the vertices, φ_i can be calculated as the angle between the vectors formed by the adjacent cell edges, e.g. $\varphi_4 = \angle \mathbf{V_1 V_4 V_3}$.

The four relative coordinates are assembled in the vector of design variables, $\mathbf{a} = \{x_1, y_1, x_2, y_2\}$. The optimization problem is formulated as follows:

$$\begin{aligned} \underset{\mathbf{a}}{\text{minimize}} \quad & \sum_{i=1}^{4} |\varphi_i - 90°| \\ \text{s.t.} \quad & ||\mathbf{V_3} - \mathbf{V_2}|| - d = 0 \\ & ||\mathbf{V_4} - \mathbf{V_3}|| - d = 0 \\ & ||\mathbf{V_1} - \mathbf{V_4}|| - d = 0 \end{aligned} \tag{3}$$

In Step 3, the remaining cells, which are kinematically constrained by the generator cells, are placed. Consider the situation in Fig. 6 c) and the cell with two dashed edges. In the cell, three vertices are known ($\mathbf{V_1}$, $\mathbf{V_2}$ and $\mathbf{V_4}$) while the last vertex ($\mathbf{V_3}$) is unknown. Again, using relative coordinates for $\mathbf{V_3}$ as in Eq. (1), the problem has two unknowns (x_1 and y_1) and the following two equations:

$$||\mathbf{V_3} - \mathbf{V_2}|| - d = 0 \tag{4}$$
$$||\mathbf{V_4} - \mathbf{V_3}|| - d = 0 \tag{5}$$

These are identical to the first and second constraint in the optimization problem in Step 2, i.e. Eq. (3). The system of equations for each cell can be solved numerically using optimization techniques. A suitable initial guess is chosen as the opposite cell edge, i.e. $\mathbf{V_4}$-$\mathbf{V_1}$.

3 MATLAB implementation

The MATLAB program is named *KinDrape* and can predict the draped pattern on a hemisphere. The code is found in Appendix A of the paper and relies on built-in functions and functionalities. These include the use of:

- The Optimization Toolbox (The MathWorks Inc., 2020) for nonlinear equation solving (Step 1 and 3) and optimization problems (Step 2). The former is achieved with the *fsolve* function using the default *Trust-Region-Dogleg* algorithm and the latter with the *fmincon* function using the *Active Set* algorithm. Both rely on finite-difference gradients.
- An *interpolation object* based on a point cloud (e.g. from a CAD file) or an *anonymous function* to represent the mold surface as a function $z = F(x, y)$. This representation enables easy querying of the z coordinate.
- Linear indexing, i.e. the use of a single index in a multi-dimensional array, for compact notation.
- Plotting of the mold surface using the *surf* function and plotting of the draped cells with shear angle color contours using the *patch* function.

KinDrape is called as a function as follows:

```
KinDrape(d,Grid,Org,Ang,OrgNode)
```

The input arguments are defined as:

- `d`: The grid discretization distance (see Fig. 3).
- `Grid`: Two-component vector with number of rows and columns in the grid (see Fig. 3).
- `Org`: Two-component vector with the origin point on the mold in x, y-coordinates (see Fig. 6 a)).

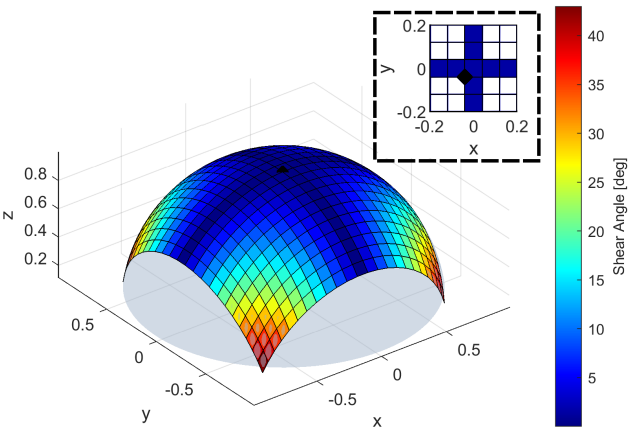

Fig. 7 Draping onto a hemisphere with the center cell located at the north pole. The zoom-in in the black dashed square shows the origin offset.

- `Ang`: Initial draping direction in degrees relative to the y-axis (see Fig. 6 a)).
- `OrgNode`: Two-component vector with row,col-index of the origin node on the ply (see Fig. 3).

The result in Fig. 7 is obtained by executing *KinDrape* as follows:

```
KinDrape(0.08,[24 24],-[0.04 0.04],0,[12 12]);
```

Notice that the origin is offset by half a discretization such that the center cell is centered in (0,0). This will produce a double-symmetric solution on the hemisphere.

The grid nodes on the mold are stored in a 3D array, `Node` (size: number of row nodes × number of column nodes × 3). The first two dimensions are the row,col-location in the grid and the pages/slices along the third dimension contain the x, y, z-coordinates. Data for plotting the colored cells is stored in the array `P` (size: number of cells × 4 × 4). The 1st dimension is the cell number (column-major order in the grid), the 2nd dimension is the vertex numbers 1-4 in that particular cell and the 3rd dimension has three coordinates and a shear angle for a particular vertex.

The main bookkeeping concept of the program is illustrated in Fig. 8. Here, the cross formed by the generators has been split up in four *arms* that span four *quadrants*. The idea is to define two propagation directions, `Dir1` and `Dir2`, that control the location in `Node` as well as the orientation of the cells. The directions rotate with the arm/quadrant number. In this way, in Step 2 (arms) vertex 1 and 2 are always known, while in Step 3 (quadrants), vertex 1, 2 and 4 are always known. Notice that `Dir2` is simply `Dir1` rotated 90 degrees.

The program KinDrape is structured as a main program (Appendix A, ll. 1-57) and four auxiliary func-

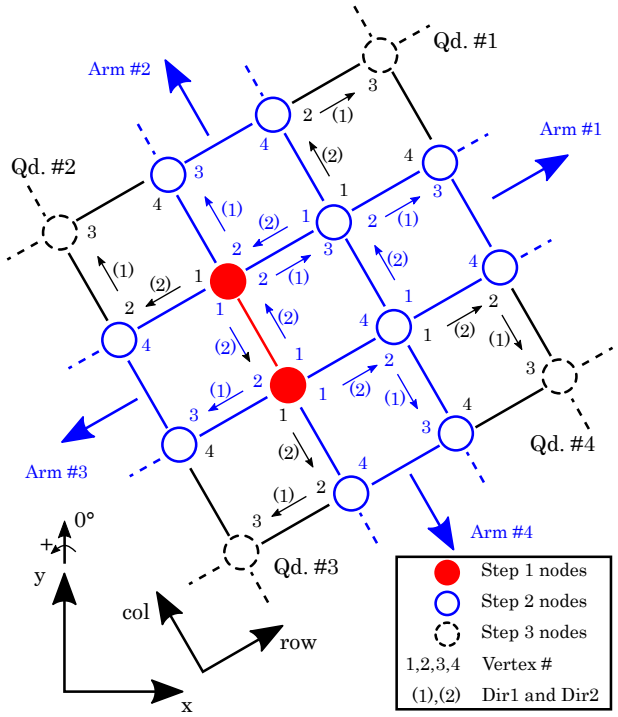

Fig. 8 Bookkeeping of cells in arms and quadrants in KinDrape.

tions (Appendix A, ll. 58-99). The auxiliary functions are:

- `CellIdx`: Given the vertex 1 row,col-index in `Node` of a cell, the function returns a 4 × 3 array of the linear indices of the vertices in which the rows correspond to the vertex number and the columns correspond to the coordinate number. The second output is the cell number used for storing data in the `P` array.
- `CellVertCoor`: For a cell, the function returns an array of all vertex coordinates (number of vertices × 3) based on the known vertices and the design variables in the equation solving/optimization.
- `DistFun`: returns the Euclidean distances between cell vertices depending on the step. This function is the equation function in Step 1 and 3 and the constraint function in Step 2 (NB: with *fmincon* the equality constraint must be the second output from the constraint function).
- `ShearFun`: Calculates the shear angles in a cell. First, four vector pairs `u` and `v` are computed such that they lie along the cell edges. Next, the angles between cell edges defined by vectors `u`, `v` are calculated using the expression $\varphi = \arctan(||u \times v||/u \bullet v)$. It is implemented in vectorized form using the 2-argument arctan function. The shear angle is obtained by subtraction of 90° and calculation of the absolute value, as in Eq. (3). The function can return the sum of shear angles (objective in Step 2), the vertex coordinates of a cell and the four shear angles of a cell.

The main program in Appendix A is elaborated in the following sections. It is recommended to execute the program in debug mode while going through the descriptions.

3.1 Mold Definition (App. A, ll. 2-5)

The hemisphere mold is defined through a *meshgrid* of spherical coordinates which is transformed to Cartesian coordinates using `sph2cart`. The values of `Theta` and `Phi` are chosen to avoid duplicate points and to avoid a vertical surface at the equator. Lastly, the interpolation object, `F`, is created.

3.2 Aux. variables and initialization (App. A, ll. 6-11)

The propagation directions of the cells are defined through the arrays `Dir1` and `Dir2` in row,col-coordinates. The arrays contain, respectively, the directions of the vertex 2-3 cell edge and the vertex 1-2 cell edge. Each row in the arrays correspond to an arm/quadrant. Further, options for *fsolve* and *fmincon* are defined and the arrays `Node` and `P` are initialized as *NaN*.

3.3 Step 1 (App. A, ll. 12-17)

In this step the origin node and a node defined by the initial draping direction are placed. First, linear indices for the vertices are retrieved using the `CellIdx` function. The origin node can readily be placed, while the 2nd node is found by solution of one equation in one variable contained in `DistFun`. The solution is achieved using *fsolve*.

3.4 Step 2 (App. A, ll. 18-34)

In this step, the generator cells are created. `GenStart` contains the starting row,col-index in `Node` for each arm and `nGenCell` contains the number of generator cells to create in each arm. A double loop iterates though each arm (i) and each cell (j). In this setup the iterator j is in fact a two-component vector with the row,col-index of vertex 1. For each cell, the linear cell indices are retrieved and bounds (`Bnd`) on the design space are defined as a box centered in the initial guess with a side length equal to the length of the initial guess. *fmincon*

is called and the new nodes and shear angles are stored in `Node` and `P`. Lastly the initial guess is updated as the previous solution.

3.5 Step 3 (App. A, ll. 35-50)

In this step the remaining constrained cells are placed. Similar to the previous step, the starting row,col-index for each quadrant is stored in `ConStart` and the number of cells to create (along rows and columns) in each quadrant is stored in `nConnCell`. An outer loop iterates through each quadrant (i) and a double loop in rows and columns iterates through each cell (j,k). For each cell, the linear cell indices are retrieved, the initial guess is computed based on the x, y-coordinates of vertex 4 and vertex 1, and lastly *fsolve* is called. Shear angles for the cell are calculated and the shear angles and new nodes are stored in `Node` and `P`.

3.6 Plotting (App. A, ll. 51-56)

In this section the following is plotted: the origin point, the mold surface in a gray-blue semi-transparent color, the draped cells colored by their shear angles and lastly a colorbar.

4 Extensions of the code

The presented code can be executed as is, and by manipulating the input parameters, much insight into the draping behavior of fabrics can be gained. Try for instance to move the origin away from the north pole, e.g. with:

```
KinDrape(0.08,[24 24],[0.0 -0.9],60,[3 3]);
```

This modification would in practice correspond to moving the fabric's first contact point when draping. The resulting draped pattern is shown in Fig. 9. Two great circles passing through the origin point have been drawn. Recall, that a great circle is a geodesic line on a sphere. As it can be seen, the generator cells follow the great circles, thus verifying the program. Consider also Fig. 10, in which the Abaqus' Composites Modeler solution to the same draping problem is shown. A high degree of similarity is observed.

Other extensions that can be readily implemented to enhance the functionality of the code are discussed in this section. These include new mold surfaces, the use of the code in an optimization framework, and introduction of pre-shear.

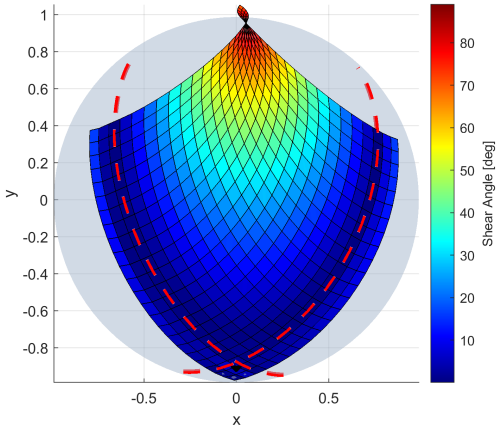

Fig. 9 Draping onto hemisphere (top view) with origin point moved away from the north pole. The red dashed lines are great circles, i.e. geodesics, intersecting at the origin point.

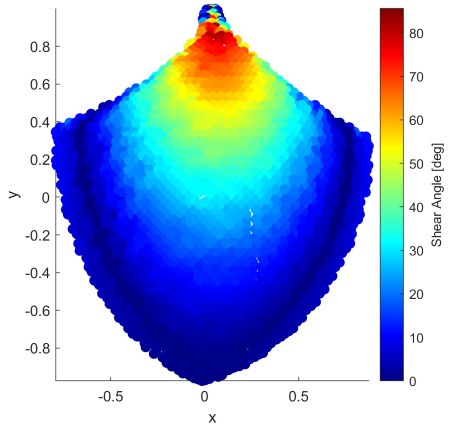

Fig. 10 Abaqus Composites Modeler solution to the draping problem in Fig. 9.

4.1 Mold definitions

Implementing another mold surface in the code can be achieved by modifying ll. 2-5 in Appendix A. If the mold surface has an analytical equation, it can be written as an anonymous function instead of an interpolation object as used previously with the hemisphere. Consider for instance the case of a single-curved parabolic cylinder. This example can demonstrate that no shearing occurs when draping onto single-curved surfaces:

```
[X,Y] = meshgrid(0:0.01:0.5);
F =@(x,y) 3*(x-0.25).^2; Z = F(X,Y);
```

After modifying the lines, the program can be called as follows, producing the result in Fig. 11:

```
KinDrape(0.022,[21 21],[0.25,0.25],5,[11 11]);
```

Also, consider the mold shown in Fig. 1, which was designed to generically represent typical double-curvatures in aerospace composite components (Krogh et al., 2019). This mold can be implemented as follows:

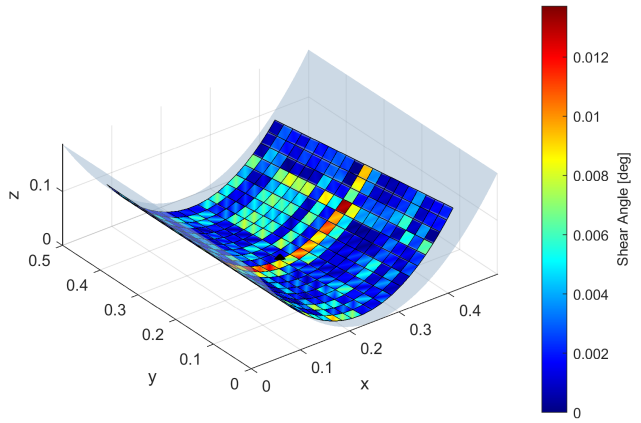

Fig. 11 Draping onto single-curved parabolic cylinder does not produce shear (notice the very low values of shear angles).

```
[X,Y] = meshgrid(0:0.01:0.5);
F =@(x,y) 1.004*x + 1.089*y - 3.667*x.^2 ...
-4.4*x.*y - 3.75*y.^2 + 3.086*x.^3 + ...
8.889*x.^2.*y + 4.321*y.^3; Z = F(X,Y);
```

The code can be called with the same input as with the single-curved parabolic cylinder example.

4.2 Optimization of draping parameters

As previously stated, the low computational expense of the kinematic draping algorithm favors its use in optimization studies on draping parameters, which is the topic of this section. The mold considered is the double-curved mold defined in Section 4.1 (Fig. 1) draped with a ply grid of `Grid = [21 21]` and a discretization of `d = 0.022`, i.e 440 mm x 440 mm.

Three design variables are used: The first design variable is the initial draping direction (`Ang`) and is bounded between [-10°; 10°]. The second and third design variables are the origin node row and column (`OrgNode`), defined relative to the center node of the ply (11, 11). They must be integers and are bounded between [-10; 9], i.e. covering the entire grid. The origin point on the mold is moved accordingly such that the ply remains inside the mold perimeter.

The objective concerns minimization of the shear angles, γ, and the angle deviations from 0° (y axis) of the warp fiber tows, ψ. The meaning of the latter is to have the warp fibers (cell edges along arm #2/ arm #4 direction) oriented as close as possible to a nominal fiber angle of 0°. The two field quantities are aggregated using a p-norm function, that gives high influence to the largest values. The two p-norms are then added with equal weighting because the two quantities have the same unit and the same order of magnitude:

$$\underset{\texttt{Ang,OrgNode}}{\text{minimize}} \quad \left(\sum_{i=1}^{N_\gamma} |\gamma_i|^p\right)^{1/p} + \left(\sum_{i=1}^{N_\psi} |\psi_i|^p\right)^{1/p} \tag{6}$$

Here N_γ and N_ψ are the number of components to aggregate for γ and ψ, respectively. The value of p is chosen to 12. The objective function is minimized taking the bounds on the design variables into account. A MATLAB script of 26 lines, which uses the `KinDrape` function can be found in Appendix B.

Two small modifications of `KinDrape` are necessary for its use in the minimization in Eq. (6). First, the double-curved mold defined in Section 4.1 must be implemented. Second, to avoid plotting in the numerous calls during the optimization, an extra input argument `Plt` can be added to the function: `KinDrape(__,Plt)`. The variable is logical (`true` / `false`) and by enclosing the plotting section of `KinDrape` (App. A, ll. 51-56) in an `if` statement, plotting can be conveniently suppressed:

```
if Plt
    figure; scatter3(Org(1),Org(2),...
    ...
    cb = colorbar; cb.Label.String = ...
end
```

The design space of the optimization problem includes several local minima and for this reason it was chosen to employ a zero-order optimization method, namely MATLAB's Genetic Algorithm, `ga`. This algorithm also handles integer design variables conveniently. The standard settings were used but they can easily be altered.

The optimization script is organized as the main program (App. B, ll. 1-10) and a function, `ObjFun`, returning, among others, the objective function for the optimizer. In the main program, MATLAB's random number generator is first reset to make the results reproducible (`rng('default')`). Next, input variables and optimization settings are defined. Using the `'MaxGenerations'` setting for `ga`, the duration of the optimization can be controlled. Finally, in the main program, the optimizer is called and afterwards the solution is plotted and maximum shear angles and warp fiber angle deviations are displayed.

In the objective function, `ObjFun`, the design variables are translated into input to `KinDrape` and the function is called. Shear angles are calculated based on the `P` array. The warp fiber angle deviations are projected to the xy-plane, i.e. with the z coordinate equal to zero. The warp fiber tows are represented by vectors constructed using the columns in the `Node` array. The

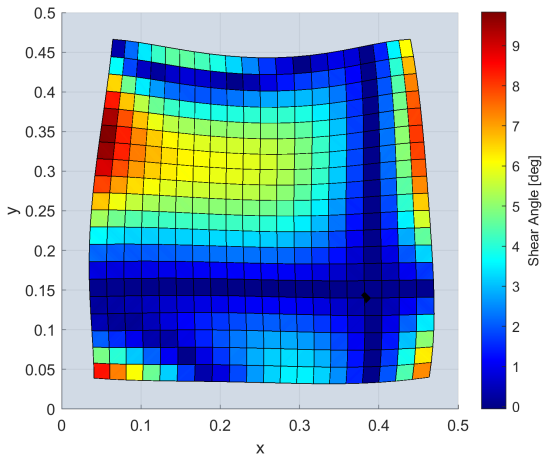

Fig. 12 Optimized draped pattern on double-curved mold with 4 generations. The initial draping direction is 1.0° and the starting node is (17, 6). The maximum shear angle is 9.7° and the maximum warp fiber angle deviation is 10.3°.

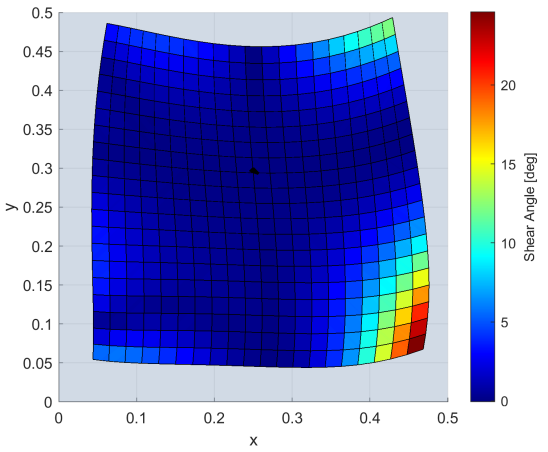

Fig. 13 Maximum shear angle draped pattern on double-curved mold with 4 generations. The initial draping direction is 5.1° and the starting node is (11, 13). The maximum shear angle is 24.6° and the maximum warp fiber angle deviation is 9.8°.

fiber angle deviations are calculated as the angles between the warp fiber tow vectors and unit vectors stored in the `NomVec` array on the form $[0, 1, 0]$ (along the third array dimension). Finally, the objective is calculated as the sum of the two p-norms.

Running the optimization with 4 generations produces the result in Fig. 12. As a reference, consider the worst case draped pattern. This pattern can be found by maximizing the shear angles, i.e. by placing a minus sign in front of `sum(abs(Shear).^p)^(1/p)` in l. 25, Appendix B. Running this optimization problem with 4 generations produces the result in Fig. 13. Thus, even for this simple mold, choosing different initial conditions for the draping can result in maximum shear angles between 9.7° and 24.6°. Reducing the shear will make the manufacturing step easier, but as previously stated, it can in some cases determine whether a composite layup is manufacturable or not. Regardless of the shear angles, the maximum warp fiber angle deviations from 0° for the two examples are around 10°, thus highlighting the importance of taking the actual fiber orientations into account in a structural analysis.

This simple example illustrates the principle behind a draping pattern optimization but the setup can easily be altered. The weighting between the shear angles and fiber angle deviations in the objective function, Eq. (6), could be changed. In such a setup, it could also be relevant to include a shear limit of the fabric as a constraint.

4.3 Introduction of pre-shear in the draping analysis

Pre-shear refers to the process of shearing the fabric before the draping is initiated. Thereby the initial angle between warp and weft tows will be different from 90°. Pre-shear can be executed in a number of ways, but in this context it is defined as a global quantity, i.e. as a pre-shear angle, γ_{pre}, between the generators at the origin of draping. In order to implement pre-shear in the draping analysis (code in Appendix A), it is necessary to work with signed shear angles in the calculations (shear can be either positive or negative depending on the direction the fiber tows rotate). The pre-shear is determined by the generators and when creating the generator cells, the objective is now to obtain shear angles in the cells, which are equivalent to the pre-shear angle, γ_{pre}. Thus, the objective function from Eq. (3) used for generator cell creation is modified to:

$$\underset{\mathbf{a}}{\text{minimize}} \quad \sum_{i=1}^{4} |\gamma_i - \gamma_{\text{pre}}| \tag{7}$$

Here, γ_i are the signed shear angles which are introduced in the following. To modify the code (note that line numbers refer to the original code in Appendix A),

1. Add an extra input argument to the `KinDrape` function, `PreShear` (pre-shear angle in degrees): `KinDrape(__,PreShear)`.
2. Add two extra input arguments to the `ShearFun` function (in the function definition and at the three function calls): `PreShear` and `i` (loop iterator): `ShearFun(__,PreShear,i)`. The latter is necessary to correctly compute the signed shear angles depending on the arm/quadrant number.
3. Remove the `abs` function from l. 97 and instead, post multiply the parenthesis by `.*[1 -1 1 -1]*(-1)^i`. The factor `(-1)^i` will change the sign depending on the arm / quadrant number and `[1 -1 1 -1]` contains the bookkeeping for angle signs within a cell.

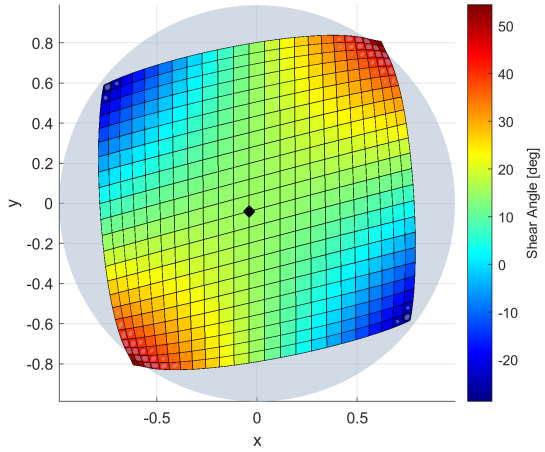

Fig. 14 Top view of hemisphere drape with a pre-shear of 15°. Notice that the shear angles are signed.

4. To change the objective function, replace l. 98 with `Obj = sum(abs(Shear - PreShear));`.

Executing the hemisphere-example from Fig. 7 with a pre-shear angle of 15°, produces the result in Fig. 14. The previous modifications will also make the plotted shear angles become signed. If the absolute values are preferred, add the following line to the end of the `ShearFun` function: `Shear = abs(Shear);`.

Pre-shear can be relevant if e.g. a certain mold geometry results in more positive than negative shear. In this case, the fabric can be pre-sheared in a negative direction, thus creating a larger reserve of absolute shear angles. Pre-shear can also be an aid to obtain some specified fiber orientations in certain locations on the mold. For these reasons, it could also be relevant to include the pre-shear as a design variable in the optimization of draping parameters presented in Section 4.2. A final point to note about the pre-shear is, that for added robustness, the pre-shear angle could be reflected in the initial guess, `a_0`, used for the creation of the first generator cells in l. 22 in Appendix A.

4.4 Further extensions

This section discusses some additional, advanced extensions and modifications. The presented implementation considers draping of rectangular plies. For some applications it is beneficial to simply fill the entire mold with fabric and afterwards export the *flat pattern*, i.e. the 2D contour used for cutting the non-draped ply. The extension can be made to the code, but the possibility of the generators not spanning the entire ply must be considered, i.e. with some concave boundaries. Here, an energy measure can be considered to complete the draping (Wang et al., 1999).

The algorithm was developed for woven fabrics, but can be applied to non-crimp fabrics (NCF). For unidirectional (UD) materials the fabric cells can be modeled as undergoing simple shear (Lim and Ramakrishna, 2002; Fengler et al., 2018).

For biaxial NCF, the applicability depends on the stitching pattern. Certain stitching patterns will in fact make the NCF behavior match that of a woven fabric. This entails that the tows exhibit minimal slip relative to each other and that the shear characteristic is not affected considerably (Boisse et al., 2017). In these cases the presented algorithm can be applied. Some stitching patterns will result in the fabric having a different characteristic in positive and negative shear. Because the generators completely define the draping pattern, a solution with geodesics might not be the minimum energy configuration. The remedy proposed in the literature is to approach the second and third step in an iterative manner by employing the fabric shear energy, see e.g. Bergsma (1995) and Long et al. (2000).

The efficiency of the code was not a target during the development but rather the simplicity and readability. To this end, some performance improvements can be pointed out. Consider for instance Step 3 in which two equations in two unknowns are solved for each cell. Another approach is to set up two spheres with centers located in vertex 2 and 4, respectively and with radii equal to the discretization. The two spheres will intersect in a circle and the problem of locating vertex 3 thus reduces to finding the intersection between the intersection circle and the mold (Robertson et al., 1981; Van West et al., 1990). A version of the code with this modification is available on the repository at doi.org/10.5281/zenodo.4316860.

The use of built-in MATLAB functions greatly simplifies the code although some are known to involve considerable overhead. Another relevant built-in feature, is the possibility of parallelization of the draping code. Because the four generator arms and quadrants can be computed independently of each other, the computations could be divided between four *MATLAB Workers*. Likewise, the genetic algorithm optimizer can also make use of parallelization, i.e. by invoking the option `'UseParallel',true`.

5 Summary

This paper has presented a simple implementation of a kinematic draping algorithm in MATLAB. The simplicity of the code is believed to facilitate its use in engineering education, bringing the concept of composite draping and its modeling into a more tangible context. Draping effects such as changed fiber orientations and

shear can be modeled and, as demonstrated, easily be subjected to optimization in regard to manufacturing process parameters. The use of the draping model in a structural stress analysis is now straightforward which will enhance the fidelity of the composite design phase.

Declarations

Funding: The work presented in the paper took place as part of the MADEBLADES project funded by the Energy Technology Development and Demonstration Program, Grant no. 64019-0514.

Conflicts of interest/Competing interests: The authors have no conflicts of interest to disclose.

Code availability/Replication of results: The code is available in the appendices of the paper and on the following repository in the latest version: doi.org/10.5281/zenodo.4316860. A Python implementation is also available.

Authors' contributions: The concept and initial code development was carried out by C. Krogh. All authors contributed to the development of the scope of the paper and the code. The first draft of the manuscript was written by C. Krogh. All authors read and approved the final manuscript.

References

Ansys Inc (2020) Composite materials simulation and failure analysis. URL https://www.ansys.com/products/structures/composite-materials

Bergsma OK (1995) Three Dimensional Simulation of Fabric Draping. Phd thesis, Delft University of Technology

Boisse P, Aimène Y, Dogui A, Dridi S, Gatouillat S, Hamila N, Khan MA, Mabrouki T, Morestin F, Vidal-Sallé E (2010) Hypoelastic, hyperelastic, discrete and semi-discrete approaches for textile composite reinforcement forming. International Journal of Material Forming 3(SUPPL. 2):1229–1240, DOI 10.1007/s12289-009-0664-9

Boisse P, Hamila N, Guzman-Maldonado E, Madeo A, Hivet G, Dell'Isola F (2017) The bias-extension test for the analysis of in-plane shear properties of textile composite reinforcements and prepregs: a review. International Journal of Material Forming 10(4):473–492, DOI 10.1007/s12289-016-1294-7

Callens SJ, Zadpoor AA (2018) From flat sheets to curved geometries: Origami and kirigami approaches. Materials Today 21(3):241–264, DOI 10.1016/j.mattod.2017.10.004

Cao J, Akkerman R, Boisse P, Chen J, Cheng HS, de Graaf EF, Gorczyca JL, Harrison P, Hivet G, Launay J, Lee W, Liu L, Lomov SV, Long A, de Luycker E, Morestin F, Padvoiskis J, Peng X, Sherwood JA, Stoilova T, Tao X, Verpoest I, Willems A, Wiggers J, Yu T, Zhu B (2008) Characterization of mechanical behavior of woven fabrics: Experimental methods and benchmark results. Composites Part A: Applied Science and Manufacturing 39(6):1037–1053, DOI 10.1016/j.compositesa.2008.02.016

Dangora LM, Mitchell CJ, Sherwood JA (2015) Predictive model for the detection of out-of-plane defects formed during textile-composite manufacture. Composites Part A: Applied Science and Manufacturing 78:102–112, DOI 10.1016/j.compositesa.2015.07.011

Dassault Systèmes (2020) Composites modeler for abaqus/cae. URL https://www.3ds.com/products-services/simulia/products/abaqus/add-ons/composites-modeler-for-abaquscae/

Fengler B, Kärger L, Henning F, Hrymak A (2018) Multi-Objective Patch Optimization with Integrated Kinematic Draping Simulation for Continuous–Discontinuous Fiber-Reinforced Composite Structures. Journal of Composites Science 2(2):22, DOI 10.3390/jcs2020022

Harrison P (2016) Modelling the forming mechanics of engineering fabrics using a mutually constrained pantographic beam and membrane mesh. Composites Part A: Applied Science and Manufacturing 81:145–157, DOI 10.1016/j.compositesa.2015.11.005

Kaufmann M, Zenkert D, Åkermo M (2010) Cost/weight optimization of composite prepreg structures for best draping strategy. Composites Part A: Applied Science and Manufacturing 41(4):464–472, DOI 10.1016/j.compositesa.2009.11.012

Krogh C, Glud JA, Jakobsen J (2019) Modeling the robotic manipulation of woven carbon fiber prepreg plies onto double curved molds: A path-dependent problem. Journal of Composite Materials 53(15):2149–2164, DOI 10.1177/0021998318822722

Kussmaul R, Jónasson JG, Zogg M, Ermanni P (2019) A novel computational framework for structural optimization with patched laminates. Structural and Multidisciplinary Optimization 60(5):2073–2091, DOI 10.1007/s00158-019-02311-w

Laroche D, Vu-Khanh T (1994) Forming of Woven Fabric Composites. Journal of Composite Materials 28(18):1825–1839, DOI 10.1177/002199839402801805

Lim TC, Ramakrishna S (2002) Modelling of composite sheet forming: A review. Composites - Part A: Applied Science and Manufacturing 33(4):515–537, DOI 10.1016/S1359-835X(01)00138-5

Long A (1994) Preform design for liquid moulding processes. PhD thesis, University of Nottingham

Long A, Souter BJ, Robitaille F (2000) A Fabric Mechanics Approach to Draping of Woven and Non-Crimp Reinforcements. In: Proceedings-American Society for Composites, pp 76–83

Mack C, Taylor HM (1956) The Fitting of Woven Cloth to Surfaces. Journal of the Textile Institute Transactions 47(9):T477–T488, DOI 10.1080/19447027.1956.10750433

Pressley A (2010) Elementary Differential Geometry, 2nd edn. Springer London

Ramgulam RB, Potluri P (2007) A differential geometry approach to forming simulation of biaxial preforms. In: ICCM International Conferences on Composite Materials

Robertson RE, Hsiue ES, Sickafus EN, Yeh GS (1981) Fiber rearrangements during the molding of continuous fiber composites. I. Flat cloth to a hemisphere. Polymer Composites 2(3):126–131, DOI 10.1002/pc.750020309

Siemens Industry Software Inc (2020) Fibersim. URL https://www.plm.automation.siemens.com/global/en/products/nx/fibersim.htmll

Skordos AA, Sutcliffe MPF, Klintworth JW, Adolfsson P (2006) Multi-objective optimisation of woven composite draping using genetic algorithms. In: 27th International Conference SAMPE EUROPE

The MathWorks Inc (2020) Optimization toolbox. URL https://www.mathworks.com/products/optimization.html

Van West BP, Pipes RB, Keefe M (1990) A simulation of the draping of bidirectional fabrics over arbitrary surfaces. Journal of the Textile Institute 81(4):448–460, DOI 10.1080/00405009008658722

Wang J, Paton R, Page JR (1999) Draping of woven fabric preforms and prepregs for production of polymer composite components. Composites Part A: Applied Science and Manufacturing 30(6):757–765, DOI 10.1016/S1359-835X(98)00187-0

A KinDrape code

```
function [Node,P] = KinDrape(d,Grid,Org,Ang,OrgNode)
%% Mold definition: Hemisphere
[Theta,Phi] = meshgrid(linspace(0,2*pi,100),linspace(pi/20,pi/2-1e-10,50));
[X,Y,Z] = sph2cart(Theta,Phi,1);
F = scatteredInterpolant(X(:),Y(:),Z(:),'linear','linear');
%% Auxiliary variables, solver settings and initialization of Node and P
Dir1 = [1 0 ; 0 1 ; -1 0 ; 0 -1]; Dir2 = [-Dir1(:,2) Dir1(:,1)];
Opt1 = optimoptions(@fsolve,'Display','off');
Opt2 = optimoptions(@fmincon,'Algorithm','active-set','Display','notify'...
```

```
    ,'MaxFunctionEvaluations',5e3);
Node = NaN([Grid 3]); P = NaN(prod(Grid-1),4,4);
%% Step 1: Place org. node (1) and node (2) defined by ini. drape angle
% Define linear indices for cell, place 1st node and solve for 2nd node
Idx = CellIdx(Grid,OrgNode(1),OrgNode(2),Dir1,Dir2,1);
Node(Idx(1,:)) = [Org(1), Org(2), F(Org(1), Org(2))];
a_sol = fsolve(@(a)DistFun(a,Node(Idx(1,:)),F,d,Ang),3/4*d,Opt1);
Node(Idx(1:2,:)) = CellVertCoor(a_sol,Node(Idx(1,:)),F,Ang);
%% Step 2: Place generator cells (initial cells) while minimizing shear
GenStart = OrgNode + [0 0 ; 1 1 ; 0 1 ; 0 0];
nGenCell = [Grid-OrgNode-[0 1]  OrgNode-1];
for i = 1:4
    a_0 = repmat(3/4*d*[cosd(Ang+(i-1)*90) sind(Ang+(i-1)*90)],1,2);
    for j = GenStart(i,:)' + (0:nGenCell(i)-1).*Dir1(i,:)'
        % Get cell idx and def. solver input. Call fmincon, assign solution
        [Idx, CellNo] = CellIdx(Grid,j(1),j(2),Dir1,Dir2,i);
        Bnd = a_0 + 1/2*norm(a_0(1:2))*[-1 -1 -1 -1; 1 1 1 1];
        a_sol = fmincon(@(a)ShearFun(a,Node(Idx),F),a_0,[],[],[],[],...
            Bnd(1,:),Bnd(2,:),@(a)DistFun(a,Node(Idx),F,d,[]),Opt2);
        [~,Node(Idx),Shear] = ShearFun(a_sol,Node(Idx),F);
        % Put current cell coord. and shear in P array and update a_0
        P(CellNo,1:4,1:4) = [Node(Idx) Shear'];
        a_0 = a_sol;
    end
end
%% Step 3: Place remaining, constrained cells
ConStart = OrgNode + [1 1 ; 0 1 ; 0 0 ; 1 0];
nConCell = nGenCell([1 2 ; 3 2 ; 3 4 ; 1 4]) - [1 0 ; 0 0 ; 0 0 ; 1 0];
for i = 1:4
    for j = ConStart(i,1) + (0:nConCell(i,1)-1)*(Dir1(i,1)+Dir2(i,1))
        for k = ConStart(i,2) + (0:nConCell(i,2)-1)*(Dir1(i,2)+Dir2(i,2))
            % Get cell idx and def. solver input. Call fsolve, assign sol.
            [Idx, CellNo] = CellIdx(Grid,j,k,Dir1,Dir2,i);
            a_0 = Node(Idx(4,1:2)) - Node(Idx(1,1:2));
            a_sol = fsolve(@(a)DistFun(a,Node(Idx),F,d,[]),a_0,Opt1);
            [~,Node(Idx),Shear] = ShearFun(a_sol,Node(Idx),F);
            % Put current cell coord. and shear in P array
            P(CellNo,1:4,1:4) = [Node(Idx) Shear'];
        end
    end
end
%% Plot
figure; scatter3(Org(1),Org(2),F(Org(1),Org(2)),'kx','LineWidth',5);
hold on; axis('equal','tight'); xlabel('x'); ylabel('y'); zlabel('z');
surf(X,Y,Z,0,'EdgeColor','none','FaceColor',[0.6,0.7,0.8],'FaceAlpha',0.5);
patch(P(:,:,1)',P(:,:,2)',P(:,:,3)',P(:,:,4)');
cb = colorbar; cb.Label.String = 'Shear Angle [deg]'; colormap('jet');
end
%% Aux. functions
function [Idx, CellNo] = CellIdx(Grid,Row,Col,Dir1,Dir2,No)
% For a cell, return linear ind. of vert. in Node (Idx) and # in P (CellNo)
Rows = Row + [0 Dir2(No,1) Dir1(No,1)+Dir2(No,1) Dir1(No,1)]';
Cols = Col + [0 Dir2(No,2) Dir1(No,2)+Dir2(No,2) Dir1(No,2)]';
Idx = Rows + (Cols-1)*Grid(1) + ((1:3)-1)*Grid(1)*Grid(2);
CellNo = Rows(No) + (Cols(No)-1)*(Grid(1)-1);
end
function Vert = CellVertCoor(a,Vert,F,Ang)
% Calculte unknown vertices in cell depending on length of design var. a
```

```
if length(a) == 1 % Step 1 (second node rel. to origin node)
    Vert(2,1:2) = Vert(1,1:2) + a*[cosd(Ang+90) sind(Ang+90)];
    Vert(2,3) = F(Vert(2,1),Vert(2,2));
elseif length(a) == 4 % Step 2 (Vert 3 rel. to 2 and Vert 4 rel. to 1)
    Vert(3,:) = [Vert(2,1:2)+a(1:2) F(Vert(2,1)+a(1),Vert(2,2)+a(2))];
    Vert(4,:) = [Vert(1,1:2)+a(3:4) F(Vert(1,1)+a(3),Vert(1,2)+a(4))];
elseif length(a) == 2 % Step 3 (Vert 3 rel. to 2)
    Vert(3,:) = [Vert(2,1:2)+a(1:2) F(Vert(2,1)+a(1),Vert(2,2)+a(2))];
end
end
function [Out1, Out2] = DistFun(a,Vert,F,d,Ang)
% Get cell vertices and return distance between vertices depending on step
Vert = CellVertCoor(a,Vert,F,Ang);
if length(a) == 1 % Step 1 (1 x dist)
    Out1 = norm(Vert(2,:)-Vert(1,:)) - d;
elseif length(a) == 4 % Step 2 ([] (inequality constr.) and 3 x dist)
    Out1 = [];
    Out2 = vecnorm(Vert([3,4,1],:)-Vert([2,3,4],:),2,2)' - d;
elseif length(a) == 2 % Step 3 (2 x dist.)
    Out1 = vecnorm(Vert([3,4],:)-Vert([2,3],:),2,2)' - d;
end
end
function [Obj, Vert, Shear] = ShearFun(a,Vert,F)
% Get cell vertices and calc. shear angles. Return shear ang. sum (obj. in
% step 2), vertex coordinates and vector of shear angles (for P array)
Vert = CellVertCoor(a,Vert,F);
% Calculate shear angles using cell edge vectors u and v
u = Vert([2 3 4 1],:)' - Vert';
v = Vert([4 1 2 3],:)' - Vert';
Shear = abs(atan2d(vecnorm(cross(u,v),2,1),dot(u,v))-90);
Obj = sum(Shear);
end
```

B Optimization script code

```
clc; clear; close all; rng('default');
% Define variables, options and call ga function
d = 0.022; Grid = [21 21]; Org0 = [0.25 0.25]; OrgNode0 = [11 11]; p = 12;
Opt3 = optimoptions(@ga,'Display','iter','MaxGenerations',4);
nDesVar = 3; lb = [-10 -10 -10]; ub = [10 9 9]; IntegerCon = [2 3];
x_opt = ga(@(x)ObjFun(x,d,Grid,Org0,OrgNode0,p,false),...
    nDesVar,[],[],[],[],lb,ub,[],IntegerCon,Opt3);
% Evaluate function with x_opt. Plot and display result
[~,Node,Shear,AngDev] = ObjFun(x_opt,d,Grid,Org0,OrgNode0,p,true);
fprintf('\nMax shear: %g, Max angle dev.: %g \n',max(Shear),max(AngDev(:)))
function [Obj,Node,Shear,AngDev] = ObjFun(x,d,Grid,Org0,OrgNode0,p,Plt)
% Obj. fun. that evaluates KinDrape and calculates shear and angle dev.
% Create input variables to KinDrape based on design variables
Ang = x(1);
OrgNode = OrgNode0 + [x(2) x(3)];
Org = Org0 + d*[x(2) x(3)];
[Node,P] = KinDrape(d,Grid,Org,Ang,OrgNode,Plt);
% Calculate shear angles (gamma) as mean of each cell
Shear = mean(P(:,:,4),2);
% Calculate the warp fiber angle deviations (psi)
WarpVec = diff(Node(:,:,1:2),1,2); WarpVec(:,:,3) = 0.0;
```

```
NomVec = reshape([0 1 0],1,1,3) .* ones(size(WarpVec));
AngDev = atan2d(vecnorm(cross(WarpVec,NomVec),2,3),dot(WarpVec,NomVec,3));
% Calculate objective as sum of p-norms
Obj = sum(abs(Shear).^p)^(1/p) + sum(abs(AngDev(:)).^p)^(1/p);
end
```