Highly efficient nonlinear structural analysis of origami assemblages using the MERLIN2 software

Ke Liu, Glaucio H. Paulino

Abstract: *The bar-and-hinge model is a highly efficient structural analysis approach for origami systems based on reduced-order modeling, which predicts their global mechanical behaviour surprisingly well. We implemented this method in MERLIN, a Matlab software for structural analysis of origami assemblages. Here we present MERLIN2, an extended version of the software, offering several new capabilities, such as implementation of a new triangulation scheme that allows for consideration of polygonal panels, convenient import/export capabilities, and displacement loading. The Matlab code and associated implementation are explained in detail, and a numerical example is presented to illustrate the MERLIN2 capabilities.*

1 Introduction

Emerging concepts associated to origami engineering [Schenk and Guest 13, Filipov et al. 15, Lang et al. 15] pose a challenge on our understanding of origami mechanics. In practical applications, engineers need to know how the origami structural system interacts with environment and responds to human control. Although origami folding is often idealized as rigid origami, concerning only geometry and kinematics; in practice, the folding process exhibits complicated behaviour, beyond simple folding, which cannot be explained by geometry alone, owing to the flexibility of panels. Modeling of origami structures by means of shell finite-elements (FE) provides high-resolution analysis, but also requires a time-consuming cycle for both modeling and computing, leading to unnecessary cost and effort especially in the preliminary design stage [Ramm and Wall 04, Ota et al. 16]. Thus, there is a need for a simple and effective analysis approach that fills the gap between the overly simplified rigid origami simulations and the detailed and expensive full-scale FE analyses, while shedding light on the essence of origami mechanics. This leads us to the bar-and-hinge approach [Filipov et al. 17, Liu and Paulino 17].

The essence of the bar-and-hinge approach [Schenk and Guest 11,Filipov et al. 17] is the simplification of the kinematics of origami assemblages, whose rationale lies with the fact that admissible deformations of origami structures are strongly confined by geometry. Deformed thin panels in origami are likely to display concentrated bending curvatures along diagonals due to the singular ridge effect [Witten 07]. The bar-and-hinge model represents the kinematic space of an origami

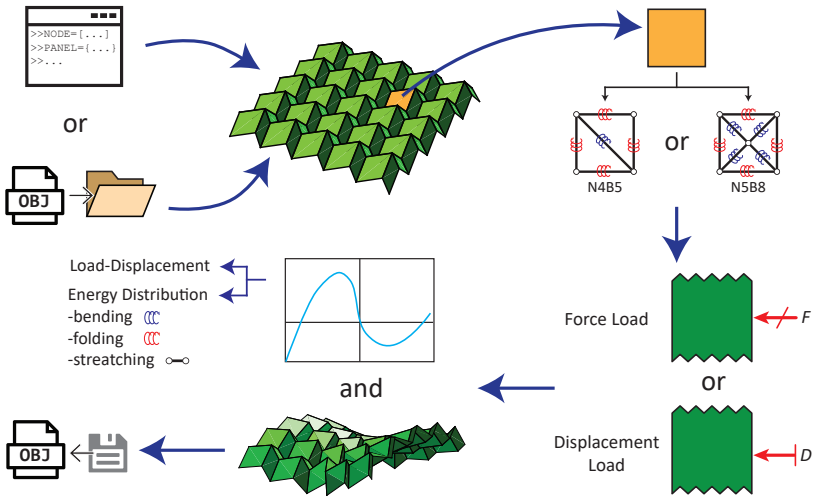

Figure 1: *A typical workflow of MERLIN2.*

structure with a bar (or truss) frame associated with constrained out-of-plane rotations, which successfully captures three fundamental deformation modes in origami: (in-plane) stretching, (out-of-plane) crease folding, and (out-of-plane) panel bending. Bars are placed along straight fold lines, and across panels for in-plane stiffness. The rotational hinges are along bars connecting panels to model folding of creases, and along bars across panels to model bending of panels. With only a few degrees of freedom, the reduced order bar-and-hinge model predicts surprisingly well the overall mechanical behaviour of origami structures.

Recently, we developed a nonlinear formulation for origami structural analysis using bar-and-hinge models to enable large deformation. We implemented the formulation in the software MERLIN, a dedicated open-source Matlab code for structural analysis of origami assemblages, which simulates the entire deformation process of an origami structure subject to prescribed applied forces. *Here we present MERLIN2, an extended version of MERLIN with several new capabilities, such as implementation of a new N5B8 super-element [Filipov et al. 17], consideration of polygonal panels, import/export capabilities with wavefront OBJ format, and displacement loading.*

2 Nonlinear elastic formulation for a general bar-and-hinge model

Here, we briefly describe the nonlinear elastic formulation of the bar-and-hinge method [Liu and Paulino 17]. We consider a discretized origami assemblage as an elastic system. The total potential energy (Π) of the system has contributions from the bars (U_S), bending hinges (U_B) and folding hinges (U_F). The total potential energy of the system can be written as:

$$\Pi(\mathbf{u}) = U_S(\mathbf{u}) + U_B(\mathbf{u}) + U_F(\mathbf{u}) - \mathbf{f}^T\mathbf{u}, \tag{1}$$

where $\mathbf{f}$ is the externally applied load, and all the other energy terms are nonlinear functions of the nodal displacements $\mathbf{u}$. Equilibrium is obtained when Π is local stationary, and therefore the equilibrium equation and the finite element matrices can be derived as [Filipov et al. 17, Liu and Paulino 17]:

$$\mathbf{T}(\mathbf{u}) = \mathbf{T}_S(\mathbf{u}) + \mathbf{T}_B(\mathbf{u}) + \mathbf{T}_F(\mathbf{u}) - \mathbf{f} = \mathbf{0}, \tag{2}$$

$$\mathbf{K}(\mathbf{u}) = \mathbf{K}_S(\mathbf{u}) + \mathbf{K}_B(\mathbf{u}) + \mathbf{K}_F(\mathbf{u}), \tag{3}$$

where:

$$\mathbf{T}_S(\mathbf{u}) = \frac{\partial U_S(\mathbf{u})}{\partial \mathbf{u}},\ \mathbf{T}_B(\mathbf{u}) = \frac{\partial U_B(\mathbf{u})}{\partial \mathbf{u}},\ \mathbf{T}_F(\mathbf{u}) = \frac{\partial U_F(\mathbf{u})}{\partial \mathbf{u}}, \tag{4}$$

and

$$\mathbf{K}_S(\mathbf{u}) = \frac{\partial^2 U_S(\mathbf{u})}{\partial \mathbf{u}^2},\ \mathbf{K}_B(\mathbf{u}) = \frac{\partial^2 U_B(\mathbf{u})}{\partial \mathbf{u}^2},\ \mathbf{K}_F(\mathbf{u}) = \frac{\partial^2 U_F(\mathbf{u})}{\partial \mathbf{u}^2}. \tag{5}$$

The system equilibrium and tangent stiffness are summations of elemental contributions, which is defined through an elastic constitutive model for each element in the system.

2.1 Bar elements

For each bar element, we define its stored energy as:

$$U_S^i = ALW(E_{xx}) \tag{6}$$

where A denotes the member area, L denotes the member length, and W is the energy density as a function of the one dimensional Green-Lagrange strain E_{xx}. The energy conjugate 2nd Piola-Kirchhoff (PK) stress and the tangent modulus are defined as

$$S_{xx} = \frac{\partial W}{\partial E_{xx}}, \quad \text{and} \quad C = \frac{\partial S_{xx}}{\partial E_{xx}}. \tag{7}$$

Thus for a single bar element, its contributions to the system equilibrium and tangent stiffness become

$$\mathbf{T}_S^i = ALS_{xx}\frac{\partial E_{xx}}{\partial \mathbf{u}}, \text{ and } \mathbf{K}_S^i = AL\left(C\frac{\partial E_{xx}}{\partial \mathbf{u}} \otimes \frac{\partial E_{xx}}{\partial \mathbf{u}} + S_{xx}\frac{\partial^2 E_{xx}}{\partial \mathbf{u}^2}\right). \tag{8}$$

Given that the Green-Lagrange strain is a function of the nodal displacements [Wriggers 08, Liu and Paulino 17], we are able to complete the components contributed by a single bar element.

2.2 Special rotational spring elements

Folding and bending deformations are both represented by rotational springs in a bar-and-hinge model (see [Liu and Paulino 17]). Conceptually, they are kinematically the same, but may have different constitutive behaviors. Examples can be found in Fig. 2 and 3. A rotational spring element consists of 4 nodes, 5 bars, and 1 dihedral angle between two triangles. Borrowing ideas from standard nonlinear elasticity, we assume a stored energy function ψ for each rotational spring to describe its constitutive behavior, which is a function of the rotation angle (θ):

$$U_B = \psi_B(\theta), \quad \text{or } U_F = \psi_F(\theta), \tag{9}$$

where ψ_B is the stored energy function of a bending hinge, while ψ_F is the stored energy function of a folding hinge. Taking a bending hinge as example, we can define the resistant moment M and tangent rotational modulus k as:

$$M = \frac{d\psi_B(\theta)}{d\theta}, \quad k = \frac{dM}{d\theta}. \tag{10}$$

We define the rotation angle $\theta \in [0, 2\pi)$ using absolute measurements such that $\theta = \pi$ when the panel is flat. We call θ_0 the neutral angle of a rotational spring if $M(\theta_0) = 0$. Following this convention, a linear elastic rotational spring can be defined by:

$$\psi_B(\theta) = \frac{1}{2}LK(\theta - \theta_0)^2, \quad M = LK(\theta - \theta_0), \quad k = LK, \tag{11}$$

where L denotes the undeformed length of the rotational hinge and K is the rotational modulus per unit length along the hinge. To differentiate bending and folding hinges, we can assign different values for K.

Thus for a bending rotational spring element, its contributions to the system equilibrium and tangent stiffness become

$$\mathbf{T}_B^i = M\frac{d\theta}{d\mathbf{u}}, \quad \text{and} \quad \mathbf{K}_B^i = k\frac{d\theta}{d\mathbf{u}} \otimes \frac{d\theta}{d\mathbf{u}} + M\frac{d^2\theta}{d\mathbf{u}^2}. \tag{12}$$

The same formulation also applies to folding rotational spring elements. Given that the the rotation angle is completely defined from the displacements and original coordinates of the 4 associated nodes, we are able to complete the components contributed by a single rotational spring element which either represents a bending hinge or folding hinge.

3 Discretization schemes

Bar-and-hinge approaches are mesh-dependent by design, and thus it is crucial to choose a representative triangulation scheme for origami structures. Currently, there are two types of triangulation schemes that differ by how they discretized quadrilateral panels. One is the N4B5 scheme, which simply divide a quadrilateral panel by one of its diagonals, discretizing it into two triangles. The other is the N5B8 scheme, which adds an extra interior node within each quadrilateral panel, known as a Steiner point in computational geometry [Bern and Eppstein 95], and divide it into four triangles. Triangular panels are not further discretized by both schemes. In this section, we review the basic concepts of these two schemes and generalize them to the triangulation of convex polygonal origami panels, as non-convex panels rarely appear in origami patterns.

3.1 The N4B5 scheme and its generalization to polygonal panels

The N4B5 discretization scheme is most commonly adopted for reduced-order modeling of origami structures with quadrilateral panels [Schenk and Guest 13,Filipov et al. 15], which divides each quadrilateral panel into two triangles by its shorter diagonal, as demonstrated in Fig. 2(a). If we assume that the panel bending

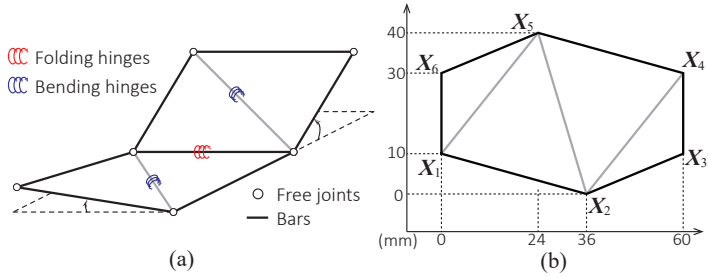

Figure 2: *(a) Illustration of a N4B5 model. (b) A hexagonal origami panel triangulated by the generalized N4B5 scheme.*

stiffness is the same per unit length along both diagonals, then shorter diagonals are easier to bend and thus require lower energy.

To extend this idea to the discretization of polygonal panels, we progressively bisect a polygon by the shortest diagonals until the original polygon is triangulated, which can be efficiently achieved using a divide and conquer algorithm [Heath 97]. In the example shown in Fig. 2(b), we first bisect hexagon 1-2-3-4-5-6 by its shortest diagonal 1-5 (could also be 2-4), dividing the hexagon into one triangle and one pentagon. Then we divide the pentagon 1-2-3-4-5 by its shortest diagonal 2-4. Finally, we dived the quadrilateral 1-2-4-5 by its shortest diagonal 2-5 to finish the triangulation.

The major advantage of the generalized N4B5 scheme is its simplicity. For certain origami patterns, such as the Miura-ori with only parallelogram panels, its accuracy is also satisfactory [Liu and Paulino 17]. However, ambiguity arises when multiple diagonals of a panel are of the same length.

3.2 The N5B8 scheme and its generalization to polygonal panels

As illustrated in Fig. 3(a), the N5B8 triangulation scheme adds a Steiner point at the intersection of the two diagonals of a quadrilateral panel, dividing it into four triangles, hence there are 5 nodes and 8 bars belong to each panel. The N5B8 scheme allows the discrete model to capture more realistic doubly curved out-of-plane deformations and isotropic in-plane behaviors of thin panels, potentially yielding higher resolution than the N4B5 scheme [Filipov et al. 17].

To extend the N5B8 discretization scheme to a polygon, a Steiner point that can simultaneously locate on all of the diagonals would be ideal. However, it is impossible for general polygonal panels to have such a Steiner point as specific quadrilaterals, as demonstrated in Fig. 3(b). Here we propose to pursue a point that has the shortest overall (weighted) distance to all diagonals of a convex polygon by solving the following optimization problem:

$$X^* = \arg\min_{X} \sum_{|j-i|>1} \frac{h_{ij}}{d_{ij}}, \tag{13}$$

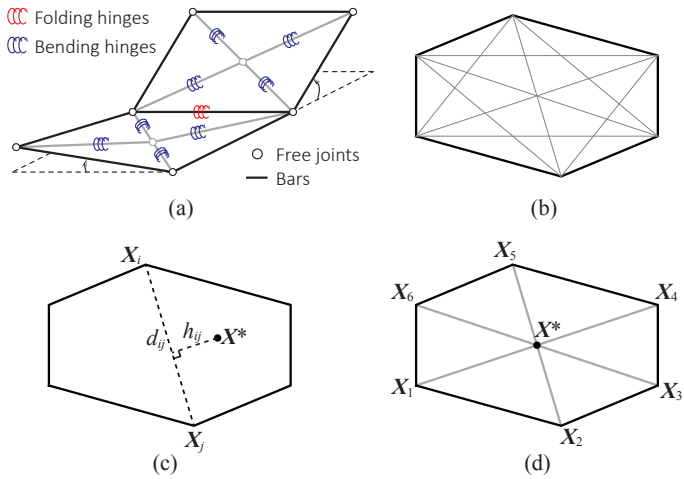

Figure 3: *(a) Illustration of a N5B8 model. (b) All diagonals of a hexagonal origami panel. (c) Illustration for d_{ij} and h_{ij}. (d) Triangulation by the generalized N4B5 scheme.*

where

$$d_{ij} = \|X_i - X_j\|, \quad \text{and} \quad h_{ij} = \frac{\|(X_i - X^*) \times (X_j - X^*)\|}{d_{ij}}. \tag{14}$$

Geometrically, d_{ij} is the length of the diagonal *i*-*j*, and h_{ij} is the distance from the Steiner point (i.e. X^*) to the diagonal *i*-*j* (see Fig. 3(c)). The measure of (h_{ij}/d_{ij}) is a weighted distance that favours shorter diagonals because they are more likely to bend than longer ones, judged by the scaled stiffness of bending ridges [Witten 07, Filipov et al. 17]. Notice that, for quadrilateral panels, the optimal solution for X^* by formula (13) is simply the intersection point of the two diagonals, and thus the extended scheme is consistent with the previous N5B8 model. In the example shown in Fig. 3(d), the optimal Steiner point for the hexagon (same as in Fig. 2(b)) is at the common intersection of three diagonals.

To recover the in-plane Poisson's effect of origami panels, the original N5B8 model [Filipov et al. 17] provides formulas for assigning bar areas based on rectangular panels. However, we have not yet derived formula for polygonal and triangular panels. Realizing that it is difficult to account for the Poisson effect for those panels, we propose to assign the bar areas such that they preserves the (linear elastic) strain energy of a continuous panel (of any shape) under uniform in-plane dilation assuming plane stress state. For a polygonal panel of thickness t and polygon area S, we assign a uniform area A for all bars belonging to this panel by:

$$A = \frac{2St}{(1-\nu)\sum_i L_i}, \tag{15}$$

where $\sum_i L_i$ is the total length of bars, and ν denotes the Poisson's ratio. When applied to rectangular panels, the results are similar to those calculated by the existing

formulas [Filipov et al. 17], which is satisfactory as an estimation tool.

3.3 Preliminary study on discretization of polygonal origami panels

Using the same hexagonal panel shown in Fig. 2(b) and Fig. 3(d) as an example, we apply load on it and compare the performance of two bar-and-hinge models triangulated by the generalized N4B5 scheme and N5B8 scheme. The applied load generates a torque on the polygonal panel (see Fig. 4(b)). Photograph of a twisted polygonal panel reveals that the surface (mean) curvature concentrates near a long diagonal, and there are two curved ridges in slight bending that spans two short diagonals, as illustrated in Fig. 4(a). We found that the generalized N5B8 model produces deformation that is closer to the physical model than the generalized N4B5 model. The deformation modes of the two discretized models are different, and the generalized N4B5 model is stiffer, as demonstrated in Fig. 4(c). We use the actual material properties of the Mylar sheet from which the polygon was made, i.e. modulus of elasticity $E = 5GPa$, Poisson's ratio $\nu = 0.35$, sheet thickness $t = 0.127mm$. Constitutive relations for bending hinges in both models implement a super-linear model [Filipov et al. 17]. We plan to make further comparison with experimental and FE results to verify and better calibrate the proposed discretization schemes.

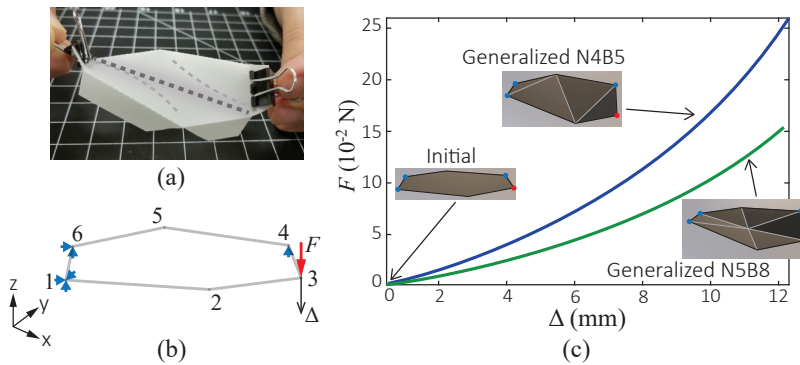

Figure 4: *(a) Photograph of a twisted hexagonal origami panel made with 0.127mm-thick Mylar sheet. Flanges are used to imitate surrounding panels. (b) Boundary conditions: blue arrows indicate fixed degrees of freedom, and red arrow indicates the applied force. (c) Load-displacement curve for both generalized N4B5 model and generalized N5B8 model. The insets show the deformed shapes of the polygonal panel using two different discretization models. The blue dots and red dots implies supporting nodes and loading node(s), respectively.*

4 Matlab implementation

In this section, we explain the details of the MERLIN2 code for origami structural analysis. We begin by describing the structure of the code, and the input/output parameters. We also make some comments on the supporting functions in the Appendix.

The kernel of MERLIN2 has two parts: discretization and nonlinear analysis. The function `PrepareData` discretizes the input origami model and formats necessary data for the structural analysis. The function `PathAnalysis` solves the nonlinear equilibrium problem defined by the formulation described in Sec. 2.

4.1 The `PrepareData` function

The following variables are inputs to the `PrepareData` function:

```
[truss,angles,AnalyInputOpt]=
  PrepareData(Node,Panel,Supp,Load,AnalyInputOpt).
```

`Node` This is a three-column array of size $N_{node} \times 3$ whose ith row represents the coordinates of the i-th node (or vertex), where N_{node} is the number of nodes in the original origami model (before discretization).

`Panel` The is a Matlab cell array containing the topology information about an origami structure. The jth entry of the cell array contains the indices of the nodes incident on the jth panel in counter-clockwise order. Each entry can vary in size, which allows for freedom on defining origami patterns that contain polygonal panels with different number of vertices.

`Supp` This is a four-column array containing information for boundary conditions of the structure. The first column holds the nodal index, and the second, third and fourth columns give support conditions for that node in the x-, y-, and z-direction, respectively. Value of 0 means that the node is free, and value of 1 specifies a fixed node.

`Load` This is structured in a similar way as `Supp`, except for the values in the second to fourth columns that represent the magnitude of the x-, y-, and z-components of the applied force or displacement.

`AnalyInputOpt` This is a Matlab structure array that contains miscellaneous information needed for the analysis. The following fields can be specified:

1. `ModelType` - This parameter specifies the discretization scheme. The user chooses between `N4B5` and `N5B8`, which refer to the generalized versions as explained in Sec. 3.
2. `MaterCalib` - This parameter specifies the calibration method for elemental constitutive behaviour. The user chooses between `manual` and `auto` mode. In the `manual` mode, the constitutive models involved in a bar-and-hinge model need to be specified explicitly using the input fields: `BarCM`, `RotSprBend`, `RotSprFold`, `Abar`, `Kb`, and `Kf`. In the `auto` mode, the constitutive models are specified implicitly based on actual material properties using the formulas defined in [Filipov et al. 17]. The following input fields are required: `ModElastic`, `Poisson`, `Thickness`, and `LScaleFactor`. The `auto` mode only applies to `N5B8` models.
3. `BarCM` This is a function that defines the constitutive model for bar elements in the following format: `[Sx,Ct,W]=BarCM(Ex)` .
   The only input is the Green-Lagrange strain E_{xx} (`Ex`), and the three outputs are the 2nd PK stress S_{xx} (`Sx`), tangent modulus C (`Ct`), and strain energy density W

(`W`). The default function is a hyper elastic model (`@Ogden`) as reported in [Liu and Paulino 17]. To customize, one can write a separate function and pass a function handle to this parameter.

4. `RotSprBend` This is a function that defines the constitutive model for bending rotational spring elements in the following format:
   `[M,k,P]=RotSprBend(he,h0,Kp,L0)`.
   The four inputs are: deformed bending angles θ (`he`), neutral angles θ_0 (`h0`), stiffness parameter(s) that may differ for each element (`Kp`), and hinge lengths L (`Lo`). The three outputs shall be resistant moment M (`M`), tangent rotational modulus k (`k`), and stored energy ψ (`P`). Input and output values (except for `Kp`) are all $N_{bend} \times 1$ arrays, where N_{bend} denotes the total number of bending hinges. The default function implements an enhanced linear elastic model [Liu and Paulino 17] (`@EnhancedLinear`) in the `manual` mode, and a super-linear model [Filipov et al. 17] (`@SuperLinearBend`) in the `auto` mode.
5. `RotSprFold` This function specifies the constitutive model for folding rotational spring elements, following the same format as `RotSprBend`, except that the size of input and output arrays shall be $N_{fold} \times 1$, where N_{fold} denotes the total number of folding hinges. The default function is the enhanced linear elastic model (`@EnhancedLinear`) in both `manual` and `auto` modes.
6. `Abar` - Areas of bar elements, which could either be a scalar (for uniform area) or a $N_{bar} \times 1$ array where N_{bar} denotes the total number of bar elements in the bar-and-hinge model.
7. `Kb` - Stiffness parameters for bending rotational spring elements, which could either be a $1 \times m$ array (for uniform property) or a $N_{bend} \times m$ array, where m is the number of required stiffness parameters, which typically equals 1. This is the input `Kp` for `RotSprBend`.
8. `Kf` - Stiffness parameters for folding rotational spring elements, following the same format as `Kb`. This is the input `Kp` for `RotSprFold`.
9. `ModElastic` - Modulus of elasticity E (or Young's modulus) of the material of the origami structure. This parameter is used in the `auto` mode.
10. `Poisson` - Poisson's ratio ν of the material, used in the `auto` mode.
11. `Thickness` - Thickness t of the origami panels, used in the `auto` mode.
12. `LScaleFactor` - Ratio of length scale factor L^*/L_F, where L_F is the length of a folding hinge and L^* is the length scale factor defined in [Filipov et al. 17]. This parameter is used to determine the rotational modulus of folding rotational springs, used in the `auto` mode. Typical values of L^*/L_F is between 1 to 5. The default value is 3.
13. `ZeroBend` - This parameter specifies the neutral angles of bending hinges. User can choose between `Flat` and `AsIs`, or specify user-defined values. This is useful when the initial configuration involves bended panels. The option `AsIs` uses the bended shapes of panels as their undeformed states; while the option `Flat` always assumes that the neutral angles of bending hinges are π (referring to flat panels). When specifying user-defined values, we use a scalar for uniform neutral angles and a $N_{bend} \times 1$ array for non-uniform neutral angles.

14. `LoadType` - User can choose between `Force` and `Displacement`.
15. `Load` - In `Force` mode, the actual applied loads are multiples of the specified load, the scalar multipliers (known as load factor) is determined automatically by a numerical continuation algorithm [Leon et al. 14]. In `Displacement` mode, the solver attempts the specified amount of displacements in an incremental manner, and stops when the specified amount of displacements is achieved.
16. `AdaptiveLoad` - This is a function that specifies an adaptive load in the following format: `[F] = LoadFun(Node,U,icrm)`.
    The inputs are initial nodal coordinates (`Node`), displacement vector (`U`), and incremental number (`icrm`), which can be regarded as pseudo-time. The output is the load vector, which can be either `Force` or `Displacement`, but each entry must corresponds to the same degree of freedom as in `U`. In `Displacement` mode, the size of each incremental step must be considered within the function. Once specified, the `AdaptiveLoad` overrides `Load`.
17. `InitialLoadFactor` - Initial load factor for the numerical continuation algorithm [Liu and Paulino 17, Leon et al. 14], used in `Force` mode.
18. `MaxIcr` - Maximum number of increments (N_{icrm}) for the numerical continuation algorithm, used in `Force` mode.
19. `DispStep` - Number of incremental steps to achieve a `Displacement` load.
20. `StopCriterion` - A function that specifies a stopping criterion for the numerical simulation based on configurational changes of the origami structure. Use the following format: `[Flag] = StopCriterion(Node,U,icrm)`. The function returns 1 when the stopping criterion is met; otherwise the function should return 0.

The outputs of the `PrepareData` function are:

`truss` This is a Matlab structure array that contains information about the bar elements. It has the following fields:

1. `Node` - This is a three-column array of size $N'_{node} \times 3$ in the same format as `Node`, but contains additional Steiner points for generalized N5B8 models.
2. `Bars` - Connectivity information of bar elements stored in a $N_{bar} \times 2$ array. The two entries in a row refer to the indices of the two end nodes of a bar element.
3. `Trigl` - Triangulation information stored in a $N_{tri} \times 3$ array, where N_{tri} equals the total number of triangles in the bar-and-hinge model after discretization. The three entries in a row are vertex indices of each triangle.
4. `B` - Compatibility matrix of the bar frame [Filipov et al. 17] of size $N_{bar} \times N_{dof}$, where N_{dof} is the total number of degrees of freedom in the system ($= 3N'_{node}$).
5. `L` - Initial lengths of bar elements stored in a $N_{bar} \times 1$ array.
6. `FixedDofs` - Indices of fixed degrees of freedom specified by `Supp`.
7. `CM` - Constitutive model for bar elements, passed from `AnalyInputOpt.BarCM`.
8. `A` - Member areas of bar elements stored in a $N_{bar} \times 1$ array.
9. `U0` - Initial displacement referring to the unformed configuration. Most of the time, the initial configuration of a structure in a simulation is the undeformed

configuration, thus `U0` is a $N_{dof} \times 1$ array of zeros. This parameter is specified outside the `PrepareData` function, but before the `PathAnalysis` function is executed. If not specified, the `PathAnalysis` function assigns `truss.U0` a vector of zeros.

`angles` This is a Matlab structure array that contains information about the rotational spring elements. It has the following fields:

1. `Panel` - Same as the aforementioned input parameter `Panel`.
2. `fold` - A $N_{fold} \times 4$ array that contains indices of associated nodes of folding rotational springs. The first two entries define the rotation axis, and the later two refer to off-axis points of the two adjacent triangles in a rotational spring element. See [Liu and Paulino 16] for examples.
3. `bend` - A $N_{bend} \times 4$ array that contains indices of associated nodes of bending rotational springs, in the same format as `angles.fold`.
4. `pf0` - Neutral angles of folding rotational springs stored in a $N_{fold} \times 1$ array.
5. `pb0` - Neutral angles of bending rotational springs stored in a $N_{bend} \times 1$ array.
6. `CMbend` - Same as `AnalyInputOpt.RotSprBend`.
7. `CMfold` - Same as `AnalyInputOpt.RotSprFold`.
8. `Kb` - Stiffness parameters for bending rotational springs specified for each element in an array of N_{bend} rows.
9. `Kf` - Stiffness parameters for folding rotational springs specified for each element in an array of N_{fold} rows.

4.2 The `PathAnalysis` function

The `PathAnalysis` function takes in the outputs of the `PrepareData` function and conducts the nonlinear structural analysis:

```
[Uhis,Fhis]=PathAnalysis(truss,angles,AnalyInputOpt).
```

In `Force` mode, the function implements a numerical continuation algorithm called Modified Generalized Displacement Control Method (MGDCM) [Leon et al. 14]. In addition, to improve convergence performance, the function heuristically adjusts the constraint radius, which determines the incremental load factors. In each increment, when the number of iterations to convergence is large, the constraint radius is reduced to make a more conservative subsequent increment; when the number of iterations is very small, the constraint radius is increased, yielding a more aggressive step for the next increment. In `Displacement` mode, the function divides the `Displacement` load into multiple small increments. Each increment is solved by a damped Newton-Raphson algorithm [Wriggers 08]. A standard incremental step of displacement load is the prescribed amount divided by the number of `DispStep`. The implemented algorithm makes attempts to reduce or increase the incremental movements, and adjusts damping factors based on the convergence performance of the current increment. Therefore, the actual number of increments N_{icrm} performed to achieve the prescribed displacement load may not equal to `DispStep`.

The outputs of the function `PathAnalysis` are:

1. `Uhis` - History of nodal displacements at the end of each increment, stored in a $N_{dof} \times N_{icrm}$ array.
2. `Fhis` - In `Force` mode, this is a $N_{icrm} \times 1$ array that stores the values of load factors at the end of each increment. In `Displacement` mode, this is a $N_{icrm} \times N_{disp}$ array, whose columns store the negative values of the resistant forces in the degrees of freedom that are imposed with displacement loads, where N_{disp} is the number of imposed degrees of freedom.

5 An example using MERLIN2

In this example, we demonstrate how MERLIN2 can be used to conduct structural analysis of a custom designed origami pattern. A generalized Miura-ori is created using the Freeform Origami software by Tomohiro Tachi [Tachi 10] and imported to MERLIN2 as an OBJ file. Simulation results are shown in Fig. 5. We choose the `N5B8` model under `auto` mode by specifying modulus of elasticity $E = 1GPa$, Poisson's ratio $\nu = 0.3$, material thickness $t = 0.25mm$, and ratio of length scale factor $L^*/L_F = 2$. The structure is loaded by displacement.

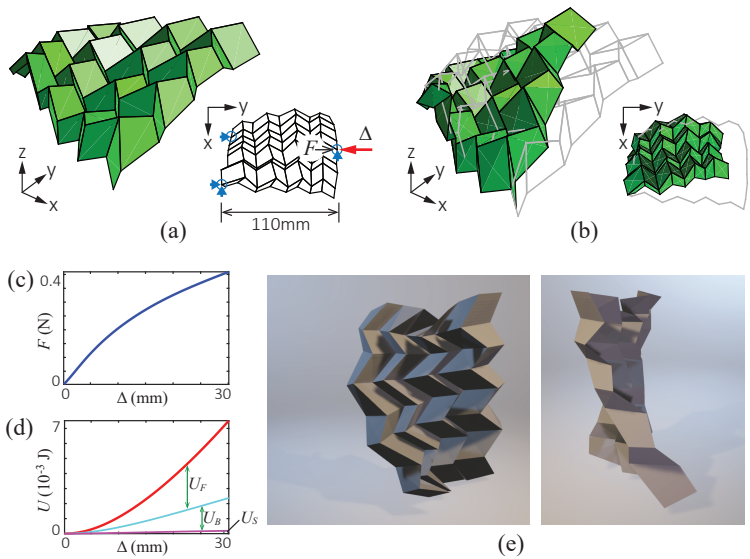

Figure 5: *(a) Initial configuration of a generalized Miura-ori. The plan view shows boundary conditions for numerical simulation. Blue arrows indicates support in x and y-direction, while blue circles indicates support in the z-direction. the red arrow shows the applied displacement load. Black arrow marks the balance force along the specific direction of loading. (b) The deformed configuration. The grey wire frame refers to initial configuration. (c) Load-displacement plot. (d) Energy-displacement plot. Contributions from three deformation modes are differentiated: folding (U_F), bending (U_B) and stretching (U_S). (e) Two views of the rendering of the deformed origami, generated using the exported OBJ file from MERLIN2.*

6 Conclusions

Sharing and publication of open-source and educational software has been a good tradition in the origami community, such as "TreeMaker" [Lang 11], "Freeform Origami" [Tachi 10], and "Rigid Origami Toolbox" [Gattas et al. 13]. In this paper, we present the MERLIN2 software as a powerful and easy-to-use tool for nonlinear structural analysis of origami assemblages, aiming at speeding origami design cycles and educating origami engineering. The MERLIN2 software is available at the following url: http://paulino.ce.gatech.edu/software.html.

Acknowledgements

This paper has been awarded the 7OSME Gabriella & Paul Rosenbaum Foundation Travel Award. We also thank the support from the US National Science Foundation (NSF) through grant no.1538830, the China Scholarship Council (CSC), and the Raymond Allen Jones Chair at the Georgia Institute of Technology.

Appendix: Some comments on the supporting functions

In addition to the kernel functions, the MERLIN2 software provides supporting functions for data communication and interpretation.

`ReadOBJ` and `Write2OBJ` The wavefront OBJ file format is a standard format for sharing 3D geometry, supported by many design software programs including the origami design tool "Freeform Origami" [Tachi 10]. The `ReadOBJ` function directly imports an OBJ coded 3D origami assemblage into MERLIN2. The `Write2OBJ` function writes the geometry of a deformed origami simulated by MERLIN2 to OBJ file. Necessary inputs to this function are: file name, nodal coordinates (`Node`), and triangulated mesh (`truss.Trigl`).

`PostProcess` This function computes physical measures for the bar-and-hinge model based on the deformation history stored in `Uhis`, such as strains of bar elements and system energies. All measures are stored in a Matlab structure array (named as `STAT`). A complete list of outputs are shown below:

1. `bar` - Information about bar elements at every increment: Green-Lagrange strain (`bar.Ex`), 2nd PK stress (`bar.Sx`), stored energy of each bar element (`bar.USi`), and total stored energy of bar elements (`bar.US`). Attributes `bar.Ex`, `bar.Sx`, and `bar.USi` are of size $N_{bar} \times N_{icrm}$. Attribute `bar.US` is a $1 \times N_{icrm}$ array.
2. `fold` - Information about folding rotational springs at every increment: folding angle (`fold.Angle`), resistant moment (`fold.RM`), stored energy of each folding hinge (`fold.UFi`), and total stored energy of folding hinges (`fold.UF`). Attributes `fold.Angle`, `fold.RM`, and `fold.UFi` are of size $N_{fold} \times N_{icrm}$. Attribute `fold.UF` is a $1 \times N_{icrm}$ array.
3. `bend` - Same as `STAT.fold`, but for bending rotational springs, which also has 4 attributes: bending angle (`bend.Angle`), resistant moment (`bend.RM`), stored energy of each bending hinge (`bend.UBi`), and total stored energy (`bend.UB`).
4. `PE` - Total potential energy of the origami structure, stored in a $1 \times N_{icrm}$ array.

`PlotOri` This function generates origami renderings with various options. The format of this function reads:

`PlotOri(Node,Panel,Trigl,Name,Value).`

Among the inputs, `Node` and `Panel` are necessary. If `Trigl` is provided, the function will draw the triangulated origami model, which should be used for plotting deformed origami structures with bent panels. We need to feed an empty array `[]` to the third input of the function when `Trigl` is not needed, e.g. `PlotOri(Node,Panel,[])`. Users may use one or more `Name,Value` pair arguments to specify rendering properties. The following options are available:

1. `PanelColor` - Specify a uniform colour for panels. For example, to plot the origami in blue colour, specify `PlotOri(__,__,__, 'PanelColor',[0,0,1])`. The default colour is green.
2. `FoldEdgeStyle` - Line style for the folding hinges. To plot the folding hinges in dashed lines, specify `PlotOri(__,__,__,'FoldEdgeStyle','--')`. The default style is solid line.
3. `BendEdgeStyle` - Line style for the bending hinges. The default style is `none`, so that the origami plot does not display bending hinges. This property is only used when `Trigl` is provided.
4. `EdgeShade` - A scalar between 0 and 1 that specifies the greyscale of edges in the plot. Use value 1 for black (default) and 0 for transparent.
5. `ShowNumber` - The `PlotOri` function shows the indices of nodes in the origami plot if this property is set to `on`. This is very helpful for specifying boundary conditions (i.e. `Supp` and `Load`). When this option is `on`, the panels are plotted without colour (transparent). The default value is `off`.
6. `FaceVertexColor` - Face and vertex colours, specified as one colour per face, or one colour per vertex for interpolated face colour. For one colour per face, use an $N_{face} \times 3$ array of RGB triplets, where N_{face} is the number of origami panels (`Panel`) when `Trigl` is not provided, or the number of triangles when `Trigl` is provided. For interpolated face colour based on vertex values, provide an $N'_{node} \times 3$ array of RGB triplets.
7. `EdgeColor` - Colours for all bars in the bar-and-hinge model, specified as one colour per bar, including bending hinges, folding hinges, and boundary edges. Use an $N_{bar} \times 3$ array of RGB triplets. To enable this property, connectivity of bar elements and N_{bend} needs to be specified. For example, if the bar colours are stored in the array `CData`, then use `PlotOri` as:
   ```
   PlotOri(__,__,__,'EdgeColor',CData,'Bars',truss.Bars,...
           'NumBendHinge',size(angles.bend,1)).
   ```

`VisualFold` This function animates the numerical simulation. The format of this function reads:

`VisualFold(Uhis,truss,angles,Fhis,instdof,Name,Value).`

The input `Uhis` is the output of `PathAnalysis` function. The input `Fhis` is optional, which allows the `VisualFold` function to animate the load-displacement digram in accordance with the animation of deformation history. The parameter `instdof` is a 2×1 array that specifies the degree of freedom for displacement

measure. The first entry stores the node being tracked, and the second entry is a signed integer defining the direction of nodal displacement to be monitored, for example, -3 indicates the minus z-direction and 2 indicates the positive y-direction. Use one or more `Name,Value` pair arguments to specify particular display options. The following options are available:

1. `Showinitial` - If set to `on`, the `VisualFold` function displays a grey-coloured wire frame of the initial configuration during the entire animation(s) of deformation history.
2. `Recordtype` - The `VisualFold` function can record the animation in video format (MP4) or animated image format (GIF). Specify `video` or `imggif` for video recording or image recording, respectively. The default option is `none`.
3. `Filename` - A file name for the generated video or animated image.
4. `IntensityMap` - This option enables the function to plot faces or edges in varying colours during the animation based on a physical measure specified in `IntensityData`. There are three choices: `Vertex`, `Edge`, and `Face`. This is useful for visualizing intensity of forces acting on nodes, strains in bars, or area change of triangulated panels.
5. `IntensityData` - A N_{icrm}-column array. Each column specifies a physical measure for all entities of a group (e.g. nodes, bars) in the origami model at one increment of simulation. Use a $N'_{node} \times N_{icrm}$ array for `IntensityMap` of `Vertex`, a $N_{bar} \times N_{icrm}$ array for `IntensityMap` of `Edge`, and a $N_{face} \times N_{icrm}$ array for `IntensityMap` of `Face`.
6. `Viewangle` - View angle for the animation of origami structure, specified in [azimuth angle, elevation angle] format.
7. `Axislim` - Axes limits for the animation of load-displacement curve. The default option automatically adjusts the axes sizes.

References

[Bern and Eppstein 95] M. Bern and D. Eppstein. "Mesh generation and optimal triagulation." In *Computing in Euclidean Geometry*, edited by D. Z. Du and Hwang F. K., Second edition, pp. 47–123. World Scientific, 1995.

[Filipov et al. 15] Evgueni T. Filipov, Tomohiro Tachi, and Glaucio H. Paulino. "Origami tubes assembled into stiff, yet reconfigurable structures and metamaterials." *Proceedings of the National Academy of Sciences* 112:40 (2015), 12321–12326.

[Filipov et al. 17] E. T. Filipov, K. Liu, T. Tachi, M. Schenk, and G. H. Paulino. "Bar and hinge models for scalable analysis of origami." *International Journal of Solids and Structures* 124 (2017), 26–45.

[Gattas et al. 13] Joseph M. Gattas, Weina Wu, and Zhong You. "Miura-base rigid origami: parameterizations of first-level derivative and piecewise geometries." *Journal of Mechanical Design* 135 (2013), 111011.

[Heath 97] M.T. Heath. *Scientific Computing: An Introductory Survey*. Second Edition, McGraw-Hill, 1997.

[Lang et al. 15] Robert J. Lang, Spencer Magleby, and Larry L. Howell. "Single-degree-of-freedom rigidly foldable cut origami flashers." *Journal of Mechanisms and Robotics* 8 (2015), 031005.

[Lang 11] Robert J. Lang. *Origami Design Secrets*, Second edition. CRC Press, 2011.

[Leon et al. 14] Sofie E. Leon, Eduardo N. Lages, Catarina N. de Araújo, and Glaucio H. Paulino. "On the effect of constraint parameters on the generalized displacement control method." *Mechanics Research Communications* 56 (2014), 123–129.

[Liu and Paulino 16] Ke Liu and Glaucio H Paulino. "MERLIN : A MATLAB implementation to capture highly nonlinear behavior of non-rigid origami." In *Proceedings of the IASS Annual Symposium*, edited by K Kawaguchi, M. Ohsaki, and T Takeuchi. Tokyo, Japan, 2016.

[Liu and Paulino 17] K. Liu and G. H. Paulino. "Nonlinear mechanics of non-rigid origami: an efficient computational approach." *Proceedings of the Royal Society A* 473 (2017), 20170348.

[Ota et al. 16] N. S. N. Ota, L. Wilson, A. Gay Neto, S. Pellegrino, and P. M. Pimenta. "Nonlinear dynamic analysis of creased shells." *Finite Elements in Analysis and Design* 121 (2016), 64–74.

[Ramm and Wall 04] E. Ramm and W. A. Wall. "Shell structures - A sensitive interrelation between physics and numerics." *International Journal for Numerical Methods in Engineering* 60:1 (2004), 381–427.

[Schenk and Guest 11] Mark Schenk and Simon D Guest. "Origami folding: A structural engineering approach." In *Origami 5*, edited by Patsy Wang-Iverson, Robert J Lang, and Mark Yim, pp. 293–305. CRC Press, 2011.

[Schenk and Guest 13] Mark Schenk and Simon D. Guest. "Geometry of Miura-folded metamaterials." *Proceedings of the National Academy of Sciences* 110:9 (2013), 3276–3281.

[Tachi 10] Tomohiro Tachi. "Freeform Variations of Origami." *Journal for Geometry and Graphics* 14:2 (2010), 203–215.

[Witten 07] T. A. Witten. "Stress focusing in elastic sheets." *Reviews of Modern Physics* 79:2 (2007), 643–675.

[Wriggers 08] Peter Wriggers. *Nonlinear Finite Element Methods*. Springer, 2008.

K. Liu
School of Civil and Environmental Engineering, Georgia Institute of Technology, Atlanta, Georgia, USA. e-mail: ke.liu@gatech.edu

G. H. Paulino
School of Civil and Environmental Engineering, Georgia Institute of Technology, Atlanta, Georgia, USA. e-mail: paulino@gatech.edu