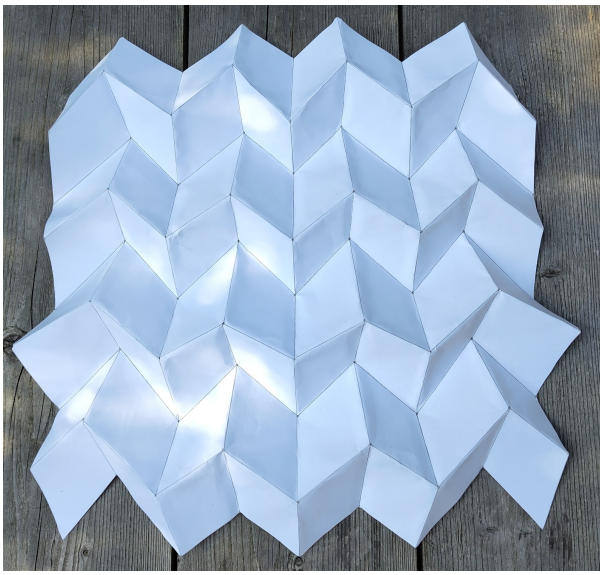
Miura-ori pattern optimization for origami shape matching using the bar-and-hinge model

Niels Beaufort

Department of Precision and Microsystems Engineering

Report no : 2024.098
Supervisor : Alejandro M. Aragón
Specialisation : Computational Design and Mechanics
Type of report : Master of Science thesis
Date : 18 October 2024



Abstract

Origami has many advantages, such as deployability, low mass, and ease of miniaturization; it has potential contributions in many applications. Specifically, origami that has been designed to fold into a target shape can be used in deployable tents, bridges, grippers, packaging, and deformable mirrors for laser communication. This work presents a two-step methodology for optimizing origami designs such that the distance between the bottom side of an origami and its target surface is minimized—the origami will thus lay "on top" of its target surface. The method combines the strengths of rigid origami and non-rigid origami: it starts with a non-foldable rigid origami Miura-ori tessellation that matches the target surface and optimizes this design such that it becomes foldable. Then it uses this crease pattern as the initial design for an optimization cycle that includes the nonlinear N5B8 bar-and-hinge non-rigid origami model and converges to the crease pattern that folds into the target shape, demonstrated in this work with a saddle, dome, and arch. The results show that for an 800 mm by 800 mm design, the folded saddle origami has a maximum distance to its target surface of 24 mm, with the other tested shapes having similar outcomes. This performance will be sufficient for some use cases, but more work is required for high-accuracy applications.

1. Introduction

Origami is the ancient Japanese art of paper folding. The first paper material was made in China in the second century BCE and this practice was transformed into a systematic process in the following centuries [1]. The earliest decorative origami were "shide"—zigzag geometric shapes that are placed on shrines—and originated in Japan around the fourteenth century [2, 3]. Recreational origami emerged in the sixteenth and seventeenth century [2–5], with examples of a Japanese sword with a picture of an origami crane, and a poem mentioning an origami butterfly [6]. In the eighteenth century, Friedrich Fröbel, inventor of the kindergarten, introduced origami into western children's education. This is also when Japanese origami was influenced by German ideas and the concept of "pure" origami arose [7]. Pure origami is made from a square piece of paper with a different colour on each side and does not allow any cutting nor adhesives. The counterpart to pure origami is kirigami, where cutting *is* allowed, but even in kirigami, there is no gluing [8]. Another good distinction to make here is rigid versus non-rigid origami. Rigid origami is a mathematical construct where all deformation of the structure is concentrated in the fold lines—the facets of the origami do not deform, they are considered infinitely stiff. In non-rigid origami, however, the facets *can* stretch, shear, and bend [9]. And although there have been investigations into origami with thick facets [10], this research will be limited to thin facets.

While origami was originally just recreational, it has various properties that are interesting for engineers. The structures can not only be made of paper-like materials, but also of more traditional engineering materials, such as metals, where the deformation is much less concentrated in the pre-made creases [11]. By effectively placing folds in metal sheets, structures can have high stiffness and strength, yet have low mass. Also, it is possible to create moving mechanisms with less parts. That is a general property of compliant mechanisms, of which origami can be considered a subset [12]. With the absence of components such as gears, bearings, and belts, miniaturization is comparatively straightforward. Finally, in terms of manufacturing, origami structures or mechanisms can be made cheaply by either laser-cutting the fold lines or using a mould [13]. Of course, origami also has disadvantages: deforming origami requires energy and fatigue is a prevalent cause of failure [14].

In the late twentieth century, mathematicians and engineers began to see the potential of origami for engineering applications. In 1985, a Japanese astrophysicist named Koryo Miura invented the Miura fold, or Miura-ori (ori translates to fold) [15,16]. A flat sheet with a Miura fold pattern can be folded very compactly and then unfolded with simple actuation. In 1995, this pattern was used to store a Japanese satellite's solar panel array in a rocket and subsequently unfold it to a flat sheet when in space [16, 17]. This example displays the deployability and size-variability of origami. Another example is an origami stent that is folded to a small diameter when it is inserted into the body. The stent can expand and hold open a blood vessel once it is in the correct location [18].

An origami field of research that emerged around 1990 is shape matching, which is the process of designing origami crease patterns that, when folded, attain a required shape. A particular challenge within this field is approximating non-developable surfaces, which are surfaces with a non-zero Gaussian curvature, which itself is defined as the product of the two curvatures at a point on a surface. Gaussian curvature can be positive, negative, or zero. Developable surfaces such as flat sheets and cylinders have zero Gaussian curvature and are simple to approximate, or even exactly match, with a flat sheet of paper. Contrarily, non-developable surfaces have a non-zero Gaussian curvature. Examples of these surfaces are saddles, spheres, and hyperboloids, and these are more difficult to approximate. The total curvature (which is the surface integral of the Gaussian curvature [19]) can never change by folding [8]. The global curvature (the curvature of the overall shape), on the other hand, can be changed by folding, and that is how origami shape matching approximates curved surfaces.

Applications of shape matching origami include many deployable structures such as tents, emergency shelters, or even bridges. Additionally, grippers that are designed for specific objects with difficult-to-grasp shapes could take advantage of such a methodology, as well as packaging that needs to absorb impacts and protect oddly-shaped objects. One particularly fitting development that could be accelerated with this technology is laser- (as opposed to radio-) communication, in particular in space. This is a developing research field that provides secure and increasingly fast data transfer in communication systems that consist of satellites and ground stations [20, 21]. These systems can use heavy solid mirrors with a non-symmetrical surface to process the laser signals [20]. Those surfaces could instead be made by placing a reflective sheet on an origami structure that was designed to fold into this target surface—creating a new design that is lighter and deformable, which gives it more possibilities as an optical component.

Before we move to the current state of origami shape matching, we first discuss origami modelling, as that is a prerequisite for shape matching. The most basic representation of origami is a list of nodes and their connectivity, similar to finite elements. And most models build on top of this representation to include folding mechanics. In the realm of rigid origami, Zimmermann et al. explored 1-degree of freedom (DOF) origami and presented the principle of three units (PTU) [22], which says that any vertex with n connected creases has $n-3$ DOFs. The framework considers inputs and outputs to every vertex and divides the vertex into three "units", which can be used to compute whether a design is rigidly foldable.

Non-rigid origami models are generally more involved than rigid origami models, as they must capture the additional mechanics of bending and folding. This can be directly modelled with the finite element method, using shell elements [23,24], which results in accurate displacement fields and stress distributions. But there are drawbacks, such as the high computational cost and numerical artefacts such as shear locking and membrane locking [25].

That is why Schenk and Guest introduced what would later be called the bar-and-hinge model—a pin-jointed truss representation of origami with triangular and quadrilateral facets [26]. The key idea of bar-and-hinge models is the following: "because of the relatively high in-plane stiffness of the sheets, a straight fold line between surfaces tends to remain straight after adjacent material deforms. A panel surrounded by such creases is highly resistant to buckling, and as a result, a triangular face tends to remain planar, while a quadrilateral face tends to exhibit bending only along one of the diagonals" [27]. In this model, every fold line is replaced by a bar with a finite stiffness. In quadrilateral facets, an additional bar is placed along the shortest diagonal, as it is likely that the facet will bend along that diagonal. Rotational springs are placed within the quadrilateral facets over the shortest diagonal and over the fold lines to simulate bending and folding stiffness, respectively. This mechanical representation is translated into an elastic, linear, finite element-like model with a stiffness matrix, nodal displacements, and nodal forces. To distinguish between these bar-and-hinge models, they have a code in the format NXBX, where the X's indicate the number of nodes and bars in the model per quadrilateral facet. The model discussed in this paragraph is an N4B5 bar-and-hinge model, which means that there are four nodes (one on each corner of the facet) and five bars (one on each side plus one diagonal).

Liu and Paulino proposed a nonlinear formulation for the N4B5 model [25], incorporating both geometric and material nonlinearities. The nonlinear analysis is solved with the modified generalized displacement control method, a variant of arc-length methods [25]. The method can accurately simulate folding and bending of origami structures, even when there is bistability involved. Liu and Paulino implemented the formulation in the MERLIN software [28]. A disadvantage of the N4B5 model is the anisotropy, resulting from the single diagonal bar in every quadrilateral facet.

Filipov et al. improved on the N4B5 model by adding a second diagonal bar and rotational spring in each quadrilateral facet [29], resulting in the N4B6 bar-and-hinge model. The addition leads to a more accurate model when there are facets with diagonals of (almost) the same length. A disadvantage of the N4B6 model is that it cannot accommodate large panel bending deformations.

In an effort to overcome the drawbacks of the N4B5 and N4B6 models, Filipov et al. introduced a fifth node in the middle of quadrilateral origami facets, creating the N5B8 bar-and-hinge model [27], later implemented in MERLIN2 [30]. Figure 1 shows how a sheet with fold lines is transformed into the N5B8 model. The blue dots are the nodes, the black lines are the bars, and the green and orange coiled lines are the springs or "hinges" for bending and folding, respectively. This model uses material properties as inputs for the model's parameters and can both isotropically simulate origami and accommodate large panel bending deformations. To verify, the authors used a sufficiently discretized finite element model as the baseline. The N5B8 results show almost the same global deformation modes as the finite element model in various load cases, such as stretching, shearing, bending, and folding. Filipov et al. also described how to perform a nonlinear analysis for large deformations. Limitations of the model include the inability to capture stress concentrations and the reduced nodal displacement accuracy compared with a discretized finite element model. That is why the model is primarily useful in early design processes or optimization cycles. In these situations, analysis speed is more important than absolute accuracy with respect to stress distributions or displacement fields.

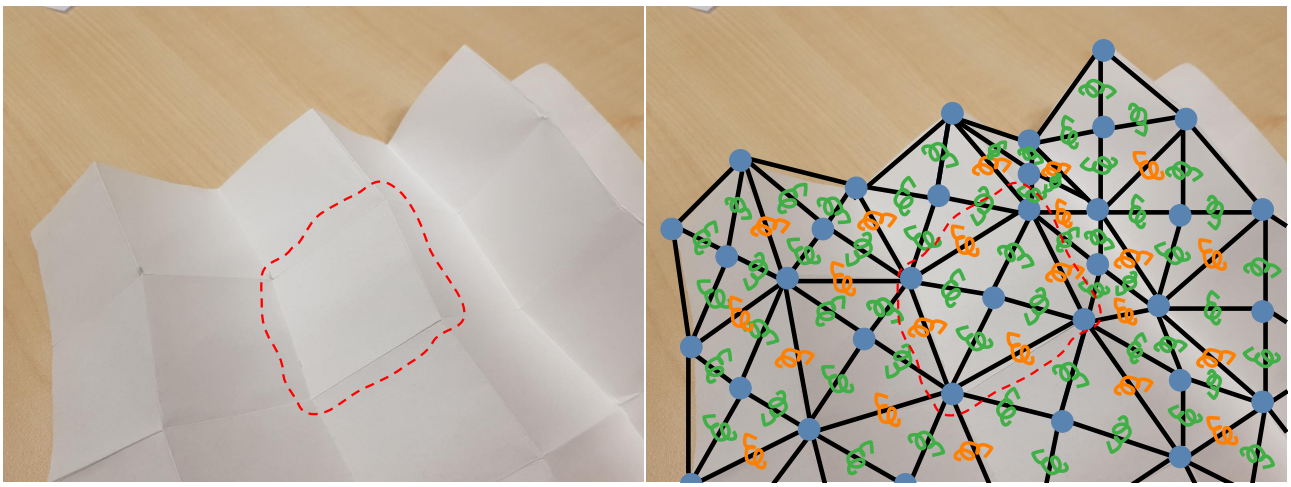


Figure 1: Transformation from creased sheet to bar-and-hinge model

With the current state of origami models covered, we will now discuss how these models are applied to perform shape matching. Historically, crease pattern design towards a desired folded shape was based on experience and intuition [31]. The mathematician Robert Lang was the first to present an algorithm that can create a crease pattern to form a wide range of shapes [31]. For this algorithm, the shape must first be represented as a main body with appendages, a so-called stick figure. For example: a giraffe would be represented by a main body, with four sticks for the legs, one for the tail, one for the neck, and one for the head. This stick-figure is then projected onto a flat, square piece of paper and, using a constrained optimization, the paper is divided up by crease lines into a base origami with the correct number and size of appendages [31]. Once the base is folded, it is relatively easy to fold the details of the figure—for experienced origami artists, that is. This strategy is very suitable for designs that can be appropriately transformed into stick figures. Examples of these are mostly artistic and recreational: animals, plants, and human figures.

For approximating more general shapes that cannot necessarily be described by a stick figure, we require other methods. Evans et al. described rigid folding principles with example configurations ranging from the chicken wire tessellation to the generalized Miura-ori pattern [32]. Demaine and Tachi solved the problem of "origamizing" any polyhedral surface in an efficient manner [33]. Whereas previous methods were very inefficient (winding the polyhedron with a thin strip [34]) or could not fold all shapes [35], this method is efficient and can fold any polyhedral surface. An important property of the crease patterns is that they are watertight. The property of watertightness is satisfied when the boundary of the square piece of paper becomes the boundary of the surface you're trying to make—an origami bowl with this property will not spill any water. The algorithm is implemented in software with the name *Origamizer* [36].

Next, both Zimmermann et al. and Walker and Stankovic used the previously discussed principle of three units to build graphs of 1-DOF origami patterns [37, 38]. These graphs are a useful tool to build and vary origami "topologies". In both papers, the authors build, expand and optimize the origami designs for objectives like following a path and shape matching.

Staying with 1-DOF rigid origami, Dudte et al. have worked on Miura-ori tessellations to create curved origami surfaces [39]. They construct a grid of Miura-ori units and optimize the structure to make it foldable, resulting in non-developable surfaces, with the global curvature matching the target surface. This work is the basis for the first (rigid origami) part of the method in section 2.

Previous paragraphs discussed rigid origami shape matching methods, which is a well-developed field. But in the field of non-rigid origami, there has been little work on shape matching. There is the work by Fuchi et al. using the N4B5 bar-and-hinge model to perform crease topology optimization and obtain the desired deformation in an actuator [40], which is similar to shape matching. And other non-rigid origami designs exist, but they do not target shape matching at all. Take Filipov et al.'s work on optimizing stiffness ratios of eigenmodes of a Miura-inspired origami tube [29]. And Li et al., for instance, designed a tunable origami structure made up of two stacked units of the Kresling pattern [41]. Ye et al. also used the Kresling pattern and adapted it to design radially closable origami valves [42]. All of these designs and design methods are interesting, but they are not shape matching, and this leaves the field of non-rigid origami shape matching relatively underdeveloped.

The absence of an accurate, non-rigid origami design strategy that helps with designing origami crease patterns that can fold into a desired shape is a gap in the scientific knowledge that this research seeks to fill. The term non-rigid is critical here, because by also considering stretching, shearing, and bending of the panels, the method will be in better agreement with the material realities of origami engineered from sheets of metal, and will also encompass a larger range of design options than rigid origami. The method is a step towards origami becoming a more standard design direction in mechanical engineers' toolboxes, and with origami's advantages and applications, that is a prospect worth looking forward to.

The design strategy's inputs will be a desired shape, and the output is the crease pattern that can fold into said shape. There are two parts to the method: first, a rigid origami part, inspired by the work by Dudte et al. [39] that gives a rough design. And second, a non-rigid origami part that uses the fast N5B8 bar-and-hinge model, the arc-length path-following method, and a zeroth order optimization algorithm to iterate on the rough design and finish with a better performing detailed design. The method was tested with three target surfaces: the saddle, the dome, and the arch, of which the saddle and the dome are non-developable shapes.

2. Method

This section describes the design workflow, which can be split into two parts, depicted in figure 2. Following the work of Dudte et al. [39], we place Miura-ori unit cells to construct rigidly-foldable crease patterns that can match curved surfaces. The result of this rigid design (part 1) is then taken as the initial design for the optimization cycle in part 2. This second part yields a crease pattern that matches the same target shape, but the folding is modelled with the non-rigid N5B8 bar-and-hinge model.

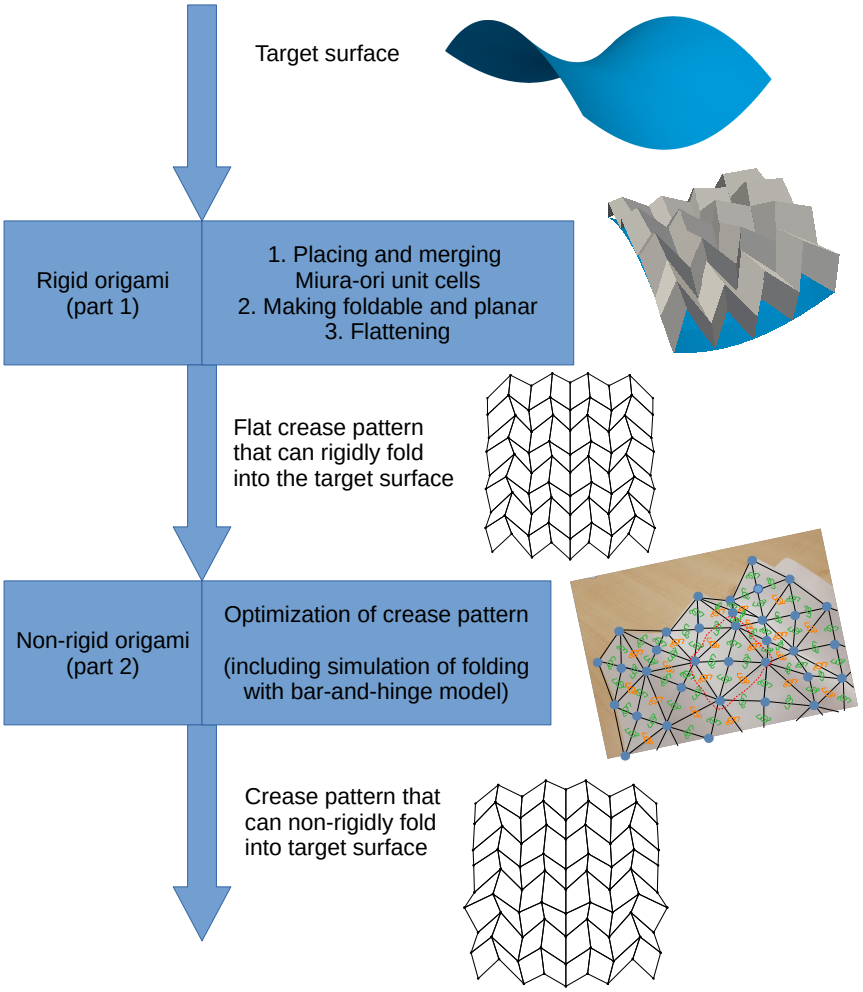


Figure 2: Overview of the method

2.1 Rigid origami shape matching with Miura-ori unit cells

Since the non-rigid origami shape matching process is relatively slow, it is beneficial to have an initial design that is close to the target surface. The rigid origami approach by Dudte et al. [39] is a fast and accurate way of using Miura-ori-like unit cells to match a curved target surface. The approach consists of three parts: placing and merging Miura-ori-like unit cells, making the pattern foldable and planar, and flattening the pattern. The origami unit cells in the first part are referred to as "Miura-ori-like unit cells", because they are artificially formed, and are not formed from a flat sheet. After they have been optimized to make them foldable from a flat sheet in the second part, they are referred to as regular Miura-ori unit cells. The three parts will be described and visualized by means of a saddle shape.

2.1.1 Miura-ori-like unit cell placement and merging

The formula for a saddle centered on the origin is $z = \left(\frac{x}{a}\right)^2 - \left(\frac{y}{b}\right)^2$, where the values for a and b determine the curvatures in two directions—the smaller a and b, the stronger the curvatures. This saddle must first be tiled with a number of quadrilaterals, see figure 3a. Every tile is then transformed by placing extra nodes in the middle of the sides and one in the center, see figure 3b. The nodes are then moved such that the cell resembles a Miura-ori unit cell. In figure 3c, one unit cell's nodes have been coloured. Pairs of nodes with the same colour correspond to the same node, but shifted. Unshifted nodes have a blue boundary around the coloured node and the shifted nodes have an orange boundary. The black nodes do not shift at all. At this stage, there is a set of disconnected unit cells on the surface. These unit cells need to be merged; which means that some pairs of nodes need to be replaced by a single node, averaging their positions, see the red and purple nodes in figure 3d. The result of this merger is an origami representation of the saddle surface (figure 3e); specifically, the nodes at the corners of the Miura-ori-like unit cells are points on a grid projected on the surface. Figure 3f shows the origami shape.

Figure 4 shows a graphic, adapted from the work of Dudte et al. [39], with a more clear depiction of the Miura-ori-like unit cell generation and merging process, but on a flat surface and with only four unit cells. Figures 4a-b show the transition from a flat unit cell to a Miura-ori-like unit cell. In figure 4c, the green nodes in the red boxes are to be averaged, and the blue nodes in the red boxes simply need to be merged into a single node. A detailed view of the merging process is provided in appendix A.2.

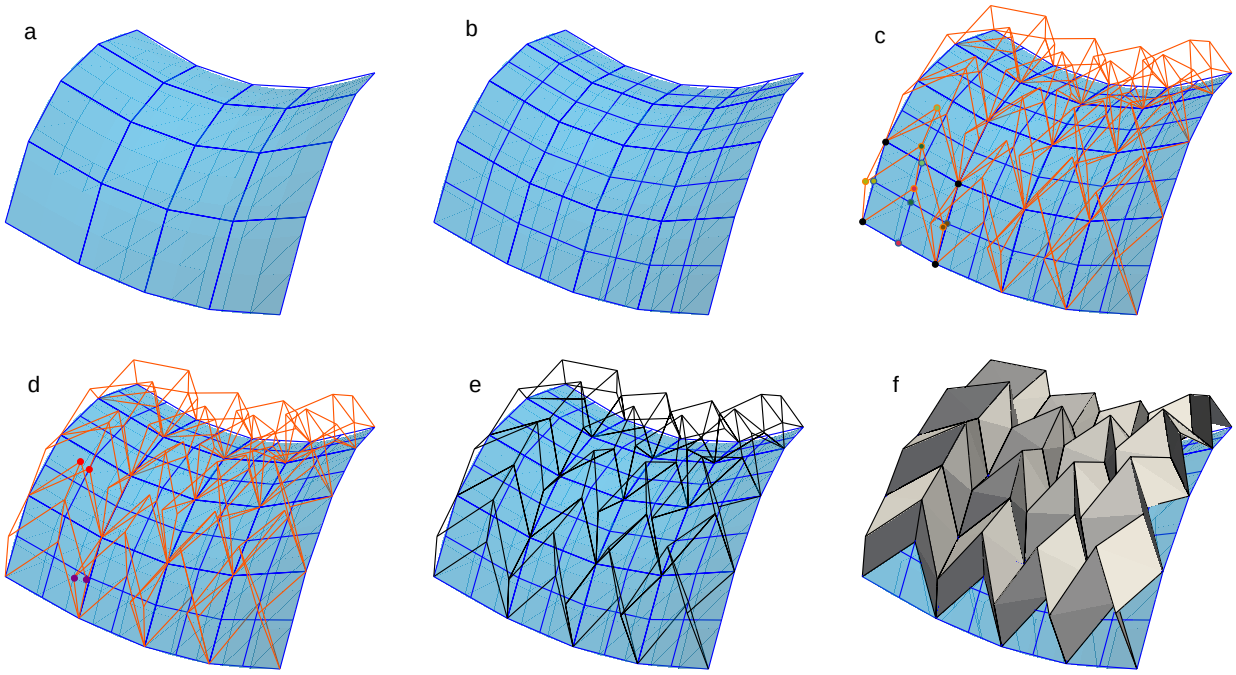


Figure 3: Step-by-step construction of Miura-ori-like pattern on a saddle surface

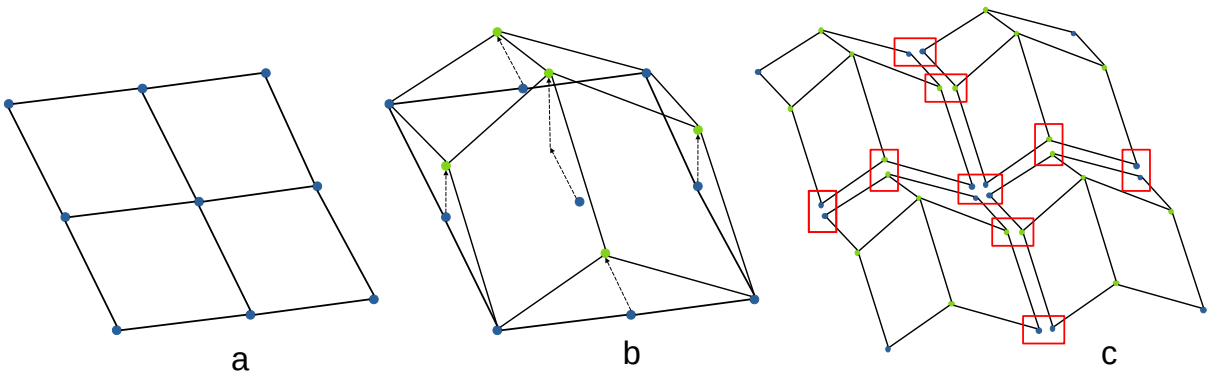


Figure 4: Simplified depiction of the unit cell generation and merging process

2.1.2 Transforming into foldable crease pattern

This newly formed origami representation, that is displayed in figure 3f, has two problems: it is not foldable and the origami facets are not planar. In other words, this origami pattern could never be folded out of a flat sheet of material. To overcome this, the design is passed through an optimization procedure that ensures the foldability and planarity of the origami while remaining close to the original target surface. The structure of this optimization procedure also follows Dudte et al. [39].

The design variables of this procedure are the x-, y-, and z-coordinates of the free nodes, which are marked green in figure 5. The blue nodes are fixed to enforce their contact with the target surface. That blue "side" of the origami will thus be the side that conforms to the target surface.

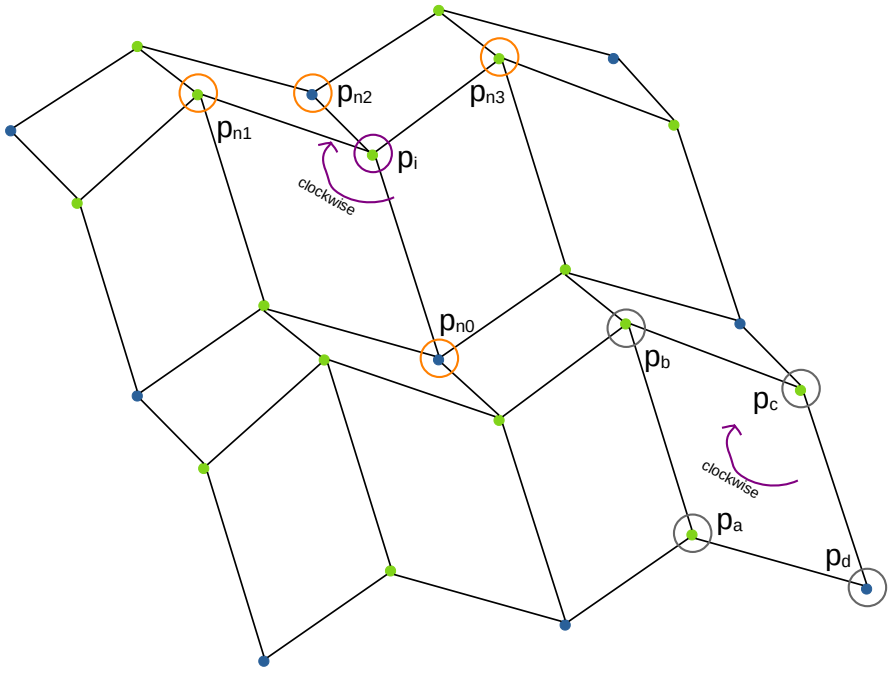


Figure 5: Pattern with clarifications for design variables and constraints

The objective function to minimize is

$$f = \sum_{i=0}^{N_{\text{edges}}-1} \frac{1}{2L_{0,i}}(L_i - L_{0,i})^2 \tag{1}$$

The function is the sum of the relative change in edge lengths, which include the edges that are seen in the pattern, but also the diagonals that can be constructed in every quadrilateral in the pattern. With this expression, the optimizer will try to keep the design as close as possible to the previously constructed origami. The sensitivity of the objective function can be found in appendix A.3.

To overcome the problems of non-foldability and non-planarity of the origami, two sets of constraints are introduced. Both can be expressed in terms of the vectors of the vertex locations of the origami.

The angles around every interior vertex must sum to 2π for the vertex to be foldable. That means there is a constraint for every interior vertex, for which the surrounding vertices are ordered clockwise. This is expressed as

$$g_{foldable,i} = 2\pi - \sum_{j=0}^{3} \angle(\boldsymbol{p_{n_j}} - \boldsymbol{p_i}, \boldsymbol{p_{n_{j+1}}} - \boldsymbol{p_i}) = 0 \tag{2}$$

with an example in the top-middle section of figure 5. The purple-circled node is the interior vertex and the orange-circled nodes are its neighbours. The angle between the vectors is calculated with the two-argument arctangent as in

$$\angle(\boldsymbol{v}, \boldsymbol{w}) = 2\ \mathrm{atan2}(\|\boldsymbol{v} \times \boldsymbol{w}\|, \|\boldsymbol{v}\|\,\|\boldsymbol{w}\| + \boldsymbol{v} \cdot \boldsymbol{w}) \tag{3}$$

The planarity constraint ensures that the four corner nodes of every facet of the origami are in the same plane. This is done with an equality constraint that expresses the tetrahedral volume of the facet:

$$g_{planar,i} = ((\boldsymbol{p_b} - \boldsymbol{p_a}) \times (\boldsymbol{p_c} - \boldsymbol{p_a})) \cdot (\boldsymbol{p_d} - \boldsymbol{p_a}) = 0 \tag{4}$$

when this equality is satisfied, the facet is planar. There is an example in the bottom-right corner of figure 5 with the grey-circled nodes. There is one constraint for every facet in the design.

The sensitivities of these constraints are available in appendix A.3.

The objective function and constraints, along with their sensitivities are used in a Byrd-Omojokun trust-region SQP optimization method [43, 44], which gives an optimal design that is close to its original form, but satisfies all constraints and is thus foldable from a flat piece of material.

2.1.3 Flattening the foldable crease pattern

The result of the above optimization process is a folded design. To advance with shape matching non-rigid origami in section 2.2, we need a flat design to start with. Transforming into a flat design starts with making one corner of the pattern lie in the xy-plane. Algorithm 1 shows a series of steps to make a quadrilateral lie in that plane, while keeping the relative distances between the corners unchanged. The vectors $\boldsymbol{p_0}, \boldsymbol{p_1}, \boldsymbol{p_2}, \boldsymbol{p_3}$ are the four corner nodes of the panel.

Algorithm 1 Steps to make a quadrilateral surface lie in the xy-plane.

$\boldsymbol{p_{01}} = (\boldsymbol{p_1} - \boldsymbol{p_0}) / \|\boldsymbol{p_1} - \boldsymbol{p_0}\|$
$\boldsymbol{p_{03}} = (\boldsymbol{p_3} - \boldsymbol{p_0}) / \|\boldsymbol{p_3} - \boldsymbol{p_0}\|$
$\boldsymbol{n} = (\boldsymbol{p_{01}} \times \boldsymbol{p_{03}}) / \|\boldsymbol{p_{01}} \times \boldsymbol{p_{03}}\|$
$\boldsymbol{z} = \begin{bmatrix} 0 & 0 & 1 \end{bmatrix}^\top$
$\boldsymbol{a} = \begin{bmatrix} a_x \\ a_y \\ a_z \end{bmatrix} = (\boldsymbol{n} \times \boldsymbol{z}) / \|\boldsymbol{n} \times \boldsymbol{z}\|$
$\theta = \arccos(\boldsymbol{n} \cdot \boldsymbol{z})$
$\boldsymbol{R} = \begin{bmatrix} \cos\theta + a_x^2(1-\cos\theta) & a_x a_y(1-\cos\theta) - a_z \sin\theta & a_x a_z(1-\cos\theta) + a_y \sin\theta \\ a_y a_x(1-\cos\theta) + a_z \sin\theta & \cos\theta + a_y^2(1-\cos\theta) & a_y a_z(1-\cos\theta) - a_x \sin\theta \\ a_z a_x(1-\cos\theta) - a_y \sin\theta & a_z a_y(1-\cos\theta) + a_x \sin\theta & \cos\theta + a_z^2(1-\cos\theta) \end{bmatrix}$
$\boldsymbol{p_{i,flat}} = \boldsymbol{R}\,\boldsymbol{p_i} \quad \forall\, i \in \{0,1,2,3\}$
$p_{i,flat,z} = 0 \quad \forall\, i \in \{0,1,2,3\}$

That first quadrilateral is the cornerstone to which all of the other panels will be attached. The panels lie in a grid of rows and columns. We start by attaching the first column to the cornerstone: going sequentially from

the panel adjacent to the corner, then to the one adjacent to that, etc. Figure 6 shows the sequence of working through the panels (first column is the blue arrow). The panels are connected to the previous panel by first making them lie in the xy-plane with algorithm 1, then rotating them in that plane so that they align with the previous panel by means of the steps in algorithm 2, and finally, the panel is translated such that it actually attaches to its predecessor. The subscripts r and l are for indicating the "root" and "leaf" panels, where the root panel is the one already in the correct place and the leaf panel is the one that is being aligned.

Algorithm 2 Steps to make a quadrilateral surface align with its predecessor in the column

$\boldsymbol{v_r} = (\boldsymbol{p_{r,2}} - \boldsymbol{p_{r,3}}) / \|\boldsymbol{p_{r,2}} - \boldsymbol{p_{r,3}}\|$
$\boldsymbol{v_l} = (\boldsymbol{p_{l,1}} - \boldsymbol{p_{l,0}}) / \|\boldsymbol{p_{l,1}} - \boldsymbol{p_{l,0}}\|$
$\theta = \arccos(\boldsymbol{v_r} \cdot \boldsymbol{v_l})$
if $(\boldsymbol{v_r} \times \boldsymbol{v_l})_z > 0$ **then**
  $\theta = -\theta$
$\boldsymbol{R} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$
$\boldsymbol{p_{l,i,aligned}} = \boldsymbol{R}\, \boldsymbol{p_{l,i}} \quad \forall\, i \in \{0, 1, 2, 3\}$

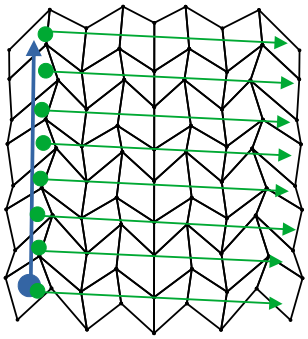


Figure 6: Sequence of the flattening process: first blue, then green

Once the first column is complete, it provides nucleations for every row. The panels in the rows are sequentially attached in a similar manner to the panels in the first column. Each panel is first rotated to the xy-plane (algorithm 1), then aligned (algorithm 3), and finally translated to attach to the previous panel. Again, see figure 6 for the sequence of panels, the green arrows are the rows.

Algorithm 3 Steps to make a quadrilateral surface align with its predecessor in the row

$\boldsymbol{v_r} = (\boldsymbol{p_{r,2}} - \boldsymbol{p_{r,1}}) / \|\boldsymbol{p_{r,2}} - \boldsymbol{p_{r,1}}\|$
$\boldsymbol{v_l} = (\boldsymbol{p_{l,3}} - \boldsymbol{p_{l,0}}) / \|\boldsymbol{p_{l,3}} - \boldsymbol{p_{l,0}}\|$
$\theta = \arccos(\boldsymbol{v_r} \cdot \boldsymbol{v_l})$
if $(\boldsymbol{v_r} \times \boldsymbol{v_l})_z > 0$ **then**
  $\theta = -\theta$
$\boldsymbol{R} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$
$\boldsymbol{p_{l,i,aligned}} = \boldsymbol{R}\, \boldsymbol{p_{l,i}} \quad \forall\, i \in \{0, 1, 2, 3\}$

Now that the first column and all rows are connected, we have a flat origami pattern that, when folded, has a side with nodes that conform to the target surface. The black pattern in figure 6 is such a crease pattern.

2.2 Shape matching with non-rigid origami

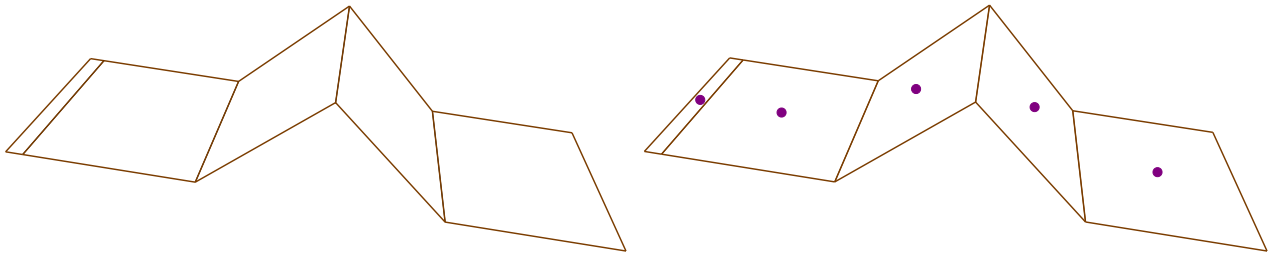
Starting from the rigid origami design, we work towards the more accurate, non-rigid origami design. The bar-and-hinge model is used to simulate folding and an optimization algorithm minimizes an objective function to find an optimized non-rigid design.

2.2.1 The bar-and-hinge model

We use the N5B8 bar-and-hinge model as developed by Filipov et al. [27] and implemented in the MERLIN2 software by Liu and Paulino [30], translated into Python for this work, see appendix A.1. What follows is a high-level description of the method and the adaptations made to it. In some parts there will be examples based on the simple "bump" configuration, a strip of material with a folded bump in the middle.

Placing bars, bends, and folds
As mentioned in the introduction, the bar-and-hinge model uses bars and rotational springs to represent the mechanics of origami elasticity. The N5B8 model starts with a mesh-like representation of the origami that is to be analysed, see figure 7a. In this model, only quadrilateral facets are allowed. The position of the fifth, or so called "Steiner" node of every facet, is computed as the intersection of the facet's diagonals. That is impossible if the panel is non-planar, in which case the fifth node is placed such that the panel is bent along the shorter diagonal. Figure 7b shows the Steiner nodes in purple. The bars of the bar-and-hinge model are then generated by connecting the nodes, including Steiner points.

At this point, the triangular trusses can freely rotate around the bars' axes, which is not representative of the structure we try to model. Therefore, hinges are introduced, which will resist rotation of the triangular trusses at either side of the hinge. The folding hinges are placed over the fold lines and the bending hinges are placed over the bend lines, see figure 8.



(a) Quadrilateral facets (b) Steiner points in purple

Figure 7: Insertion of Steiner points for the N5B8 approach

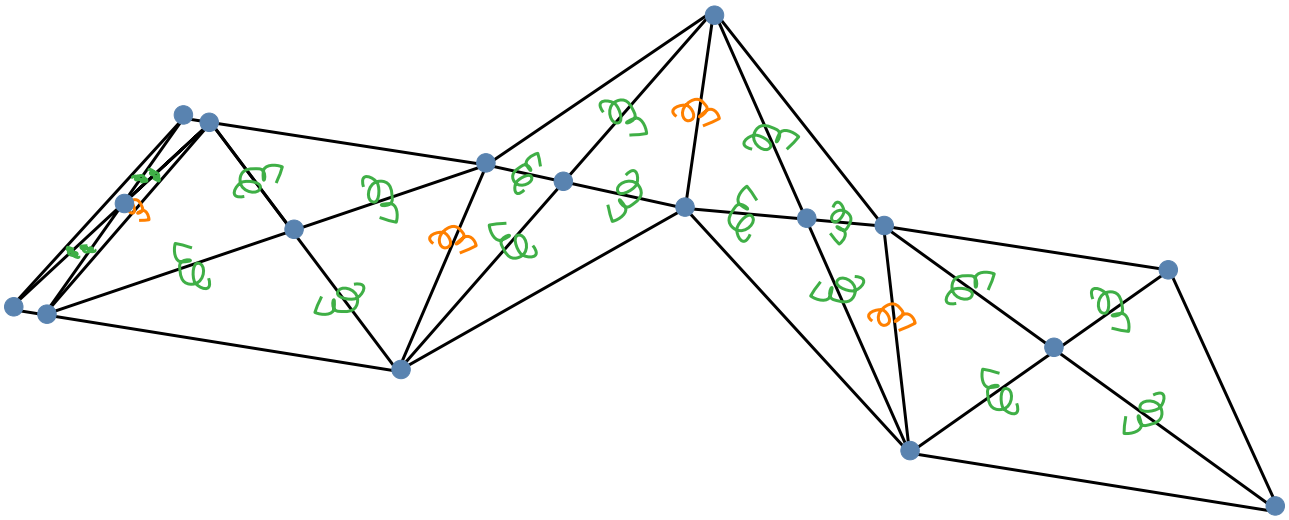


Figure 8: Nodes are blue, bars are black, folding hinges are orange, and bending hinges are green

The properties of the bars, bends, and folds are then derived from the material properties and thickness of the origami. The selected parameters are suitable for paper, which is useful for quick experimentation and validation. Paper's Young's modulus and Poisson's ratio are very much dependent on the loading direction and the type of paper [45], but, taking average values, we reach $E = 4\,\text{GPa}$, $\nu = 0.3$, and $t = 0.08\,\text{mm}$ [45].

Tangent stiffness matrix

For calculating the force-displacement relationship, we need the tangent stiffness matrix at every step of the incremental process. Every static analysis is defined by three equations. The first one is the continuity equation, which describes the relation between displacements and strains. The second equation is the material (constitutive) model, which relates deformations to stresses. Finally, the equilibrium equation gives a relation between the internal stresses and the externally applied loads.

In the calculation of the tangent stiffness matrix, these three equations together define a direct relation between the applied loads and the displacements of the origami in the form of $\boldsymbol{KU} = \boldsymbol{F}$. The bar-and-hinge model has three DOFs for every node—one DOF for each coordinate. The tangent stiffness matrix will thus have the shape $(3 \cdot N_{\text{nodes}} \times 3 \cdot N_{\text{nodes}})$ and is symmetric. The matrix contains a part that accounts for the bars' stiffnesses and a part that accounts for the bends' and folds' stiffnesses: $\boldsymbol{K} = \boldsymbol{K^b} + \boldsymbol{K^h}$.

We now look at a single bar and define its contribution $\boldsymbol{k^b}$ to the global bar stiffness matrix $\boldsymbol{K^b}$ with help of figure 9. The local matrix $\boldsymbol{k^b}$ with shape (6×6) can be split into four parts:

$$\boldsymbol{k^b} = \boldsymbol{k^b_{el}} + \boldsymbol{k^b_1} + \boldsymbol{k^b_2} + \boldsymbol{k^b_g} \tag{5}$$

where $\boldsymbol{k^b_{el}}$ is the linear stiffness matrix, $\boldsymbol{k^b_1} + \boldsymbol{k^b_2}$ together is the initial displacement matrix, and $\boldsymbol{k^b_g}$ is the geometric stiffness matrix [25]:

$$\begin{aligned}
\boldsymbol{k^b_{el}} &= E^b A^b L^b \boldsymbol{B_1}^\top \boldsymbol{B_1} \\
\boldsymbol{k^b_1} &= E^b A^b L^b ((\boldsymbol{B_2 u^b}) \boldsymbol{B_1} + \boldsymbol{B_1}^\top (\boldsymbol{B_2 u^b})^\top) \\
\boldsymbol{k^b_2} &= E^b A^b L^b (\boldsymbol{B_2 u^b})(\boldsymbol{B_2 u^b})^\top \\
\boldsymbol{k^b_g} &= S^b_x A^b L^b \boldsymbol{B_2}
\end{aligned} \tag{6}$$

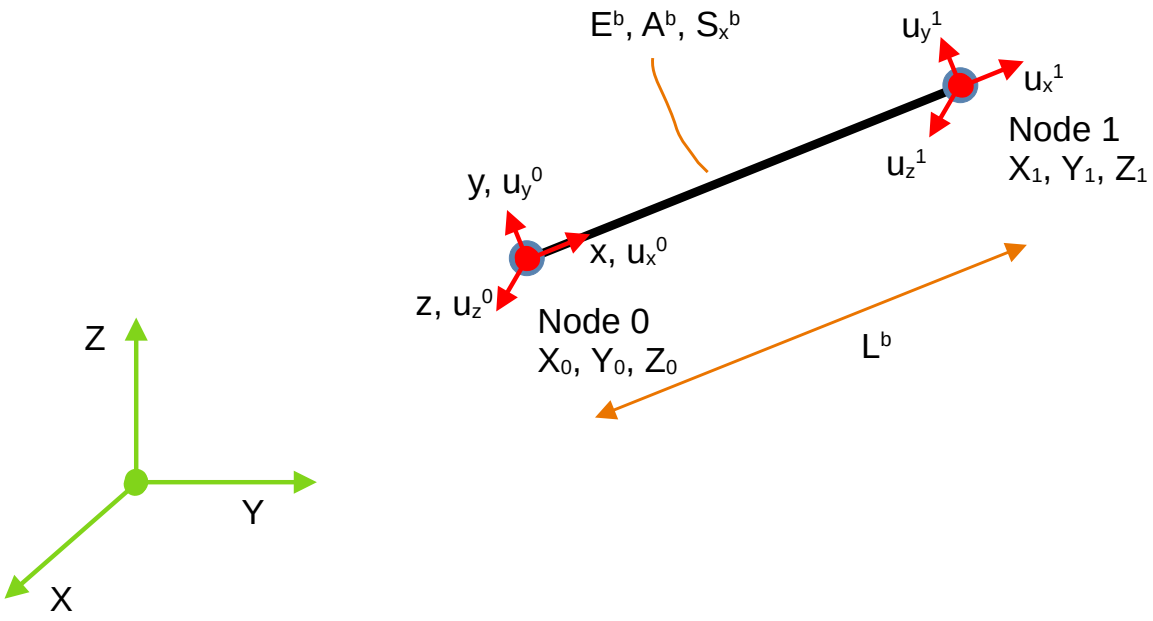


Figure 9: A bar in the global and local coordinate system

E^b and A^b are the tangent modulus and section area, which are derived from the material properties and thickness of the sheet of material that is modelled. L^b is simply the length of the bar, and S_x^b is the internal stress in the direction of the bar's longitudinal axis. Moreover, the vector $\boldsymbol{u^b}$ contains the displacements of the two connected nodes in the form $\begin{bmatrix} u_x^0 & u_y^0 & u_z^0 & u_x^1 & u_y^1 & u_z^1 \end{bmatrix}^\top$, expressed in the local coordinate system, see figure 9. Finally, $\boldsymbol{B_1} = \begin{bmatrix} -1 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}$, and $\boldsymbol{B_2} = \frac{1}{(L^b)^2} \begin{bmatrix} 1 & 0 & 0 & -1 & 0 & 0 \\ 0 & 1 & 0 & 0 & -1 & 0 \\ 0 & 0 & 1 & 0 & 0 & -1 \\ -1 & 0 & 0 & 1 & 0 & 0 \\ 0 & -1 & 0 & 0 & 1 & 0 \\ 0 & 0 & -1 & 0 & 0 & 1 \end{bmatrix}$ [25].

To assemble the contributions of all individual bars and place them into the global stiffness matrix, they need to be transformed to the global coordinate system by substituting $\tilde{\boldsymbol{B_1}} = \frac{1}{L^b} \left[-\left(\frac{\boldsymbol{X_1}-\boldsymbol{X_0}}{L^b}\right)^\top \left(\frac{\boldsymbol{X_1}-\boldsymbol{X_0}}{L^b}\right)^\top \right]$ in $\boldsymbol{B_1}$'s place in equation 6, where $\boldsymbol{X_0} = \begin{bmatrix} X_0 & Y_0 & Z_0 \end{bmatrix}^\top$ and $\boldsymbol{X_1} = \begin{bmatrix} X_1 & Y_1 & Z_1 \end{bmatrix}^\top$, which are the global coordinates of the nodes at the ends of the bar [25]. Finally, the (6×6) matrix $\boldsymbol{k_b}$ can be placed in the global stiffness matrix for the bars, $\boldsymbol{K^b}$, in the same way as in the standard finite element method, taking the relevant DOFs into account.

With help of figure 10, we will now look at how the stiffness matrix for the hinges that represent the bends and folds—$\boldsymbol{K^h}$—is built. In essence, the bends and folds work the same way, only their rotational stiffnesses differ. The stiffness matrix of a single hinge has shape (12×12) and is expressed in

$$\boldsymbol{k^h} = \frac{dM}{d\theta} \frac{d\theta}{d\boldsymbol{x^h}} \otimes \frac{d\theta}{d\boldsymbol{x^h}} + M \frac{d^2\theta}{(d\boldsymbol{x^h})^2} \tag{7}$$

from Liu and Paulino's work [25]. M is the hinge moment, θ is the dihedral angle (the angle between the two facets) and $\frac{dM}{d\theta}$ is the tangent rotational stiffness, defined in the constitutional model. There are four relevant nodes for a single hinge, illustrated as nodes i, j, k, and l. The derivative of the dihedral angle to the nodes' coordinates, i.e. the Jacobian $\frac{d\theta}{d\boldsymbol{x^h}}$ is expressed as $\begin{bmatrix} \frac{\partial\theta}{\partial \boldsymbol{x_i}} \\ \frac{\partial\theta}{\partial \boldsymbol{x_j}} \\ \frac{\partial\theta}{\partial \boldsymbol{x_k}} \\ \frac{\partial\theta}{\partial \boldsymbol{x_l}} \end{bmatrix}$, and the Hessian $\frac{d^2\theta}{(d\boldsymbol{x^h})^2}$ takes the form

$\begin{bmatrix} \frac{\partial^2\theta}{\partial \boldsymbol{x}_i^2} & \frac{\partial^2\theta}{\partial \boldsymbol{x}_i \partial \boldsymbol{x}_j} & \frac{\partial^2\theta}{\partial \boldsymbol{x}_i \partial \boldsymbol{x}_k} & \frac{\partial^2\theta}{\partial \boldsymbol{x}_i \partial \boldsymbol{x}_l} \\ \frac{\partial^2\theta}{\partial \boldsymbol{x}_j \partial \boldsymbol{x}_i} & \frac{\partial^2\theta}{\partial \boldsymbol{x}_j^2} & \frac{\partial^2\theta}{\partial \boldsymbol{x}_j \partial \boldsymbol{x}_k} & \frac{\partial^2\theta}{\partial \boldsymbol{x}_j \partial \boldsymbol{x}_l} \\ \frac{\partial^2\theta}{\partial \boldsymbol{x}_k \partial \boldsymbol{x}_i} & \frac{\partial^2\theta}{\partial \boldsymbol{x}_k \boldsymbol{x}_j} & \frac{\partial^2\theta}{\partial \boldsymbol{x}_k^2} & \frac{\partial^2\theta}{\partial \boldsymbol{x}_k \partial \boldsymbol{x}_l} \\ \frac{\partial^2\theta}{\partial \boldsymbol{x}_l \partial \boldsymbol{x}_i} & \frac{\partial^2\theta}{\partial \boldsymbol{x}_l \boldsymbol{x}_j} & \frac{\partial^2\theta}{\partial \boldsymbol{x}_l \boldsymbol{x}_k} & \frac{\partial^2\theta}{\partial \boldsymbol{x}_l^2} \end{bmatrix}$. For the formulas for the entries in the Jacobian and Hessian, please consult appendix A.4.

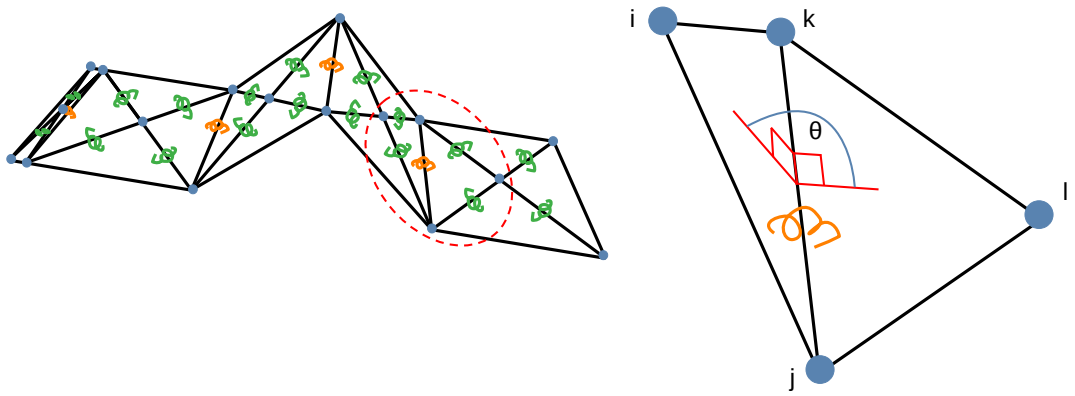


Figure 10: A hinge and the relevant nodes for calculating stiffness

Since $\boldsymbol{k^h}$ is directly expressed with respect to global coordinates, its contribution to the global hinge stiffness matrix $\boldsymbol{K^h}$ can easily be inserted. And combined with the previously discussed bar stiffness matrix, we can compute the full system stiffness matrix as $\boldsymbol{K} = \boldsymbol{K^b} + \boldsymbol{K^h}$.

The arc-length path-following method

With linear systems, a single evaluation of the stiffness matrix is sufficient to know the entire equilibrium path. Elastic origami deformation, however, is a highly nonlinear process, involving both geometrical and material nonlinearities. To solve for such a nonlinear equilibrium path, we need a method that evaluates the stiffness matrix multiple times to build the path step-by-step. In this work, we use the arc-length method, which is an advanced incremental-iterative procedure that can track negative stiffness and snap-back behaviour. For a wider-scope introduction into path-following methods, see appendix A.5.

Where the standard Newton method uses

$$\boldsymbol{dU_{t,j}} = \boldsymbol{K_{t,j}^{-1}} \left(\boldsymbol{F_t^{ext}} - \boldsymbol{F_{t,j}^{int}} \right) \tag{8}$$

for the iterative change in displacement, the arc-length method uses a formula that splits the displacement step into two parts:

$$\begin{aligned} \boldsymbol{dU_{t,j}} &= \boldsymbol{dU_{t,j}^{I}} + d\lambda_{t,j} \cdot \boldsymbol{dU_{t,j}^{II}} \\ \boldsymbol{dU_{t,j}^{I}} &= \boldsymbol{K_{t,j}^{-1}} \left(\boldsymbol{F_{t,j}^{ext}} - \boldsymbol{F_{t,j}^{int}} \right) \\ \boldsymbol{dU_{t,j}^{II}} &= \boldsymbol{K_{t,j}^{-1}} \boldsymbol{F} \end{aligned} \tag{9}$$

where t is the increment counter, j is the iteration counter, λ is the load factor, and $\boldsymbol{F}$ is the nominal external load—it is customary for this to be a unit load, but that is not necessary. The external load at a particular iteration is computed as $\boldsymbol{F_{t,j}^{ext}} = \lambda_{t,j} \cdot \boldsymbol{F}$.

A key difference with Newton's method is that the load factor step is not fixed, but is an extra variable that changes at every iteration. Since we have an extra variable, we also need an extra equation to get a unique solution:

$$\boldsymbol{\Delta U_{t,j}^{\top}} \boldsymbol{\Delta U_{t,j}} + \beta^2 d\lambda_{t,j}^2 \boldsymbol{F}^{\top} \boldsymbol{F} = \Delta l^2 \tag{10}$$

which constrains the length (or L_2-norm) of the incremental displacement vector step, augmented with the incremental load factor step [46]. Δl is the length of the augmented vector, $\boldsymbol{\Delta U_{t,j}}$ is the incremental displacement step, and β is a value that weighs the importance of the incremental load factor step in that augmented vector. The symbols Δ and d convey an incremental and iterative step, respectively.

Equation 10 gives two solutions for the iterative load factor step. Luckily, we can choose the solution that best aligns with the direction of the previous increment. It is also possible, however, for the solutions to become imaginary—in the sense of $\sqrt{-1}$—at which point we need to decrease the previous step size [47]. To avoid such workarounds, we choose to linearize equation 10 to

$$\boldsymbol{\Delta U_{t,j-1}^{\top}} \boldsymbol{\Delta U_{t,j}} + \beta^2 d\lambda_{t,j-1} d\lambda_{t,j} \boldsymbol{F}^{\top} \boldsymbol{F} = \Delta l^2 \tag{11}$$

as described by Ramm [48]. With the help of equation 9, we can rewrite equation 11 to an explicit expression for $d\lambda_{t,j}$:

$$d\lambda_{t,j} = \frac{\Delta l^2 - \boldsymbol{\Delta U_{t,j-1}^{\top}} \boldsymbol{\Delta U_{t,j-1}} - \boldsymbol{\Delta U_{t,j-1}^{\top}} \boldsymbol{dU_{t,j}^{I}}}{\boldsymbol{\Delta U_{t,j-1}^{\top}} \boldsymbol{dU_{t,j}^{II}} + \beta^2 d\lambda_{t,j-1} \boldsymbol{F}^{\top} \boldsymbol{F}} \tag{12}$$

Now, since $\Delta l \approx \boldsymbol{\Delta U_{t,j-1}^{\top}} \boldsymbol{\Delta U_{t,j-1}}$, and since we are linearizing anyway, we insert the expression into equation 12 and get

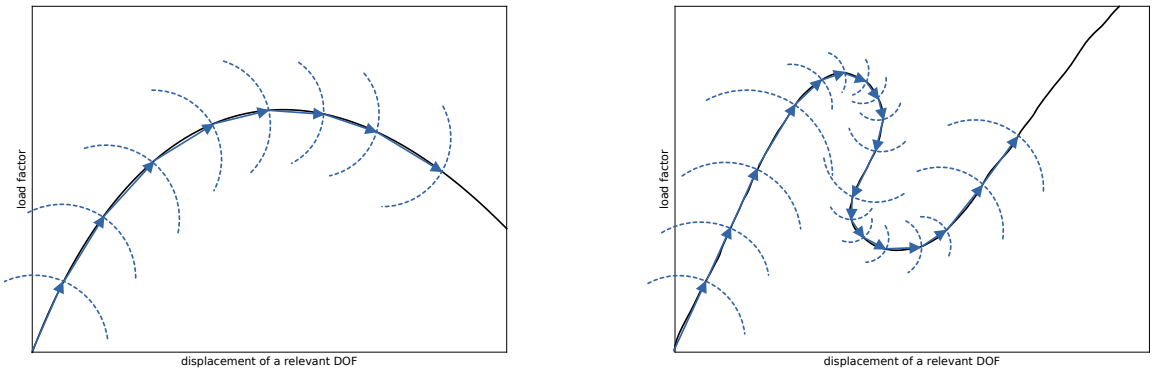
$$d\lambda_{t,j} = -\frac{\boldsymbol{\Delta U_{t,j-1}^{\top}} \boldsymbol{dU_{t,j}^{I}}}{\boldsymbol{\Delta U_{t,j-1}^{\top}} \boldsymbol{dU_{t,j}^{II}} + \beta^2 d\lambda_{t,j-1} \boldsymbol{F}^{\top} \boldsymbol{F}} \tag{13}$$

which is greatly simplified with Δl even completely removed. Finally, de Borst and Sluys write: "Numerical experience indicates that the value of β does not seem to influence the performance of the method very much. In fact, the simple version $\beta = 0$ seems to show the most robust behaviour." [47]. We thus use $\beta = 0$ and reach

$$d\lambda_{t,j} = -\frac{\boldsymbol{\Delta U_{t,j-1}^{\top}} \boldsymbol{dU_{t,j}^{I}}}{\boldsymbol{\Delta U_{t,j-1}^{\top}} \boldsymbol{dU_{t,j}^{II}}} \tag{14}$$

There are some additional considerations that can improve the arc-length method. First, there is the choice of initial load factor step, which can influence which of multiple possible equilibrium paths the method will take. Secondly, there is the possibility to change the load factor step while the algorithm is running; when there are very many iterations within an increment, that means that convergence is slow or even not happening at all, which is a sign that the load step is too large. We can set a maximum to the number of iterations, and if this maximum is exceeded, we can reduce the load factor step and reset to the beginning of the increment to try again. And there is the opposite case: if it takes very few (i.e. one or two) iterations to converge, we are wasting computing power and we should increase the load factor step as we go to the next increment. Thirdly, suppose we are not just looking for the general equilibrium path, but are instead interested in the displacement field at a specific load magnitude. We are unlikely to reach this specific load if the load factor step is unpredictable. The problem is that we might overshoot this specific load magnitude and thus never get to know the displacements at that specific load. A solution to that problem is to take the increment that overshot, calculate the fraction of the load factor step that overshoots the target load factor, and finally, subtract an equal fraction of the final displacement step from the displacement vector to come to the displacement vector at the target load factor. On a different note, another choice we can make is the value for the tolerance on the residual forces to reach zero. That choice will control the number of iterations during each step, which in turn influences the speed and accuracy of the solution. Finally, the calculations of the tangent stiffness matrix and the internal forces share a lot of computationally expensive steps, and it is thus beneficial to combine the two calculations. This requires some reordering of the algorithm, which reduces understandability but increases the speed.

Figure 11 shows how the arc-length procedure traverses negative stiffness and snap-back equilibrium paths, and algorithm 4 is a complete arc-length procedure.



(a) An equilibrium path with negative stiffness

(b) Snap-back behaviour

Figure 11: Two extraordinary equilibrium paths that are tracked with the arc-length method

In MERLIN2, the authors implemented a linearization that transformed equation 11 into

$$\boldsymbol{\Delta U_{t,0}^{\top}} \boldsymbol{\Delta U_{t,j}} + \beta^2 d\lambda_{t,0} d\lambda_{t,j} \boldsymbol{F}^{\top} \boldsymbol{F} = \Delta l^2 \tag{15}$$

which leads to

$$d\lambda_{t,j} = -\frac{\boldsymbol{\Delta U_{t,0}^{\top}} \boldsymbol{dU_{t,j}^{I}}}{\boldsymbol{\Delta U_{t,0}^{\top}} \boldsymbol{dU_{t,j}^{II}}} \tag{16}$$

as the expression for the iterative load factor step. The arc-length method implemented in this work was rewritten and improved by removing said linearization from the MERLIN2 implementation. This was necessary because the linearization led to convergence problems during the calculation of the equilibrium path of the origami saddle.

Algorithm 4 Pseudo-code of the arc-length method

$t = 0$ # *increment counter*
$j = 0$ # *iteration counter*
$\boldsymbol{U_{t,0}} = \mathbf{0}$ # *initial displacement is zero*
$\lambda_{t,j} = 0$ # *initial load factor is zero*
$d\lambda_{1,0} = 0.01$ # *initial load factor step*
$\text{tol} = 0.00001$ # *tolerance of residual loads*

while $\lambda_{t,j} < 1$ # *until we reach a load factor of 1*
  $t = t + 1$ # *increase increment counter*
  $\boldsymbol{U_{t,0}} = \boldsymbol{U_{t-1,j}}$ # *set first displacement of this incr. to the final value of the previous incr.*
  $\lambda_{t,0} = \lambda_{t-1,j}$ # *set first load factor of this incr. to the final value of the previous incr.*

  $j = 0$ # *reset iteration counter*
  $\boldsymbol{\Delta U_{t,0}} = \mathbf{0}$ # *set incremental displacement step to zero*
  $\Delta\lambda_{t,0} = 0$ # *set incremental load factor step to step*

  $\boldsymbol{F_{t,j}^{ext}} = \lambda_{t,j} \cdot \boldsymbol{F}$ # *external load*
  calculate $\boldsymbol{F_{t,j}^{int}}(\boldsymbol{U_{t,j}})$ # *internal forces*
  $\boldsymbol{R_{t,j}} = \boldsymbol{F_{t,j}^{ext}} - \boldsymbol{F_{t,j}^{int}}$ # *residual load*
  $\text{err} = 1000$ # *any large number*

  while $\text{err} > \text{tol} \cdot \left\| \boldsymbol{F_{t,j}^{ext}} \right\|$ # *until the residual loads are almost zero*
    calculate $\boldsymbol{K_{t,j}}(\boldsymbol{U_{t,j}})$ # *stiffness matrix at last force-displacement point*
    $\boldsymbol{dU_{t,j}^{I}} = \boldsymbol{K_{t,j}^{-1}} \boldsymbol{R_{t,j}}$ # *first part of the displacement step*
    $\boldsymbol{dU_{t,j}^{II}} = \boldsymbol{K_{t,j}^{-1}} \boldsymbol{F}$ # *second part of the displacement step*

    if $j == 0$ **then** # *if we are in the first iteration of an increment*
      if $t == 1$ **then** # *if we are also in the first increment*
        $d\lambda_{t,j} = d\lambda_{1,0}$ # *choose the user-given initial load factor*
      else # *if we are not in the first increment*
        $d\lambda_{t,j} = \text{sign}(\boldsymbol{\Delta U_{t-1}^{\top}} \boldsymbol{dU_{t,j}^{II}}) \cdot \sqrt{\frac{\text{abs}(\Delta\lambda_{t-1} \cdot \boldsymbol{F}^{\top} \boldsymbol{\Delta U_{t-1}})}{abs(\boldsymbol{F}^{\top} \boldsymbol{dU_{t,j}^{II}})}}$ # *from De Borst and Sluys [47], page 46*
    else
      $d\lambda_{t,j} = -\frac{\boldsymbol{\Delta U_{t,j}^{\top}} \boldsymbol{dU_{t,j}^{I}}}{\boldsymbol{\Delta U_{t,j}^{\top}} \boldsymbol{dU_{t,j}^{II}}}$ # *equation 14, iteration number changed because of program's structure*

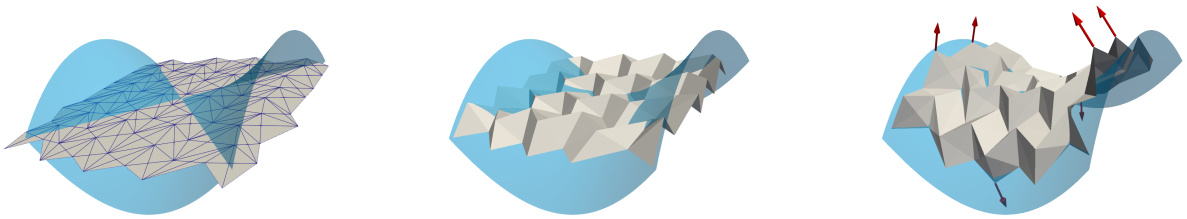
    $\boldsymbol{dU_{t,j}} = \boldsymbol{dU_{t,j}^{I}} + d\lambda_{t,j} \boldsymbol{dU_{t,j}^{II}}$ # *add up contributions to iterative displacement step*

    $j = j + 1$ # *increase iteration counter*
    $\boldsymbol{\Delta U_{t,j}} = \boldsymbol{\Delta U_{t,j-1}} + \boldsymbol{dU_{t,j-1}}$ # *update incremental displacement step with iterative step*
    $\boldsymbol{U_{t,j}} = \boldsymbol{U_{t,j-1}} + \boldsymbol{dU_{t,j-1}}$ # *update displacement with iterative step*
    $\Delta\lambda_{t,j} = \Delta\lambda_{t,j-1} + d\lambda_{t,j-1}$ # *update incremental load factor step with iterative step*
    $\lambda_{t,j} = \lambda_{t,j-1} + d\lambda_{t,j-1}$ # *update load factor with iterative step*

    calculate $\boldsymbol{F_{t,j}^{int}}(\boldsymbol{U_{t,j}})$ # *internal forces at new force-displacement point*
    $\boldsymbol{F_{t,j}^{ext}} = \lambda_{t,j} \cdot \boldsymbol{F}$ # *external load at new force-displacement point*
    $\boldsymbol{R_{t,j}} = \boldsymbol{F_{t,j}^{ext}} - \boldsymbol{F_{t,j}^{int}}$ # *residual loads at new force-displacement point*
    $\text{err} = \|\boldsymbol{R_{t,j}}\|$ # *magnitude of the residual load*

  $\boldsymbol{\Delta U_t} = \boldsymbol{\Delta U_{t,j}}$ # *save incremental displacement step for $d\lambda$ calculation*
  $\Delta\lambda_t = \Delta\lambda_{t,j}$ # *save incremental load factor step for $d\lambda$ calculation*

The bar-and-hinge model and arc-length method are used to calculate the equilibrium paths of origami designs. First, the flat design (figure 12a) is compressed such that it gets its pre-folded Miura-ori structure (figure 12b). After this initial folding, new loads are applied that fold the design into its target configuration (figure 12c), which can be judged on its closeness to the target surface, illustrated here by a saddle.



(a) An initial design with the fold and bend lines marked in blue

(b) A design that has undergone initial folding towards a Miura-ori structure

(c) A design that has been folded to the target surface (loads in red)

Figure 12: Three stages in the analysis of a design

There are examples of analyses that display the capabilities of the bar-and-hinge model and arc-length method in appendix A.10.

2.2.2 Objective function

From the bar-and-hinge model and the method for discovering the equilibrium path, we now move to defining a measure of "quality" of a design. For the purpose of shape-matching, we consider one side of the deformed origami surface and change the design such that that surface is as close as possible to our target surface, which is the same as was done in the rigid origami part of the method. The non-zero Gaussian curvature shape that is a saddle will be used as an example here, but the results will also include a dome and an arch. Now, as we have seen, a saddle can be defined as

$$z = \left(\frac{x}{a}\right)^2 - \left(\frac{y}{b}\right)^2 \tag{17}$$

with parameters a and b giving the magnitude of the curvature.

We define the bottom side of the origami (figure 13) as the side that must be as close as possible to this mathematical saddle. In other words, we want to minimize the maximum distance between the bottom of the origami and the saddle; the maximum distance is defined as

$$d_{max} = \max\left(\left\|\left(\frac{x_i}{a}\right)^2 - \left(\frac{y_i}{b}\right)^2 - z_i\right\|\right) \;\forall\, i \in I_b \tag{18}$$

where I_b is the set that contains the indices of the bottom nodes. This distance function is not smooth and, as we will see in the results section, smooth functions can perform better in the optimization process than non-smooth functions. We will thus smoothen this function with the so-called p-norm [49] and define the objective function as

$$f = \left(\sum_{i=0}^{N-1}\left\|\left(\frac{x_i}{a}\right)^2 - \left(\frac{y_i}{b}\right)^2 - z_i\right\|^p\right)^{\frac{1}{p}} \tag{19}$$

where N is the number of nodes in the bottom of the origami, and p is a free parameter, which we will experiment with to find the best-performing value. The smooth equation 19 approximates the non-smooth equation 18 better as $p \to \infty$. But minimizing the objective function with lower p's will *also* minimize the function that we actually want to minimize, which is d_{max}, and "push" the bottom nodes closer and closer to the saddle, achieving our goal of making an origami design match a target surface.

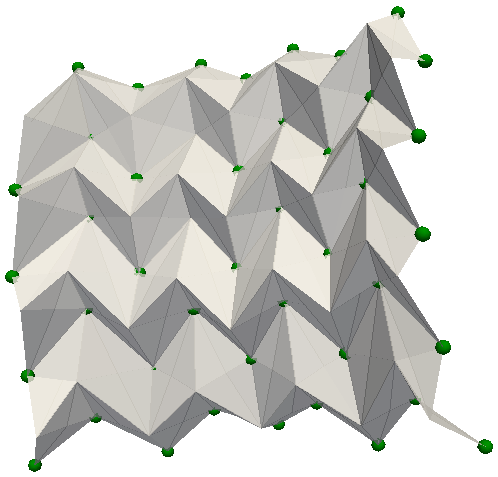
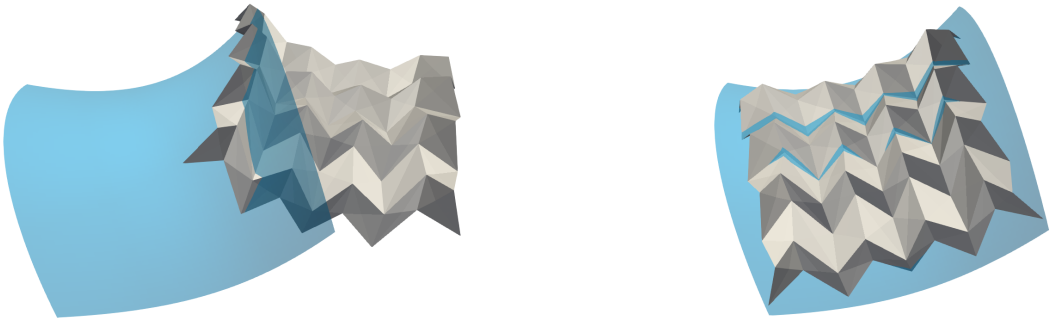


Figure 13: The nodes that are part of the "bottom" of the origami are marked green.

A folded origami design can be misaligned and misplaced such that it does not conform to the target surface, even if it matches the surface very well (figure 14a). This rigid body motion difference must be filtered out for the objective function to be effective in calculating the quality of a design. This filter will consist of three steps: a rigid body translation to center the design on the origin, a rigid body rotation to give the design the correct orientation, and finally another translation to make the bottom of the design touch the target surface, resulting in a well-placed design such as in figure 14b.



(a) A misplaced design

(b) A well-placed design

Figure 14: A design before and after filtering out the rigid body motion

The target shape's middle point should be located on the origin—this is beneficial because it maintains symmetry. We can then start by also centering the origami onto the origin. The origami design is simply a set of nodes, connected by lines to form panels. We calculate the center of mass of this set of nodes by taking their average position, and subsequently subtract the center of mass' position from all nodes, which results in a design that is centered onto the origin. This method will not work for all origami structures and target surfaces, but it will in this Miura-ori case that targets a saddle, dome, or arch surface. For a continuum approach to the calculation of the center of mass, see appendix A.6.

With the origami centered onto the origin, we will change its orientation such that it aligns with the target shape. We do this by applying a rotation matrix to its nodes. To calculate this rotation matrix, we start with the symmetric pseudo-inertia tensor $\boldsymbol{I}$, adapted from Vallery and Schwab [50]:

$$\boldsymbol{I} = \begin{bmatrix} I_{xx} & I_{xy} & I_{xz} \\ I_{yx} & I_{yy} & I_{yz} \\ I_{zx} & I_{zy} & I_{zz} \end{bmatrix} = \sum_{i=0}^{N_{\text{nodes}}-1} \begin{bmatrix} y_i^2 + z_i^2 & -x_i \cdot y_i & -x_i \cdot z_i \\ -y_i \cdot x_i & x_i^2 + z_i^2 & -y_i \cdot z_i \\ -z_i \cdot x_i & -z_i \cdot y_i & x_i^2 + y_i^2 \end{bmatrix} \tag{20}$$

It is a pseudo-inertia tensor because it does not calculate the actual inertia of the origami design. Instead, it

calculates the inertia tensor of the set of nodes in the bar-and-hinge model, assigning them all a unit mass. Since we are using this pseudo-inertia tensor only for geometric orientation, this approach is sufficient and we will simply call it the inertia tensor. For a continuum approach to the calculation of the inertia tensor, see appendix A.7.

From the inertia tensor, we calculate the principle axes of inertia by means of the following eigenvalue problem: $\boldsymbol{I}\boldsymbol{v} = \lambda\boldsymbol{v}$, where $\boldsymbol{v}$ is a principle axis and λ is a principle moment of inertia. With our three-dimensional design, there are three eigenvector solutions to this equation: $\boldsymbol{v_0}$, $\boldsymbol{v_1}$, and $\boldsymbol{v_2}$. And by using the previously laid out rigid origami method for the initial design, the design's orientation is already close to that of the target surface (see appendix A.8 for a more robust method that is to be used when this is not the case). That allows us to easily order the principle axes such that the first one aligns most with the x-axis, the second one with the y-axis, and the third one with the z-axis. As an example, axes $\boldsymbol{v_0} = \begin{bmatrix}0.150\\0.966\\0.211\end{bmatrix}$, $\boldsymbol{v_1} = \begin{bmatrix}-0.274\\-0.164\\0.948\end{bmatrix}$, and $\boldsymbol{v_2} = \begin{bmatrix}0.950\\-0.200\\0.240\end{bmatrix}$ will be ordered as $\boldsymbol{v_2}$, $\boldsymbol{v_0}$, $\boldsymbol{v_1}$. The rotation matrix that we were looking for is then constructed as $\boldsymbol{R} = \begin{bmatrix}\boldsymbol{v_2} & \boldsymbol{v_0} & \boldsymbol{v_1}\end{bmatrix}^\top$. We apply this rotational transformation to all of the nodes' displaced positions and obtain a design that is aligned with the target surface. If any principle axis points in the negative direction, it needs to be flipped during the construction of $\boldsymbol{R}$, otherwise unwanted reflections occur.

The objective function measures how close the bottom side of the origami is to the target surface. The final step in filtering out rigid body motions is thus to translate this bottom towards the target shape. Because the middle point of all target surfaces discussed is at $(0, 0, 0)$, we can take the middle of the bottom of the origami and translate that towards the origin. The most appropriate translation is generated when we define the middle by taking the average position of the three nodes in the center of the middle row.

During the optimization, some final rigid body motion parameters are optimized to further filter out rigid body motion differences between the folded design and the target surface. These parameters are: the rotation around the x-axis, the translation in y-direction, and the translation in z-direction.

2.2.3 Optimization algorithm

We consider a 4-by-4 Miura-ori unit cell origami design, as depicted in figure 13. During optimization, we approach an optimized design by determining design variables that minimize our objective function. We begin by listing the design variables:

- end points of the fold lines in the flat sheet of material;
- pitch (rotation around the x-axis);
- translation in y-direction;
- translation in z-direction;

We have been referring to the end points of the fold lines simply as *nodes*. The design's symmetry can be used to our advantage and allows us to only turn a little bit more than half of the design's nodes into design variables. To come to a full model, we simply reflect the coordinates over the yz-plane. These coordinates are the largest set of design variables, so this reduces the design space by a factor of almost two. In the case of the 4-by-4 Miura-ori design, this creates a total of $2 \cdot 36 + 9 + 3 = 84$ design variables, where the $2 \cdot 36$ comes from the x- and y-coordinates of the nodes to the left of the symmetry line, the 9 comes from the y-coordinates of the nodes on the symmetry line (the middle column), and the final three design variables are for filtering out the final rigid body motions that misplace the origami with respect to the target surface.

The general form of any optimization cycle looks like figure 15, where $\boldsymbol{x}$ is the vector of design variables, f is the objective function, $\boldsymbol{g}$ is the vector of inequality constraints, and $\boldsymbol{h}$ is the vector of equality constraints. In our case, there are no constraints (except for bounds on the design variables) and the model does not calculate any derivatives. The design variables are continuous, as opposed to discrete, and we have a single objective function that is nonlinear and non-convex. The model in our case is the combination of the N5B8 bar-and-hinge model, the arc-length method and the rigid body motion filter. Regarding the optimization algorithm, there is a wide range of algorithms available in engineering optimization. 2^{nd}-order methods use the objective function's value, as well as its first and second derivatives to determine a new set of design variables. When derivatives are difficult to compute, one can choose to compute them numerically using a finite difference scheme, but this

is normally slower than an analytical expression. Taking one step down, 1^{st}-order methods use the objective function's value and only its first derivative. Finally, in 0^{th}-order methods no derivatives are used at all and only the objective function's value is used. In appendix A.15, an attempt at using a 1^{st}-order method is discussed, but the optimizer that was ultimately selected is the 0^{th}-order Nelder-Mead simplex method [51].

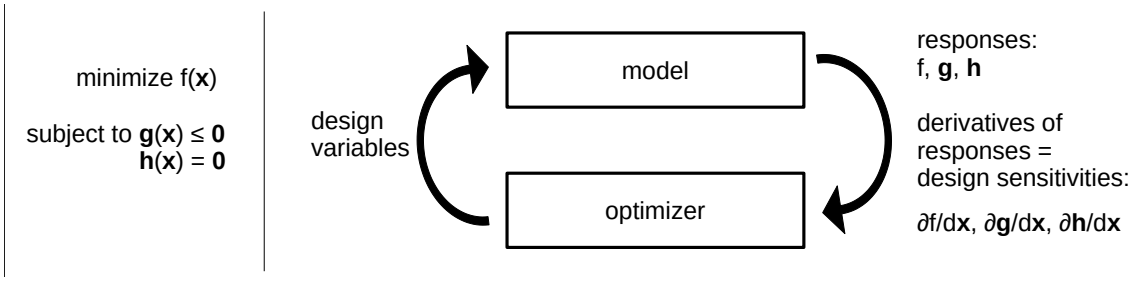


Figure 15: General depiction of an optimization cycle

The Nelder-Mead simplex method, as the name implies, uses a simplex to search the design space and find the minimum of the objective function. A simplex is a shape that consists of $n+1$ vertices, where n is the number of dimensions of the function's argument; this simplex still exists in the n-dimensional space. In the case of an objective function that takes two design variables, $n = 2$, the simplex is a triangle that lives on a plane, which is the example by which the method will be explained here. Figure 16 shows a diagram of the Nelder-Mead method. We start in the top-left corner of the diagram, where the simplex is formed around the initial design, and the designs $\boldsymbol{u}$, $\boldsymbol{v}$, and $\boldsymbol{w}$ are named such that $f(\boldsymbol{u}) < f(\boldsymbol{v}) < f(\boldsymbol{w})$. The worst design $\boldsymbol{w}$ is "reflected" through the middle point between $\boldsymbol{u}$ and $\boldsymbol{v}$ to reach $\boldsymbol{r}$. The objective function at $\boldsymbol{r}$ is then evaluated and based on that result, one of three directions is taken. If $\boldsymbol{r}$ performs better than $\boldsymbol{v}$, but not better than $\boldsymbol{u}$, we take $\boldsymbol{r}$ as the third point that, together with $\boldsymbol{u}$ and $\boldsymbol{v}$, forms the simplex for the next iteration, marked in green. That was the leftmost option in the diagram. Conversely, if $\boldsymbol{r}$ outperforms both $\boldsymbol{v}$ and $\boldsymbol{u}$, we take the middle route. Because the direction of the reflected point gave a better design, the reflected point is extended and a new point, $\boldsymbol{e}$, is generated. If $f(\boldsymbol{e})$ is lower than $f(\boldsymbol{r})$, $\boldsymbol{e}$ will be the third vertex of the new simplex; otherwise, $\boldsymbol{r}$ will simply take that place. In the unfortunate event that $\boldsymbol{r}$ does not outperform $\boldsymbol{u}$ or $\boldsymbol{v}$, the rightmost route is taken. There, the vector from $\boldsymbol{w}$ to $\boldsymbol{r}$ is contracted to a quarter and three quarters of its length to form designs $\boldsymbol{c_i}$ and $\boldsymbol{c_o}$, respectively. If one or both of those designs outperform $\boldsymbol{v}$, the best of $\boldsymbol{c_i}$ and $\boldsymbol{c_o}$ is chosen as part of the next simplex. Finally, if $\boldsymbol{c_i}$ and $\boldsymbol{c_o}$ do not outperform $\boldsymbol{v}$, we move to the last option and shrink the simplex towards the best vertex ($\boldsymbol{u}$).

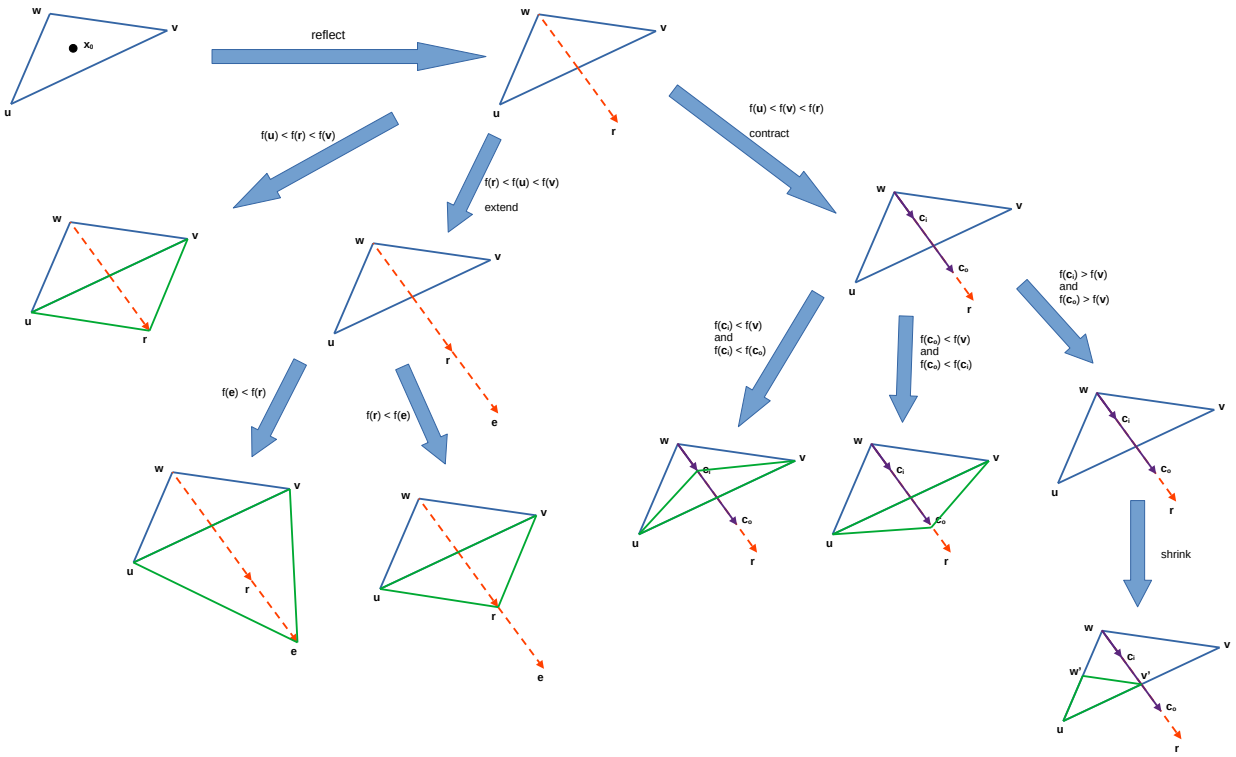


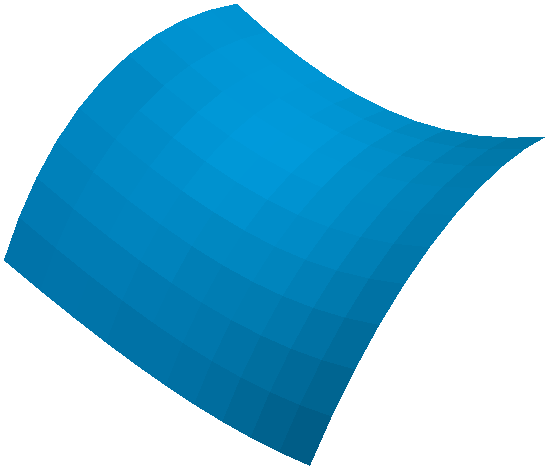
Figure 16: Diagram of the Nelder-Mead simplex method

The iterative transformation of the simplex continues until a termination condition is reached, which itself consists of two sub-conditions. Firstly, the difference between the final objective function value and *all* previous objective function values must be smaller than f_{tol}. Secondly, the difference between any design variable in the final design and the value of the same design variable in *all* previous designs must be smaller than x_{tol}. The magnitudes of f_{tol} and x_{tol} are both 10^{-5}. When these conditions are met, the algorithm stops and outputs the best discovered design. At that point, we say the algorithm has converged, which does not mean that it has found the global minimum of the objective function, since it could have found a local minimum.

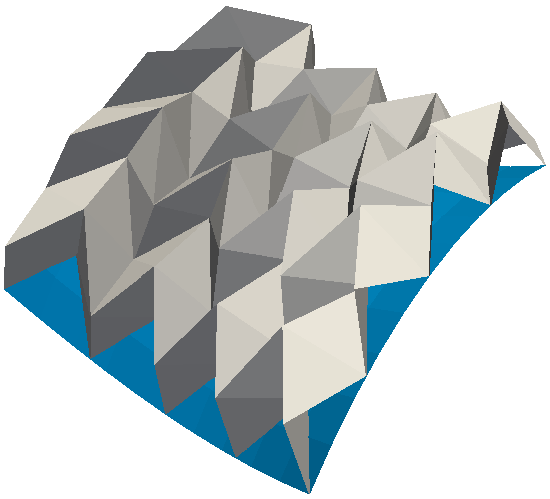
As previously discussed, the number of design variables for a 4-by-4 Miura-ori unit cell design is 84, which means the simplex will have 85 nodes, and all nodes will exist in an 84-dimensional space.

3. Results

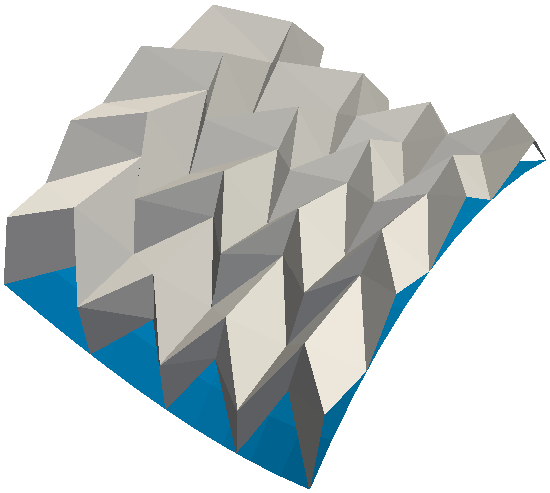
We start with the results for the first part of the method, which uses rigid origami. Figure 17 shows the rigid origami design as it is placed onto the target saddle shape, and its process of optimization towards a foldable design. Figure 17b is the same as figure 3f. Further, the course of the objective function (equation 1) and constraints (equations 2 and 4) are displayed in figure 18. Finally, figure 19 shows the flat origami pattern that can rigidly fold into the target shape. In this section, only the results for the saddle target shape are included; the results for the dome and arch target shapes are given in appendix A.9.



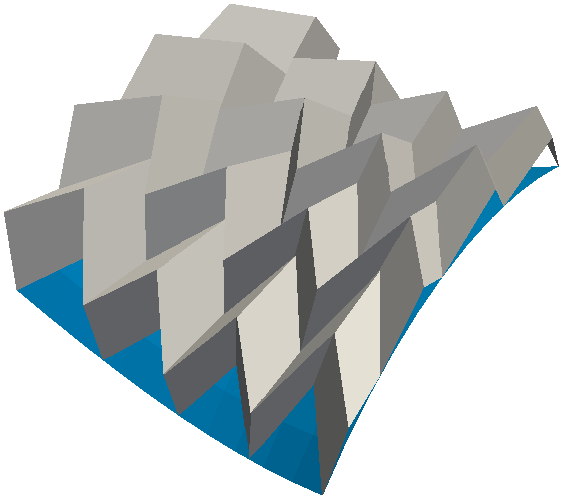
(a) Target shape



(b) Design made by placing Miura-ori-like unit cells (not foldable)

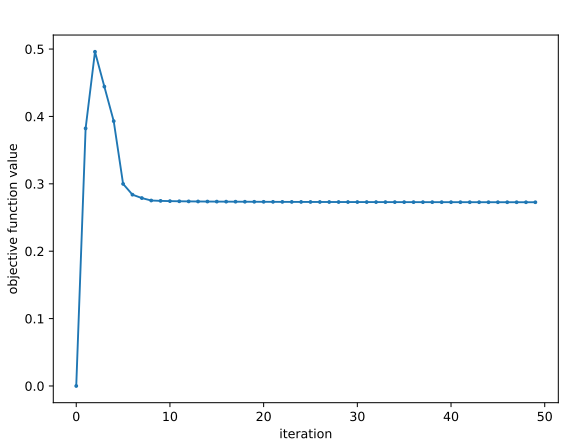


(c) Intermediate design during optimization

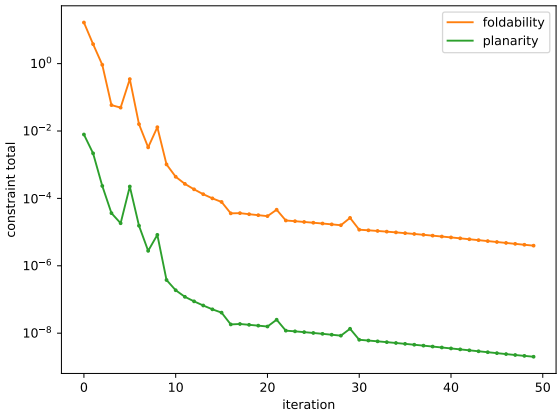


(d) Optimized design which is foldable

Figure 17: Sequence of steps for optimizing a rigid origami design such that it is foldable



(a) Objective function during optimization



(b) Sum of constraints during optimization (logarithmic vertical axis)

Figure 18

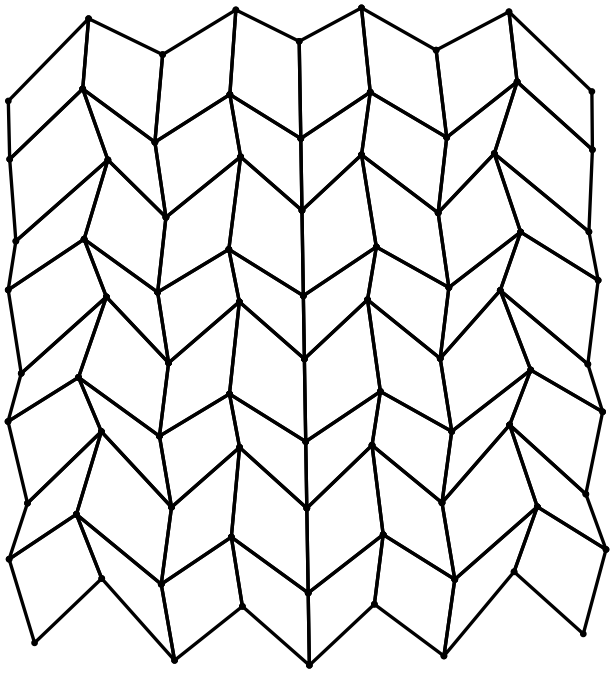


Figure 19: Flattened pattern of the optimized design

The optimization of a single rigid origami design with the "trust-constr" algorithm from SciPy's `optimize.minimize` library was terminated after 50 iterations, equivalent to around 30 seconds. The initial design has a perfect objective function value but is not foldable. As seen in figure 18, for the first few iterations, the objective function value increases as the optimizer greatly reduces the constraint violations. This Byrd-Omojokun trust-region SQP method then quickly drives the objective function down to close to its final value while continually reducing the constraint violations, to the point where the constraint violation is approximately one millionth of the original value.

We now move to the non-rigid origami shape matching results, where we use the rigid origami results as initial designs. In figure 20, we compare the performance of the optimization under different values for the parameter p. This comparison was only done for the saddle target shape case. This figure does not show the actual objective function values (which is the smoothened maximum distance; for that, see appendix A.14), but instead it shows

the function we are truly interested in, which is the real maximum distance. If we simply use the maximum distance as the objective function, the optimization reaches a maximum distance of 0.0288 m after almost seven thousand iterations. But if we instead use the smooth maximum distance as the objective function, it reaches lower values for the real maximum distance as well. In this case (the saddle case), $p = 3$ and $p = 5$ outperform the "no smoothing" strategy, while $p = 1$, $p = 2$, and $p = 4$ do not. We can see that $p = 3$ performs best, so that is the value that was used for the remaining optimizations as well.

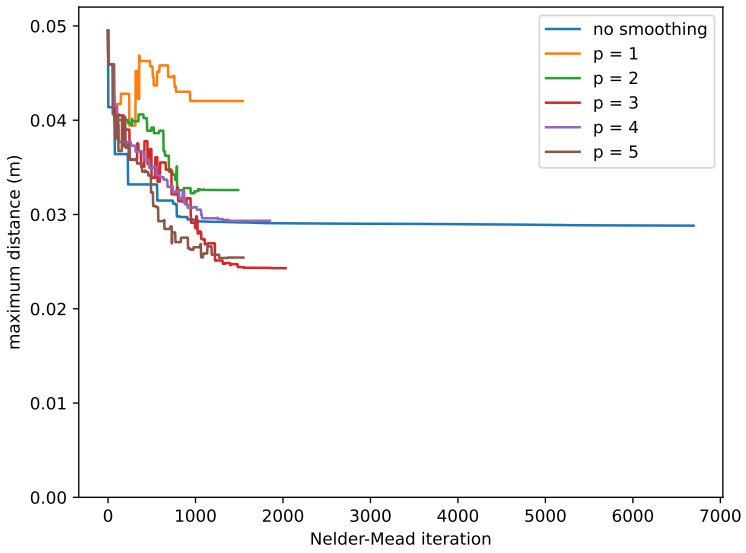
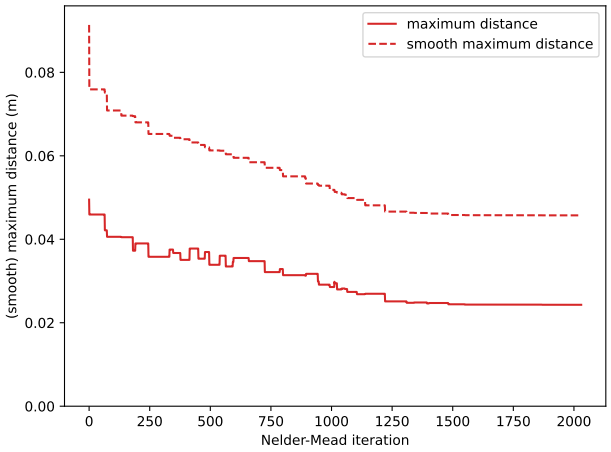
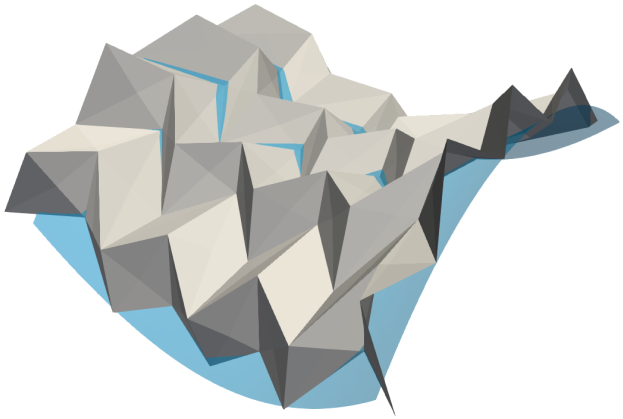


Figure 20: Maximum distance function for a non-smoothened objective function and for smoothened objective functions with different values of p

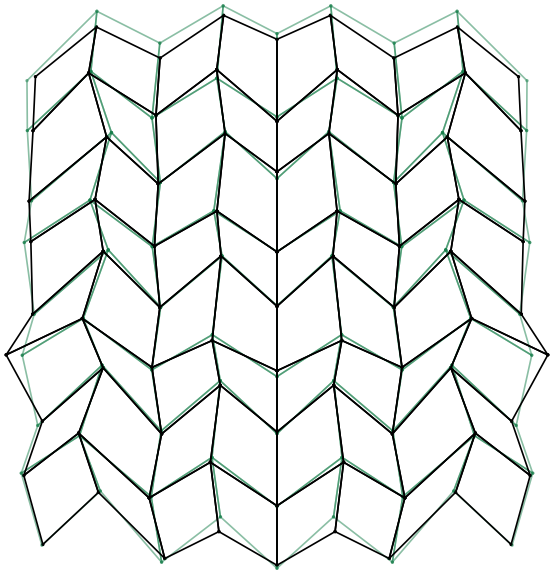
Figure 21a presents the development of the objective function and the maximum distance during the optimization cycle of the saddle origami. We are most interested in the maximum distance, which starts at 0.0495 m and finishes at 0.0243 m. The maximum distance between the folded origami and its target shape has thus more than halved. For perspective: the size of the flat origami designs is around 800 mm by 800 mm, also for the dome and arch shapes. Looking at how the flat origami design has evolved in figure 21c, we see that most nodes have stayed close to their original position, but there are also relatively large changes to the design, mostly on the outer nodes.



(a) (Smooth) maximum distance over the optimization iterations



(b) Folded saddle on the target surface

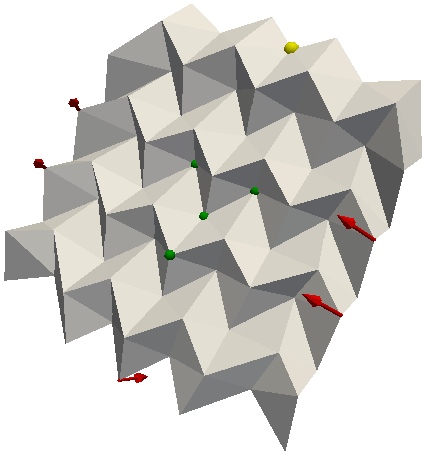


(c) Flat origami design; initial design in green, final design in black

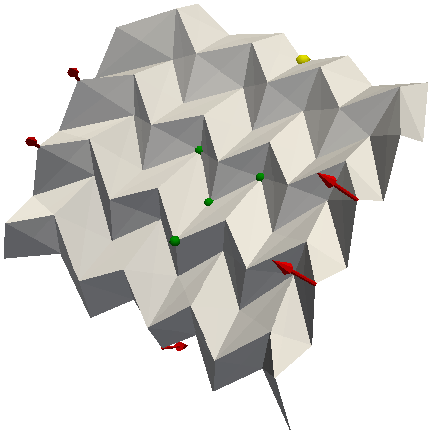
Figure 21: Non-rigid shape matching results for the saddle

Watch the animation here: https://www.youtube.com/watch?v=o-_GPq-biL0

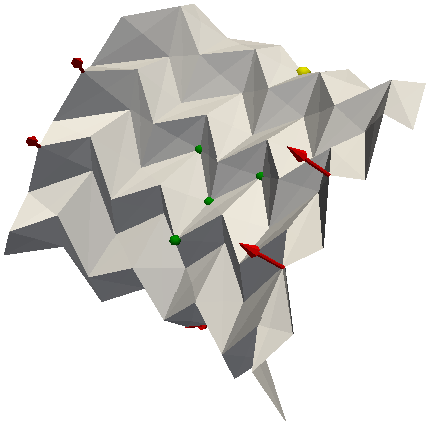
Figure 22 shows the mechanical simulation results of a Miura-ori sheet folding into a saddle shape. To the left are the physical configurations and deformed states of the origami, with the constrained nodes in green (note that not necessarily all three components are constrained), the forces in red, and the relevant DOF for measuring displacement in yellow. On the right, the figure depicts the development of the load factor over the arc-length increments, and the force-displacement curve.



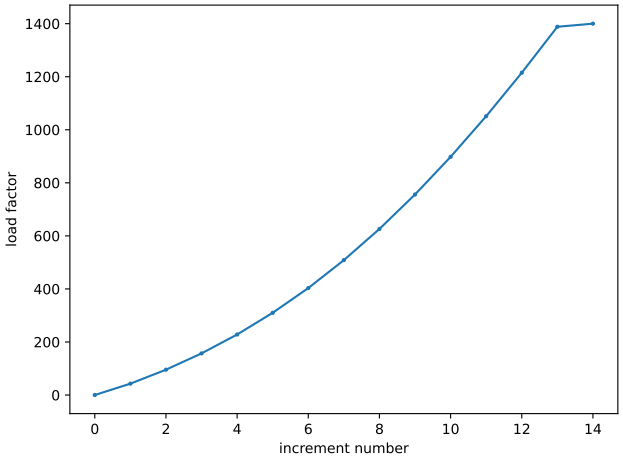
(a) Initial state of the origami



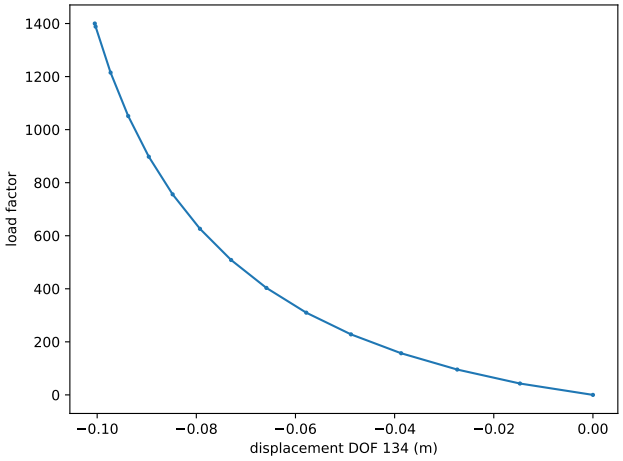
(b) Intermediate state of the origami



(c) Final state of the origami



(d) Load factor over the increments of the arc-length method

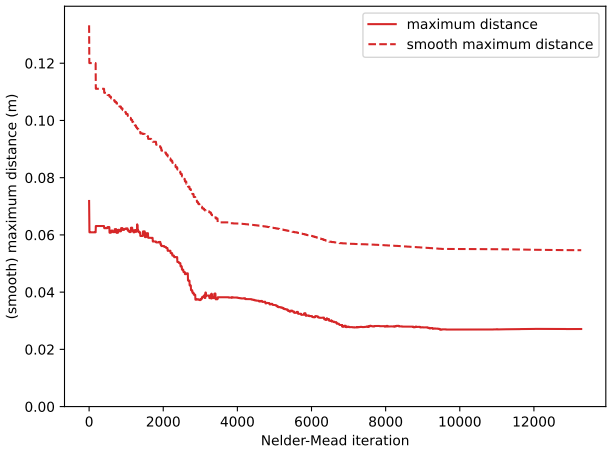


(e) Force-displacement graph

Figure 22: Mechanical simulation of a compressed Miura-ori sheet folding into a saddle shape

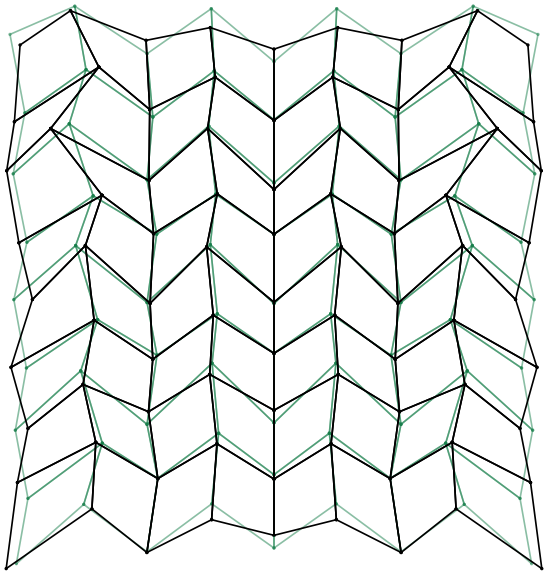
Watch the animation here: https://www.youtube.com/watch?v=6auzJVA97uI

Figure 23 presents the results for the non-rigid shape-matching towards a dome shape. The origami targeting the dome shows a reduction in the maximum distance from 0.0719 m to 0.0271 m. And the design has again changed mostly on the outer nodes of the origami, with greater differences with respect to the initial design than we saw for the saddle shape. It also took many more iterations for the optimization algorithm to converge and terminate: 13277 for the dome case as opposed to 2031 for the saddle case.



(a) (Smooth) maximum distance over the optimization iterations

(b) Folded dome on the target surface

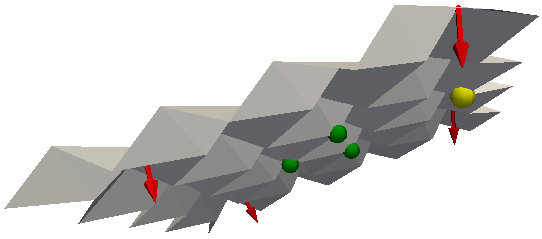


(c) Flat origami design; initial design in green, final design in black

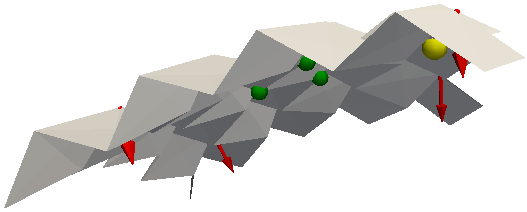
Figure 23: Non-rigid shape matching results for the dome

Watch the animation here: https://www.youtube.com/watch?v=xmVy3zccJeE

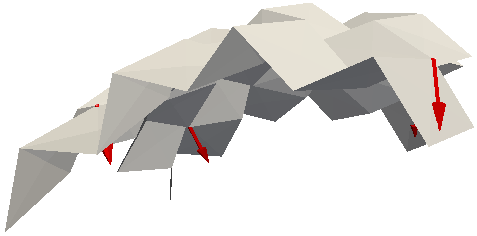
Figure 24 shows the mechanical simulation results of a Miura-ori sheet folding into a dome shape.



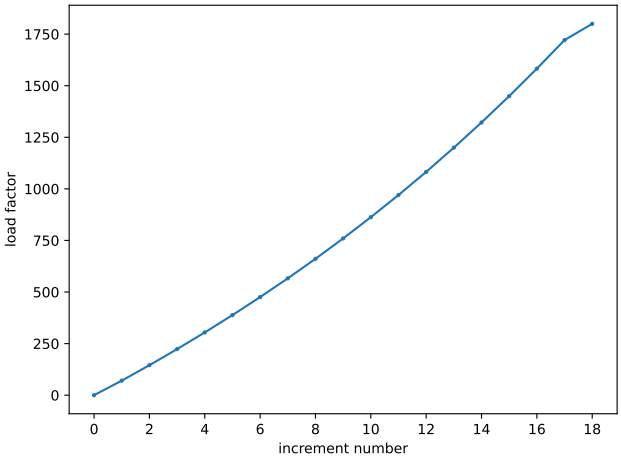
(a) Initial state of the origami



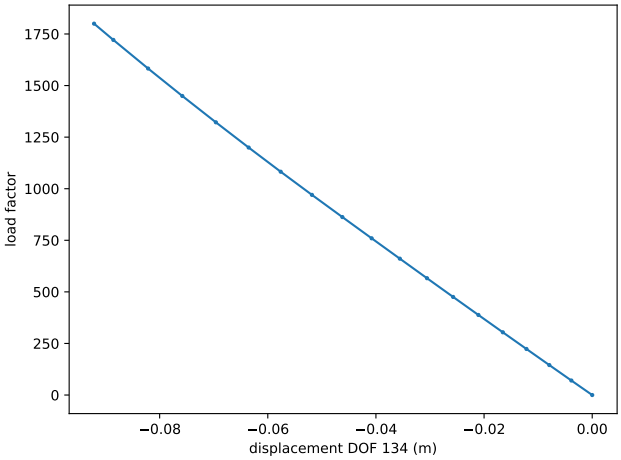
(b) Intermediate state of the origami



(c) Final state of the origami



(d) Load factor over the increments of the arc-length method

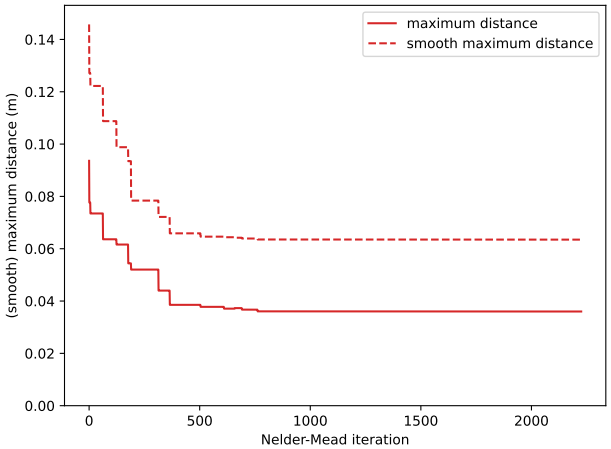


(e) Force-displacement graph

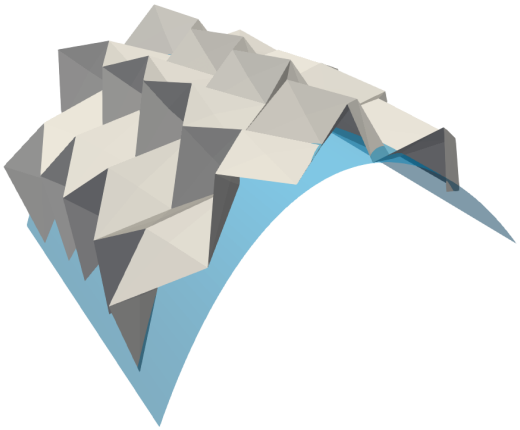
Figure 24: Results of the mechanical simulation of a compressed Miura-ori sheet folding into a dome shape

Watch the animation here: https://www.youtube.com/watch?v=dlpCePKIHqs

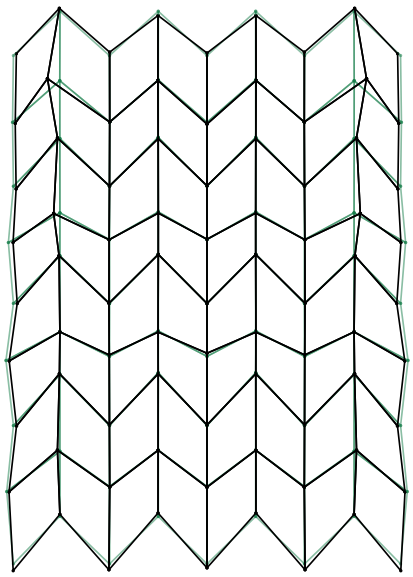
Finally, figure 25 presents the results for the non-rigid shape-matching towards an arch shape. The optimization shows similar behaviour to the saddle and dome shapes, with the maximum distance starting at 0.0936 m and finishing at 0.0360 m. The changes in this design are concentrated in just a few nodes, but the maximum distance was still reduced by a factor of 2.6 in 2226 iterations.



(a) (Smooth) maximum distance over the optimization iterations



(b) Folded arch on the target surface

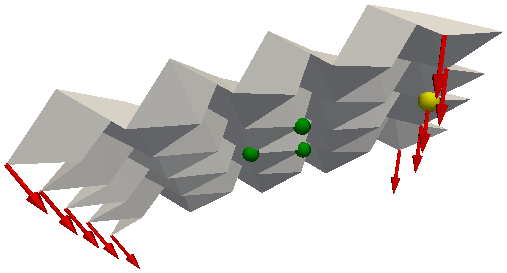


(c) Flat origami design; initial design in green, final design in black

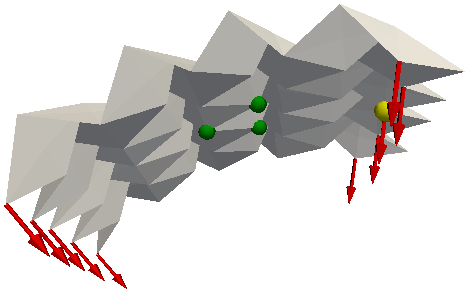
Figure 25: Non-rigid shape matching results for the arch

Watch the animation here: https://www.youtube.com/watch?v=QYko74TE6Uc

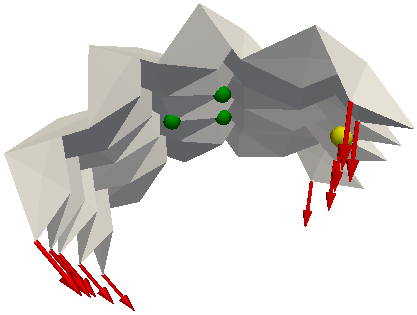
Figure 26 shows the mechanical simulation results of a Miura-ori sheet folding into an arch shape.



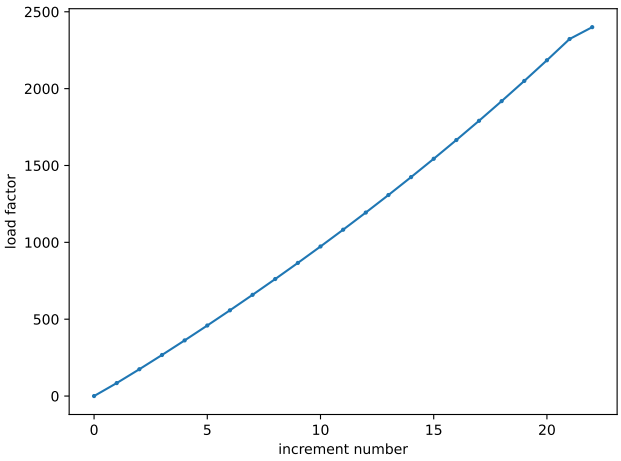
(a) Initial state of the origami



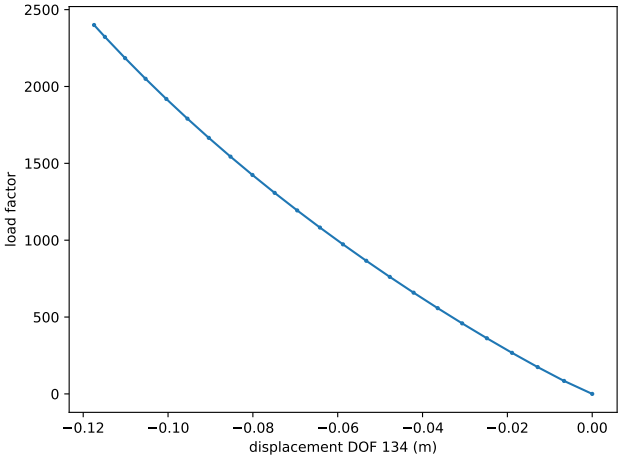
(b) Intermediate state of the origami



(c) Final state of the origami



(d) Load factor over the increments of the arc-length method



(e) Force-displacement graph

Figure 26: Results of the mechanical simulation of a compressed Miura-ori sheet folding into an arch shape

Watch the animation here: https://www.youtube.com/watch?v=rvjcen-PpNk

4. Discussion and conclusion

In this thesis we have introduced a two-step methodology for origami shape matching, where the first step uses rigid origami and creates an initial design for the second step which uses non-rigid origami.

The results for the rigid origami shape matching process are satisfactory: the origami designs can rigidly fold into a shape with its bottom touching the target shape. This process was proven to be successful for at least the saddle, dome, and arch shapes.

While attempting to produce designs with larger curvatures, a problem arose. During the optimization towards a foldable design, some of the quadrilateral facets would transform into unphysical shapes. Normally, all quadrilaterals in the Miura-ori origami are convex, but they were transformed into concave and even complex quadrilaterals during this process (figure 27). Those shapes do not fit within the Miura-ori pattern and must thus be avoided. Some mitigation techniques were tested but these could not prevent the unwanted transformations, so we limited the procedure to shapes with moderate curvature to avoid this problem.

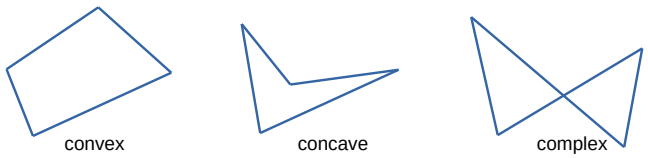


Figure 27: A convex, a concave, and a complex quadrilateral

Compared to the paper by Dudte et al. [39], this work is more limited in the types of shapes that are possible to create with the rigid origami approach. Where the scope in this work is limited to shapes that can be parametrized by a function of the form $z = f(x, y)$, Dudte et al.'s results present differently parametrized shapes such as a sphere and a curved vase. Their results also show origami designs with many more Miura-ori unit cells than those presented here, but that is also possible with the method in this work, as illustrated in appendix A.13.

Now we will discuss the bar-and-hinge model and the arc-length path-following technique. The results show that this model is successful in simulating the mechanics of a compressed Miura-ori sheet folding into a saddle, dome, or arch shape, with results for the "bump", "curl" and the unfolding or compression of a Miura-ori sheet in appendix A.10. The force-displacement graphs show smooth and nonlinear behaviour.

This work's results are very close to the results obtained from the MERLIN2 method. In fact, they are identical (to machine precision) when the originally implemented MERLIN2 arc-length method is used. As discussed in the methods section, the path-following method in this work was rewritten and improved by removing a linearization. It proved to be more robust and able to simulate the folding of Miura-ori sheets into the saddle, dome, and arch shapes. Even with this improved arc-length method, the path-following technique still has a major shortcoming: it is sensitive to the initial load factor. Depending on the initial load factor, the model will choose different equilibrium paths. And for certain initial load factors, the method will not converge at all, also for small load factors. It is thus required to experiment with this initial load factor until a desired and realistic result emerges.

One other interesting observation was that with the changes in the design during the non-rigid origami optimization cycle, the path-following procedure could act radically differently: on multiple occasions there was a design of which the first incremental point in its equilibrium path could readily be computed in a few iterations, but when this design was altered, it took multiple reductions in the initial load factor step for the equilibrium equations to converge. The cause of this behaviour is not known, other than that the implemented arc-length method is sensitive to the initial load factor. The final achieved equilibrium path of these two similar designs would nonetheless also be similar.

The non-rigid origami model was checked for frame-invariance. This check was performed by rotating and translating the problem, then calculating the displacements, and finally rotating and translating the resultant displaced nodal positions back with the inverse transformation. These positions were compared to the results of a problem that was never transformed and were found to be equal.

With this project, the accessibility and openness of the N5B8 bar-and-hinge model was improved. It is now available in the free and open-source Python programming language as part of the `crease-machine` project available on GitLab. The URL to this repository can be found in appendix A.1.

Finally, we will discuss the shape matching results of the second part of the method, which use non-rigid origami. The origami designs are subjected to loads, which causes them to deform towards their target surfaces. The magnitudes of these loads have been chosen intuitively and are not optimized; the interested reader can find the loads in the source code.

By comparing different values for the smoothing parameter p in the p-norm within the objective function, we find that $p = 3$ gives the best results out of $\{1, 2, 3, 4, 5\}$ for at least the saddle shape. It should still be investigated whether this value is optimal for other shapes too, and experiments with higher values for p would give a broader insight into the effect of the parameter. When inserting the final design, which was computed with $p = 3$, into smooth maximum distance objective functions with other values for p, we see that this design outperforms the optimized design that was directly obtained from an optimization with this other value for p. This proves that the optimization process can converge to a local minimum, which can be expected of the Nelder-Mead simplex method, because once it starts to approach a local minimum in a convex part of the design space, the simplex will reflect, contract, shrink, and ultimately converge to that local minimum.

It is interesting that the lines in figure 20 are not monotonically decreasing, because this shows that if we use smooth maximum distance functions, the optimizer sometimes changes the design such that the distance to the target shape of one or multiple nodes decreases at the cost of *increasing* the distance to the target shape of a different node (the furthest away node). That strategy leads to more possibilities to traverse the design space and apparently performs better than directly minimizing the maximum distance.

The method is expected to work for many shapes that can be defined by $z = f(x, y)$ and have limited curvature, but we will now look at the shape matching results for the three investigated shapes: the saddle, dome, and arch. The saddle and arch optimizations took around 16 hours to complete, and the dome optimization took around 40 hours because of its slow convergence, all on a laptop computer with an Intel Core i7-8750H CPU with six cores at 2.2 GHz. The most computational time is spent on the matrix operations while building the bar-and-hinge model's stiffness matrix. The changes in the origami design are mostly concentrated on the outer nodes, except for the design that approximates the arch shape, in which case the design's changes are almost exclusively concentrated in four out of the eighty-one nodes. And in all cases, the maximum distance from the origami to the target surface was less than half of what it was at the start of the optimization.

This work has successfully demonstrated a novel two-step approach for Miura-ori origami shape matching, with the resultant maximum distances to the target surfaces between 24 mm and 36 mm on origami designs that are approximately 800 mm by 800 mm in size. We have seen that by including non-rigid origami techniques, in addition to rigid origami techniques, the performances of the designs are improved. With that, Miura-ori tessellations have reached new possibilities and can be more effectively used to design various technologies, such as deployable tents, packaging, and solar panel arrays. Applications that are more sensitive to small deviations, such as deformable mirrors for laser communication, require better accuracy than the presented method can give at this moment.

Recommendations for researchers that would like to continue with this or similar work are:

- We are already taking advantage of the symmetry by using design variables for only half of the design. Another way to take advantage of symmetric structures is to simulate the mechanics of only half of the structure—you would need to set the proper boundary conditions in the bar-and-hinge model. This would almost halve the number of DOFs for large structures with many Miura-ori unit cells and greatly reduce the computational cost.
- Currently, the design variables are the locations of the nodes; it might be beneficial to instead use the deviations in the locations of the nodes with respect to their initial locations. By doing that, the magnitudes of the design variables are no longer explicitly influenced by their proximity to the origin. Nodes near the origin would no longer result in design variables that are much smaller than the design variables that come from nodes that are further away. The initial design vector would contain only zeroes. Preliminary tests with this strategy, combined with Nelder-Mead, did not improve the results, but combined with a different optimizer, it did improve the results, see appendix A.15.
- Using a gradient-based optimizer could reduce the computational resources and improve the final design. Some preliminary results, using SQP with BFGS updates, are shown in appendix A.15. Other gradient-

based optimizers are also possible and they may be more successful in avoiding local minima.

- Exploring the effects of using other materials within this framework could give new insights. This research was conducted with paper material only, but the mechanical simulation should be capable of producing accurate results for more conventional engineering materials like steel or aluminium.
- The development of this method should be governed by performance criteria that are derived from the specific application, thereby giving it direction and the potential to contribute to positive technological innovation.

5. References

[1] James Reardon-Anderson. Science and Civilisation in China. Volume 5. Chemistry and Chemical Technology. Part I. Paper and Printing. By Tsien Tsuen-Hsuin. Edited by Joseph Needham. [Cambridge: Cambridge University Press, 1985. 485 pp.]. *The China Quarterly*, 108:733–735, December 1986. Publisher: Cambridge University Press. doi:10.1017/S0305741000037309.

[2] Origami, April 2023. Page Version ID: 1148237943. URL: https://en.wikipedia.org/w/index.php?title=Origami&oldid=1148237943.

[3] Origami - prologue, Kyushu University, May 2021. URL: https://web.archive.org/web/20210507223620/https://guides.lib.kyushu-u.ac.jp/origami/prologue.

[4] History of Origami - Tokyo Origami Museum and Nippon Origami Association, November 2022. URL: https://www.origami-noa.jp.

[5] Origami - mid18C, Kyushu University, May 2021. URL: https://web.archive.org/web/20210507224948/https://guides.lib.kyushu-u.ac.jp/origami/-mid18C.

[6] K's Origami : History of Origami. URL: https://origami.ousaan.com/library/historye.html.

[7] Koshiro Hatori. *History of Origami in the East and West before Interfusion*. CRC Press, April 2016. Google-Books-ID: E7LMBQAAQBAJ.

[8] Nicholas Turner, Bill Goodwine, and Mihir Sen. A review of origami applications in mechanical engineering. *Proceedings of the Institution of Mechanical Engineers, Part C: Journal of Mechanical Engineering Science*, 230(14):2345–2362, August 2016. Publisher: IMECHE. doi:10.1177/0954406215597713.

[9] Marco Meloni, Jianguo Cai, Qian Zhang, Daniel Sang-Hoon Lee, Meng Li, Ma Ruijun, Teo Parashkevov, and Jian Feng. Engineering Origami: A Comprehensive Review of Recent Applications, Design Methods, and Tools. *Advanced Science*, 8, July 2021. doi:10.1002/advs.202000636.

[10] Yan Chen, Rui Peng, and Zhong You. Origami of thick panels. *Science*, 349(6246):396–400, July 2015. Publisher: American Association for the Advancement of Science. URL: https://www.science.org/doi/full/10.1126/science.aab2870, doi:10.1126/science.aab2870.

[11] K. C. Francis, J. E. Blanch, S. P. Magleby, and L. L. Howell. Origami-like creases in sheet materials for compliant mechanism design. *Mechanical Sciences*, 4(2):371–380, November 2013. Publisher: Copernicus GmbH. URL: https://ms.copernicus.org/articles/4/371/2013/, doi:10.5194/ms-4-371-2013.

[12] H. C. Greenberg, M. L. Gong, S. P. Magleby, and L. L. Howell. Identifying links between origami and compliant mechanisms. *Mechanical Sciences*, 2(2):217–225, December 2011. Publisher: Copernicus GmbH. URL: https://ms.copernicus.org/articles/2/217/2011/, doi:10.5194/ms-2-217-2011.

[13] Jie Liu, Haifeng Ou, Rong Zeng, Jiaxi Zhou, Kai Long, Guilin Wen, and Yi Min Xie. Fabrication, dynamic properties and multi-objective optimization of a metal origami tube with Miura sheets. *Thin-Walled Structures*, 144:106352, November 2019. URL: https://www.sciencedirect.com/science/article/pii/S0263823119306500, doi:10.1016/j.tws.2019.106352.

[14] Yasuhiro Miyazawa, Hiromi Yasuda, and Jinkyu Yang. Design of compliant mechanisms for origami metamaterials. *Acta Mechanica Sinica*, 39(7):723169, July 2023. doi:10.1007/s10409-023-23169-x.

[15] Koryo Miura. Method of Packaging and Deployment of Large Membranes in Space. *The Institute of Space and Astronautical Science report*, 618:1–9, December 1985. URL: https://jaxa.repo.nii.ac.jp/?action=pages_view_main&active_action=repository_view_main_item_detail&item_id=31382&item_no=1&page_id=13&block_id=21.

[16] Koryo Miura. The Science of Miura-Ori: A Review. In *Origami 4*. A K Peters/CRC Press, 2009. Num Pages: 14.

[17] Origami Techniques Applied to Space Development | December 2021 | Highlighting Japan. URL: https://www.gov-online.go.jp/eng/publicity/book/hlj/html/202112/202112_05_en.html.

[18] Guilherme V. Rodrigues, Larissa M. Fonseca, Marcelo A. Savi, and Alberto Paiva. Nonlinear dynamics of an adaptive origami-stent system. *International Journal of Mechanical Sciences*, 133:303–318, November 2017. URL: https://linkinghub.elsevier.com/retrieve/pii/S0020740317308871, doi:10.1016/j.ijmecsci.2017.08.050.

[19] Gaussian curvature, January 2024. Page Version ID: 1200821220. URL: https://en.wikipedia.org/w/index.php?title=Gaussian_curvature&oldid=1200821220#Total_curvature.

[20] Jerome Caron and Stefan Bäumer. Progress in freeform mirror design for space applications. In *International Conference on Space Optics — ICSO 2020*, volume 11852, pages 747–767. SPIE, June 2021. URL: https://www.spiedigitallibrary.org/conference-proceedings-of-spie/11852/118521S/Progress-in-freeform-mirror-design-for-space-applications/10.1117/12.2599322.full, doi:10.1117/12.2599322.

[21] PJ du Plooy, Gilles Ackaert, G. Witvoet, and W. Crowcombe. System Architecture for a CubeSat DTE Laser Communication System. *Small Satellite Conference*, August 2022. URL: https://digitalcommons.usu.edu/smallsat/2022/all2022/334.

[22] Luca Zimmermann, Kristina Shea, and Tino Stanković. Conditions for Rigid and Flat Foldability of Degree-n Vertices in Origami. *Journal of Mechanisms and Robotics*, 12(1), October 2019. doi:10.1115/1.4045249.

[23] Rob Burgoon, Zoë Wood, and Eitan Grinspun. Discrete Shells Origami. *21st International Conference on Computers and their Applications (CATA 2006) Proceedings*, March 2006. URL: https://digitalcommons.calpoly.edu/csse_fac/204.

[24] Cai Jianguo, Deng Xiaowei, Zhou Ya, Feng Jian, and Tu Yongming. Bistable Behavior of the Cylindrical Origami Structure With Kresling Pattern. *Journal of Mechanical Design*, 137(061406), June 2015. doi:10.1115/1.4030158.

[25] K. Liu and G. H. Paulino. Nonlinear mechanics of non-rigid origami: an efficient computational approach. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 473(2206):20170348, October 2017. Publisher: Royal Society. URL: https://royalsocietypublishing.org/doi/full/10.1098/rspa.2017.0348, doi:10.1098/rspa.2017.0348.

[26] Mark Schenk and Simon D. Guest. Origami Folding: A Structural Engineering Approach. In *Origami 5*. A K Peters/CRC Press, 2011. Num Pages: 14.

[27] 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*, 124:26–45, October 2017. URL: https://www.sciencedirect.com/science/article/pii/S0020768317302408, doi:10.1016/j.ijsolstr.2017.05.028.

[28] Ke Liu and Glaucio H Paulino. MERLIN: A MATLAB implementation to capture highly nonlinear behavior of non-rigid origami. *Proceedings of IASS Annual Symposia*, 2016(13):1–10, September 2016.

[29] E.T. Filipov, T. Tachi, and G.H. Paulino. Toward optimization of stiffness and flexibility of rigid, flat-foldable origami structures. *In Origami 6, Proc. 6th Int. Meeting on Origami Science, Mathematics, and Education*, page 409419, 2016.

[30] Ke Liu and G. H. Paulino. Highly efficient nonlinear structural assemblages using the MERLIN2 software. 2018.

[31] Robert J. Lang. A computational algorithm for origami design. In *Proceedings of the twelfth annual symposium on Computational geometry - SCG '96*, pages 98–105, Philadelphia, Pennsylvania, United States, 1996. ACM Press. URL: http://portal.acm.org/citation.cfm?doid=237218.237249, doi:10.1145/237218.237249.

[32] Thomas A. Evans, Robert J. Lang, Spencer P. Magleby, and Larry L. Howell. Rigidly foldable origami gadgets and tessellations. *Royal Society Open Science*, 2(9):150067, 2015. Publisher: Royal Society. URL: https://royalsocietypublishing.org/doi/full/10.1098/rsos.150067, doi:10.1098/rsos.150067.

[33] Erik Demaine and Tomohiro Tachi. Origamizer: A practical algorithm for folding any polyhedron. *DROPS*, 2017. Accepted: 2021-12-20T19:22:07Z. URL: https://dspace.mit.edu/handle/1721.1/137685.2.

[34] Erik D. Demaine, Martin L. Demaine, and Joseph S. B. Mitchell. Folding flat silhouettes and wrapping polyhedral packages: new results in computational origami. In *Proceedings of the fifteenth annual symposium on Computational geometry*, pages 105–114, Miami Beach Florida USA, June 1999. ACM. URL: https://dl.acm.org/doi/10.1145/304893.304933, doi:10.1145/304893.304933.

[35] Tomohiro Tachi. Origamizing Polyhedral Surfaces. *IEEE transactions on visualization and computer graphics*, 16:298–311, March 2010. doi:10.1109/TVCG.2009.67.

[36] Tomohiro Tachi. Origamizer, 2008. URL: https://origami.c.u-tokyo.ac.jp/~tachi/software/.

[37] Luca Zimmermann, Kristina Shea, and Tino Stanković. A Computational Design Synthesis Method for the Generation of Rigid Origami Crease Patterns. *Journal of Mechanisms and Robotics*, 14(3), December 2021. doi:10.1115/1.4052847.

[38] Andreas Walker and Tino Stankovic. Algorithmic design of origami mechanisms and tessellations. *Communications Materials*, 3(1):1–8, January 2022. Number: 1 Publisher: Nature Publishing Group. URL: https://www.nature.com/articles/s43246-022-00227-5, doi:10.1038/s43246-022-00227-5.

[39] Levi H. Dudte, Etienne Vouga, Tomohiro Tachi, and L. Mahadevan. Programming curvature using origami tessellations. *Nature Materials*, 15(5):583–588, May 2016. Number: 5 Publisher: Nature Publishing Group. URL: https://www.nature.com/articles/nmat4540, doi:10.1038/nmat4540.

[40] Kazuko Fuchi, Philip R. Buskohl, Giorgio Bazzan, Michael F. Durstock, Gregory W. Reich, Richard A. Vaia, and James J. Joo. Origami Actuator Design and Networking Through Crease Topology Optimization. *Journal of Mechanical Design*, 137(9), July 2015. doi:10.1115/1.4030876.

[41] Zhen Li, Vipin Agarwal, Liangmo Wang, and K. W. Wang. On-demand tuning of mechanical stiffness and stability of Kresling origami harnessing its nonrigid folding characteristics. *Smart Materials and Structures*, 32(8):085025, July 2023. Publisher: IOP Publishing. URL: https://dx.doi.org/10.1088/1361-665X/ace0eb, doi:10.1088/1361-665X/ace0eb.

[42] Siyuan Ye, Pengyuan Zhao, Yinjun Zhao, Fatemeh Kavousi, Huijuan Feng, and Guangbo Hao. A Novel Radially Closable Tubular Origami Structure (RC-ori) for Valves. *Actuators*, 11(9):243, September 2022. Number: 9 Publisher: Multidisciplinary Digital Publishing Institute. URL: https://www.mdpi.com/2076-0825/11/9/243, doi:10.3390/act11090243.

[43] scipy.optimize.minimize — SciPy v1.12.0 Manual. URL: https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.minimize.html.

[44] Marucha Lalee, Jorge Nocedal, and Todd Plantenga. On the Implementation of an Algorithm for Large-Scale Equality Constrained Optimization. *SIAM Journal on Optimization*, 8(3):682–706, August 1998. Publisher: Society for Industrial and Applied Mathematics. URL: https://epubs.siam.org/doi/abs/10.1137/S1052623493262993, doi:10.1137/S1052623493262993.

[45] Alan R. Jones. An experimental investigation of the in-plane elastic moduli of paper, June 1967.

[46] M. A. Crisfield. A fast incremental/iterative solution procedure that handles "snap-through". *Computers & Structures*, 13(1):55–62, June 1981. URL: https://www.sciencedirect.com/science/article/pii/0045794981901085, doi:10.1016/0045-7949(81)90108-5.

[47] R. de Borst and L.J. Sluys. *Computational methods in non-linear solid mechanics*. Delft, April 2015.

[48] E. Ramm. Strategies for Tracing the Nonlinear Response Near Limit Points. In W. Wunderlich, E. Stein, and K.-J. Bathe, editors, *Nonlinear Finite Element Analysis in Structural Mechanics*, pages 63–89, Berlin, Heidelberg, 1981. Springer. doi:10.1007/978-3-642-81589-8_5.

[49] Norm (mathematics), September 2024. Page Version ID: 1244275537. URL: https://en.wikipedia.org/w/index.php?title=Norm_(mathematics)&oldid=1244275537#p-norm.

[50] Heike Vallery and Arend Schwab. *Advanced Dynamics*. Stichting Newton-Euler, 2021. URL: https://filelist.tudelft.nl/3mE/Organisatie/Afdelingen/BmechE/DBL/Publications/doc/AdvancedDynamicsVallerySchwab_3ed_ExtendedPreview.pdf.

[51] J. A. Nelder and R. Mead. A Simplex Method for Function Minimization. *The Computer Journal*, 7(4):308–313, January 1965. doi:10.1093/comjnl/7.4.308.

[52] A. R. Abdulghany. Generalization of parallel axis theorem for rotational inertia. *American Journal of Physics*, 85(10):791–795, October 2017. doi:10.1119/1.4994835.

A. Appendix

A.1 URL to the project's source code

https://gitlab.com/niels_b/crease-machine

MERLIN2 is programmed in Matlab, a non-free and closed-source programming language. Contrarily, the Python programming language is well aligned with the open philosophy of the scientific community, as it is free to inspect, use, and modify—it is open-source. This means that anyone can run Python programs and see what is happening "behind the scenes". That is why I chose to translate the powerful MERLIN2 software into Python.

A.2 Miura-ori-like unit cell merging procedure

Here, the merging of the separate Miura-ori-like unit cells that are placed on the target surface is explained in more detail. We loop over all of the design's nodes, determine where they fit in the pattern and derive their coordinates from the existing Miura-ori-like unit cells. Figure 28 shows from which nodes of which Miura-ori-like unit cells the nodes will inherit their positions. Figure 29 shows how the terminology of i^{th} node, strip, column, row, etc. are defined.

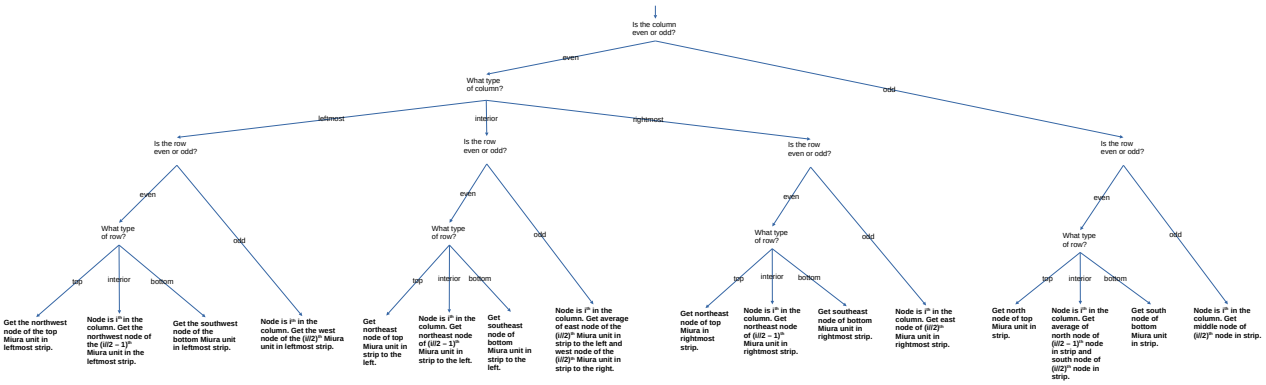


Figure 28: Detailed view of how a node's coordinates are derived

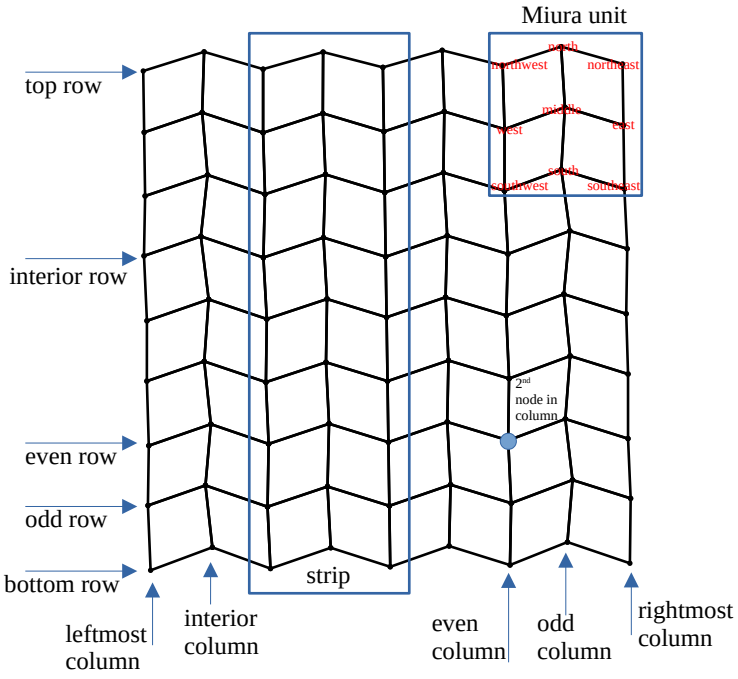


Figure 29: Explanation of terminology for merging

A.3 Sensitivities in the rigid part

The formula for the objective function was $f = \sum_{i=1}^{N_{\text{edges}}} \frac{1}{2L_0^i} \left(L^i - L_0^i\right)^2$. Its sensitivity takes the form: $\begin{bmatrix} \frac{\partial f}{\partial d_0} \\ \frac{\partial f}{\partial d_1} \\ . \\ . \\ \frac{\partial f}{\partial d_{n-1}} \end{bmatrix}$, where d_0 is the zeroth design variable. The number of design variables here is n. The derivative of f to d_k is $\frac{\partial f}{\partial d_k} = \frac{\partial \sum_{i=1}^{N_{\text{edges}}} \frac{1}{2L_0^i} \left(L^i - L_0^i\right)^2}{\partial d_k}$. Only edges that actually connect to the node with d_k have a contribution to the derivative; we need to filter for that during the calculation. Looking at a single edge: $\frac{\partial f_i}{\partial d_k} = \frac{\partial \frac{1}{2L_0^i} \left(L^i - L_0^i\right)^2}{\partial d_k} = \frac{1}{2L_0^i} * 2\left(L^i - L_0^i\right) * \frac{\partial L^i}{\partial d_k} = \left(\frac{L^i}{L_0^i} - 1\right) * \frac{\partial L^i}{\partial d_k}$ with $\frac{\partial L^i}{\partial d_k} = \frac{1}{2\sqrt{(x_L - x_R)^2 + (y_L - y_R)^2 + (z_L - z_R)^2}} * 2\left(x_L - x_R\right) * 1 = \frac{x_L - x_R}{L}$ in the case that d_k corresponds to an x-coordinate. If it corresponds to a y- or z-coordinate, the symbol in the previous numerator must also be y or z. If d_k is a coordinate of the node of the edge that is defined as 'right', there is an extra factor of -1 in the equation. All contributions of connected edges must be summed to give the value for $\frac{\partial f}{\partial d_k}$.

The sensitivities of the constraints are more involved. They take the shape $jac = \begin{bmatrix} \frac{\partial g_0}{\partial d_0} & \frac{\partial g_0}{\partial d_1} & \dots & \frac{\partial g_0}{\partial d_{n-1}} \\ \frac{\partial g_1}{\partial d_0} & \frac{\partial g_1}{\partial d_1} & \dots & \dots \\ \dots & \dots & \dots & \dots \\ \frac{\partial g_{m-1}}{\partial d_0} & \dots & \dots & \frac{\partial g_{m-1}}{\partial d_{n-1}} \end{bmatrix}$ where m is the number of constraints and n is the number of design variables. There are two types of constraints: the foldability and planarity constraints.

First, we'll treat the foldability constraints, of which a single one could be expressed as $g_{fold,i} = 2\pi - \sum_{j=0}^{3} \angle\left(\boldsymbol{p_{n_j}} - \boldsymbol{p_i}, \boldsymbol{p_{n_{j+1}}} - \boldsymbol{p_i}\right) = 0$ with $\angle(\boldsymbol{v}, \boldsymbol{w}) = 2\ \text{atan2}\left(\|\boldsymbol{v} \times \boldsymbol{w}\|, \|\boldsymbol{v}\| \|\boldsymbol{w}\| + \boldsymbol{v} \cdot \boldsymbol{w}\right)$. When calculating the derivative of constraint g_i to design variable d_k, we must first check whether d_k is in the equation for that specific g_i. If it's not, we can put a 0 in the position (i, k) in the sensitivity matrix. But if d_k is in the equation for g_i, we must calculate the sensitivity. $\text{atan2}(x, y)$ is actually the same as $\arctan\left(\frac{x}{y}\right)$ up to a constant for

non-zero x and y. The derivative of $\text{atan2}(x, y)$ is therefore the same as the derivative of $\arctan\left(\frac{x}{y}\right)$ for non-zero inputs—this derivative is $\frac{1}{1+\left(\frac{x}{y}\right)^2}$. Filling in the values for x and y and applying the chain rule, we get

$$\frac{\partial \text{atan2}\left(\|\boldsymbol{v}\times\boldsymbol{w}\|, \|\boldsymbol{v}\|\|\boldsymbol{w}\| + \boldsymbol{v}\cdot\boldsymbol{w}\right)}{\partial d_k} = \frac{1}{1+\left(\frac{\|\boldsymbol{v}\times\boldsymbol{w}\|}{\|\boldsymbol{v}\|\|\boldsymbol{w}\|+\boldsymbol{v}\cdot\boldsymbol{w}}\right)^2} * \frac{\left(\|\boldsymbol{v}\|\|\boldsymbol{w}\| + \boldsymbol{v}\cdot\boldsymbol{w}\right)\frac{\partial\|\boldsymbol{v}\times\boldsymbol{w}\|}{\partial d_k} - \|\boldsymbol{v}\times\boldsymbol{w}\|\frac{\partial(\|\boldsymbol{v}\|\|\boldsymbol{w}\|+\boldsymbol{v}\cdot\boldsymbol{w})}{\partial d_k}}{\left(\|\boldsymbol{v}\|\|\boldsymbol{w}\| + \boldsymbol{v}\cdot\boldsymbol{w}\right)^2} \tag{21}$$

Two terms in this equation still need to be expressed: $\frac{\partial\|\boldsymbol{v}\times\boldsymbol{w}\|}{\partial d_k}$ and $\frac{\partial(\|\boldsymbol{v}\|\|\boldsymbol{w}\|+\boldsymbol{v}\cdot\boldsymbol{w})}{\partial d_k}$, they are shown in equations 22 and 23, respectively. The expressions will be different, depending on whether d_k is in $\boldsymbol{p_i}$, $\boldsymbol{p_{n_j}}$, or $\boldsymbol{p_{n_{j+1}}}$ and whether it is an x-, y-, or z-coordinate. That is why there are multiple possibilities given in the equations, the remainder of equations can be inferred from the stated equations and can be automatically generated in computer code. The variable x_i is the x-coordinate of node $\boldsymbol{p_i}$, $z_{n_{j+1}}$ is the z-coordinate of $\boldsymbol{p_{n_{j+1}}}$, etc.

$$\begin{aligned}
\frac{\partial\|\boldsymbol{v}\times\boldsymbol{w}\|}{\partial x_i} =& \frac{((z_{n_j}-z_i)(x_{n_{j+1}}-x_i)-(z_{n_{j+1}}-z_i)(x_{n_j}-x_i))*(-(z_{n_j}-z_i)+(z_{n_{j+1}}-z_i))}{\|\boldsymbol{v}\times\boldsymbol{w}\|} \\
&+ \frac{((x_{n_j}-x_i)(y_{n_{j+1}}-y_i)-(x_{n_{j+1}}-x_i)(y_{n_j}-y_i))*(-(y_{n_{j+1}}-y_i)+(y_{n_j}-y_i))}{\|\boldsymbol{v}\times\boldsymbol{w}\|} \\
\frac{\partial\|\boldsymbol{v}\times\boldsymbol{w}\|}{\partial y_i} =& \frac{((y_{n_j}-y_i)(z_{n_{j+1}}-z_i)-(y_{n_{j+1}}-y_i)(z_{n_j}-z_i))*(-(z_{n_{j+1}}-z_i)+(z_{n_j}-z_i))}{\|\boldsymbol{v}\times\boldsymbol{w}\|} \\
&+ \frac{((x_{n_j}-x_i)(y_{n_{j+1}}-y_i)-(x_{n_{j+1}}-x_i)(y_{n_j}-y_i))*(-(x_{n_j}-x_i)+(x_{n_{j+1}}-x_i))}{\|\boldsymbol{v}\times\boldsymbol{w}\|} \\
\frac{\partial\|\boldsymbol{v}\times\boldsymbol{w}\|}{\partial x_{n_j}} =& \frac{((z_{n_j}-z_i)(x_{n_{j+1}}-x_i)-(z_{n_{j+1}}-z_i)(x_{n_j}-x_i))*-(z_{n_{j+1}}-z_i)}{\|\boldsymbol{v}\times\boldsymbol{w}\|} \\
&+ \frac{((x_{n_j}-x_i)(y_{n_{j+1}}-y_i)-(x_{n_{j+1}}-x_i)(y_{n_j}-y_i))*(y_{n_{j+1}}-y_i)}{\|\boldsymbol{v}\times\boldsymbol{w}\|} \\
\frac{\partial\|\boldsymbol{v}\times\boldsymbol{w}\|}{\partial y_{n_j}} =& \frac{((y_{n_j}-y_i)(z_{n_{j+1}}-z_i)-(y_{n_{j+1}}-y_i)(z_{n_j}-z_i))*(z_{n_{j+1}}-z_i)}{\|\boldsymbol{v}\times\boldsymbol{w}\|} \\
&+ \frac{((x_{n_j}-x_i)(y_{n_{j+1}}-y_i)-(x_{n_{j+1}}-x_i)(y_{n_j}-y_i))*-(x_{n_{j+1}}-x_i)}{\|\boldsymbol{v}\times\boldsymbol{w}\|} \\
\frac{\partial\|\boldsymbol{v}\times\boldsymbol{w}\|}{\partial x_{n_{j+1}}} =& \frac{((z_{n_j}-z_i)(x_{n_{j+1}}-x_i)-(z_{n_{j+1}}-z_i)(x_{n_j}-x_i))*(z_{n_j}-z_i)}{\|\boldsymbol{v}\times\boldsymbol{w}\|} \\
&+ \frac{((x_{n_j}-x_i)(y_{n_{j+1}}-y_i)-(x_{n_{j+1}}-x_i)(y_{n_j}-y_i))*(y_{n_j}-y_i)}{\|\boldsymbol{v}\times\boldsymbol{w}\|}
\end{aligned} \tag{22}$$

$$\begin{aligned}
\frac{\partial\|\boldsymbol{v}\|\|\boldsymbol{w}\|+\boldsymbol{v}\cdot\boldsymbol{w}}{\partial x_i} &= -\left(x_{n_j}-x_i\right)\left(\frac{\|\boldsymbol{w}\|}{\|\boldsymbol{v}\|}+1\right) - \left(x_{n_{j+1}}-x_i\right)\left(\frac{\|\boldsymbol{v}\|}{\|\boldsymbol{w}\|}+1\right) \\
\frac{\partial\|\boldsymbol{v}\|\|\boldsymbol{w}\|+\boldsymbol{v}\cdot\boldsymbol{w}}{\partial x_{n_j}} &= \left(x_{n_j}-x_i\right)\frac{\|\boldsymbol{w}\|}{\|\boldsymbol{v}\|} + \left(x_{n_{j+1}}-x_i\right) \\
\frac{\partial\|\boldsymbol{v}\|\|\boldsymbol{w}\|+\boldsymbol{v}\cdot\boldsymbol{w}}{\partial x_{n_{j+1}}} &= \left(x_{n_{j+1}}-x_i\right)\frac{\|\boldsymbol{v}\|}{\|\boldsymbol{w}\|} + \left(x_{n_j}-x_i\right)
\end{aligned} \tag{23}$$

The individual derivatives $\frac{\partial\text{atan2}(\|\boldsymbol{v}\times\boldsymbol{w}\|,\|\boldsymbol{v}\|\|\boldsymbol{w}\|+\boldsymbol{v}\cdot\boldsymbol{w})}{\partial d_k}$ need to be multiplied with 2 and summed over $j = 0, 1, 2, 3$ according to the formula for the foldability constraint to get the correct value for the sensitivity.

Now for the planarity constraints, we have seen that $g_{planar,i} = ((\boldsymbol{p_b}-\boldsymbol{p_a})\times(\boldsymbol{p_c}-\boldsymbol{p_a}))\cdot(\boldsymbol{p_d}-\boldsymbol{p_a}) = 0$. Again, for every design variable d_k we must first check if it plays a part in the specific constraint, i.e. is it one of the corner nodes of the panel? Then we must take the derivative of $g_{planar,i}$, which can be rewritten to equation

24.

$$
\begin{aligned}
g_{planar,i} = \quad & x_d y_b z_c - x_d y_b z_a - x_d y_a z_c - x_a y_b z_c - x_d y_c z_b \\
& + x_d y_c z_a + x_d y_a z_b + x_a y_c z_b + x_c y_d z_b - x_c y_d z_a \\
& - x_c y_a z_b - x_a y_d z_b - x_b y_d z_c + x_b y_d z_a + x_b y_a z_c \\
& + x_a y_d z_c + x_b y_c z_d - x_b y_c z_a - x_b y_a z_d - x_a y_c z_d \\
& - x_c y_b z_d + x_c y_b z_a + x_c y_a z_d + x_a y_b z_d
\end{aligned} \tag{24}
$$

In this form, it is easy to take the derivative of the constraint. An example of a sensitivity for the case that $d_k = y_c$ is

$$
\frac{\partial g_{planar,i}}{\partial y_c} = -x_d z_b + x_d z_a + x_a z_b + x_b z_d - x_b z_a - x_a z_d \tag{25}
$$

Other cases are easy to derive and can be automatically generated with computer code.

The sensitivities for the objective function, foldability constraints, and planarity constraints were all verified with finite difference calculations.

A.4 Entries in the Jacobian and Hessian of the dihedral angle in a bend or fold

During the calculation of the hinge stiffness matrix in the bar-and-hinge model, we require the Jacobian and Hessian of the dihedral angle θ with respect to coordinates of nodes i, j, k, and l, see figure 30. Here we will explicitly express the entries in the Jacobian and Hessian, all from Liu and Paulino's work [25]. First we define the vector $\boldsymbol{r_{ab}}$ to be the vector from node b to node a, thus $\boldsymbol{r_{ab}} = \boldsymbol{x_a} - \boldsymbol{x_b}$, where a and b are placeholders for the nodes i, j, k, and l. We also define two normal vectors to the facets, not necessarily of unit length: $\boldsymbol{m} = \boldsymbol{r_{ij}} \times \boldsymbol{r_{kj}}$ and $\boldsymbol{n} = \boldsymbol{r_{kj}} \times \boldsymbol{r_{kl}}$.

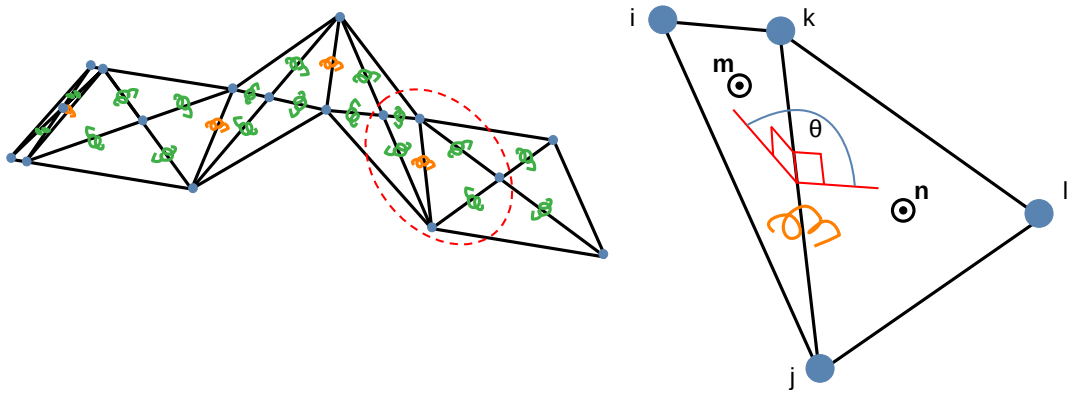


Figure 30: A hinge and the relevant nodes and vectors for calculating stiffness

The entries in the Jacobian $\frac{d\theta}{d\boldsymbol{x}^h} = \begin{bmatrix} \frac{\partial\theta}{\partial \boldsymbol{x_i}} \\ \frac{\partial\theta}{\partial \boldsymbol{x_j}} \\ \frac{\partial\theta}{\partial \boldsymbol{x_k}} \\ \frac{\partial\theta}{\partial \boldsymbol{x_l j}} \end{bmatrix}$ are expressed as.

$$
\begin{aligned}
\frac{\partial\theta}{\partial \boldsymbol{x_i}} &= \frac{\|\boldsymbol{r_{kj}}\|}{\|\boldsymbol{m}\|^2}\boldsymbol{m} \\
\frac{\partial\theta}{\partial \boldsymbol{x_l}} &= \frac{\|\boldsymbol{r_{kj}}\|}{\|\boldsymbol{n}\|^2}\boldsymbol{n} \\
\frac{\partial\theta}{\partial \boldsymbol{x_j}} &= \left(\frac{\boldsymbol{r_{ij}}\cdot\boldsymbol{r_{kj}}}{\|\boldsymbol{r_{kj}}\|^2}-1\right)\frac{\partial\theta}{\partial \boldsymbol{x_i}} - \frac{\boldsymbol{r_{kl}}\cdot\boldsymbol{r_{kj}}}{\|\boldsymbol{r_{kj}}\|^2}\frac{\partial\theta}{\partial \boldsymbol{x_l}} \\
\frac{\partial\theta}{\partial \boldsymbol{x_k}} &= \left(\frac{\boldsymbol{r_{kl}}\cdot\boldsymbol{r_{kj}}}{\|\boldsymbol{r_{kj}}\|^2}-1\right)\frac{\partial\theta}{\partial \boldsymbol{x_l}} - \frac{\boldsymbol{r_{ij}}\cdot\boldsymbol{r_{kj}}}{\|\boldsymbol{r_{kj}}\|^2}\frac{\partial\theta}{\partial \boldsymbol{x_i}}
\end{aligned}
\tag{26}
$$

There are ten independent 3×3 submatrices in the symmetric Hessian $\frac{d^2\theta}{(d\boldsymbol{x^h})^2} = \begin{bmatrix} \frac{\partial^2\theta}{\partial \boldsymbol{x_i}^2} & \frac{\partial^2\theta}{\partial \boldsymbol{x_i}\partial \boldsymbol{x_j}} & \frac{\partial^2\theta}{\partial \boldsymbol{x_i}\partial \boldsymbol{x_k}} & \frac{\partial^2\theta}{\partial \boldsymbol{x_i}\partial \boldsymbol{x_l}} \\ \frac{\partial^2\theta}{\partial \boldsymbol{x_j}\partial \boldsymbol{x_i}} & \frac{\partial^2\theta}{\partial \boldsymbol{x_j}^2} & \frac{\partial^2\theta}{\partial \boldsymbol{x_j}\partial \boldsymbol{x_k}} & \frac{\partial^2\theta}{\partial \boldsymbol{x_j}\partial \boldsymbol{x_l}} \\ \frac{\partial^2\theta}{\partial \boldsymbol{x_k}\partial \boldsymbol{x_i}} & \frac{\partial^2\theta}{\partial \boldsymbol{x_k}\boldsymbol{x_j}} & \frac{\partial^2\theta}{\partial \boldsymbol{x_k}^2} & \frac{\partial^2\theta}{\partial \boldsymbol{x_k}\partial \boldsymbol{x_l}} \\ \frac{\partial^2\theta}{\partial \boldsymbol{x_l}\partial \boldsymbol{x_i}} & \frac{\partial^2\theta}{\partial \boldsymbol{x_l}\boldsymbol{x_j}} & \frac{\partial^2\theta}{\partial \boldsymbol{x_l}\boldsymbol{x_k}} & \frac{\partial^2\theta}{\partial \boldsymbol{x_l}^2} \end{bmatrix}$.

To be somewhat concise, we define A, B, and their derivatives:

$$
\begin{aligned}
A &= \frac{\boldsymbol{r_{ij}}\cdot\boldsymbol{r_{kj}}}{\|\boldsymbol{r_{kj}}\|^2} \\
B &= \frac{\boldsymbol{r_{kl}}\cdot\boldsymbol{r_{kj}}}{\|\boldsymbol{r_{kj}}\|^2} \\
\frac{\partial A}{\partial \boldsymbol{x_j}} &= \frac{1}{\|\boldsymbol{r_{kj}}\|^2}((2A-1)\boldsymbol{r_{kj}} - \boldsymbol{r_{ij}}) \\
\frac{\partial B}{\partial \boldsymbol{x_j}} &= \frac{1}{\|\boldsymbol{r_{kj}}\|^2}(2B\boldsymbol{r_{kj}} - \boldsymbol{r_{kl}}) \\
\frac{\partial A}{\partial \boldsymbol{x_k}} &= \frac{1}{\|\boldsymbol{r_{kj}}\|^2}(-2A\boldsymbol{r_{kj}} + \boldsymbol{r_{ij}}) \\
\frac{\partial B}{\partial \boldsymbol{x_k}} &= \frac{1}{\|\boldsymbol{r_{kj}}\|^2}((1-2B)\boldsymbol{r_{kj}} + \boldsymbol{r_{kl}})
\end{aligned}
\tag{27}
$$

And just as Liu and Paulino, we define the operator $\diamond$

$$
\boldsymbol{a}\diamond\boldsymbol{b} := \boldsymbol{a}\otimes\boldsymbol{b} + \boldsymbol{b}\otimes\boldsymbol{a} \qquad \forall\ \boldsymbol{a},\boldsymbol{b}\in\mathbb{R}^3 \tag{28}
$$

We can now express the ten independent submatrices of the Hessian:

$$
\begin{aligned}
\frac{\partial^2\theta}{\partial \boldsymbol{x}_i^2} &= -\frac{\|r_{kj}\|}{\|\boldsymbol{m}\|^4}(\boldsymbol{m}\diamond(\boldsymbol{r_{kj}}\times\boldsymbol{m}))\\
\frac{\partial^2\theta}{\partial \boldsymbol{x}_i\partial \boldsymbol{x}_j} &= \frac{\boldsymbol{m}\otimes\boldsymbol{r_{kj}}}{\|\boldsymbol{m}\|^2\|\boldsymbol{r_{kj}}\|}+\frac{\|\boldsymbol{r_{kj}}\|}{\|\boldsymbol{m}\|^4}(\boldsymbol{m}\diamond((\boldsymbol{r_{kj}}-\boldsymbol{r_{ij}})\times\boldsymbol{m}))\\
\frac{\partial^2\theta}{\partial \boldsymbol{x}_i\partial \boldsymbol{x}_k} &= \frac{\boldsymbol{m}\otimes\boldsymbol{r_{kj}}}{\|\boldsymbol{m}\|^2\|\boldsymbol{r_{kj}}\|}+\frac{\|\boldsymbol{r_{kj}}\|}{\|\boldsymbol{m}\|^4}(\boldsymbol{m}\diamond(\boldsymbol{r_{ij}}\times\boldsymbol{m}))\\
\frac{\partial^2\theta}{\partial \boldsymbol{x}_i\partial \boldsymbol{x}_l} &= \mathbf{0}_{3\times3}\\
\frac{\partial^2\theta}{\partial \boldsymbol{x}_j\partial \boldsymbol{x}_l} &= \frac{\boldsymbol{r_{kj}}\otimes\boldsymbol{n}}{\|\boldsymbol{r_{kj}}\|\|\boldsymbol{n}\|^2}-\frac{\|\boldsymbol{r_{kj}}\|}{\|\boldsymbol{n}\|^4}(\boldsymbol{n}\diamond(\boldsymbol{r_{kl}}\times\boldsymbol{n}))\\
\frac{\partial^2\theta}{\partial \boldsymbol{x}_k\partial \boldsymbol{x}_l} &= \frac{\boldsymbol{r_{kj}}\otimes\boldsymbol{n}}{\|\boldsymbol{r_{kj}}\|\|\boldsymbol{n}\|^2}-\frac{\|\boldsymbol{r_{kj}}\|}{\|\boldsymbol{n}\|^4}(\boldsymbol{n}\diamond((\boldsymbol{r_{kj}}-\boldsymbol{r_{kl}})\times\boldsymbol{n}))\\
\frac{\partial^2\theta}{\partial \boldsymbol{x}_j^2} &= \frac{\partial\theta}{\partial \boldsymbol{x}_i}\otimes\frac{\partial A}{\partial \boldsymbol{x}_j}+(A-1)\frac{\partial^2\theta}{\partial \boldsymbol{x}_i\partial \boldsymbol{x}_j}-\left(\frac{\partial\theta}{\partial \boldsymbol{x}_l}\otimes\frac{\partial B}{\partial \boldsymbol{x}_j}+B\left(\frac{\partial^2\theta}{\partial \boldsymbol{x}_j\partial \boldsymbol{x}_l}\right)^\top\right)\\
\frac{\partial^2\theta}{\partial \boldsymbol{x}_j\partial \boldsymbol{x}_k} &= \frac{\partial\theta}{\partial \boldsymbol{x}_i}\otimes\frac{\partial A}{\partial \boldsymbol{x}_k}+(A-1)\frac{\partial^2\theta}{\partial \boldsymbol{x}_i\partial \boldsymbol{x}_k}-\left(\frac{\partial\theta}{\partial \boldsymbol{x}_l}\otimes\frac{\partial B}{\partial \boldsymbol{x}_k}+B\left(\frac{\partial^2\theta}{\partial \boldsymbol{x}_k\partial \boldsymbol{x}_l}\right)^\top\right)\\
\frac{\partial^2\theta}{\partial \boldsymbol{x}_k^2} &= \frac{\partial\theta}{\partial \boldsymbol{x}_l}\otimes\frac{\partial B}{\partial \boldsymbol{x}_k}+(B-1)\left(\frac{\partial^2\theta}{\partial \boldsymbol{x}_k\partial \boldsymbol{x}_l}\right)^\top-\left(\frac{\partial\theta}{\partial \boldsymbol{x}_i}\otimes\frac{\partial A}{\partial \boldsymbol{x}_k}+A\frac{\partial^2\theta}{\partial \boldsymbol{x}_i\partial \boldsymbol{x}_k}\right)\\
\frac{\partial^2\theta}{\partial \boldsymbol{x}_l^2} &= \frac{\|\boldsymbol{r_{kj}}\|}{\|\boldsymbol{n}\|^4}(\boldsymbol{n}\diamond(\boldsymbol{r_{kj}}\times\boldsymbol{n}))
\end{aligned}
\tag{29}
$$

The other six 3×3 blocks are found by:

$$
\begin{aligned}
\frac{\partial^2\theta}{\partial \boldsymbol{x}_j\partial \boldsymbol{x}_i} &= \left(\frac{\partial^2\theta}{\partial \boldsymbol{x}_i\partial \boldsymbol{x}_j}\right)^\top\\
\frac{\partial^2\theta}{\partial \boldsymbol{x}_k\partial \boldsymbol{x}_l} &= \left(\frac{\partial^2\theta}{\partial \boldsymbol{x}_l\partial \boldsymbol{x}_k}\right)^\top\\
\frac{\partial^2\theta}{\partial \boldsymbol{x}_k\partial \boldsymbol{x}_j} &= \left(\frac{\partial^2\theta}{\partial \boldsymbol{x}_j\partial \boldsymbol{x}_k}\right)^\top\\
\frac{\partial^2\theta}{\partial \boldsymbol{x}_l\partial \boldsymbol{x}_i} &= \left(\frac{\partial^2\theta}{\partial \boldsymbol{x}_i\partial \boldsymbol{x}_l}\right)^\top\\
\frac{\partial^2\theta}{\partial \boldsymbol{x}_l\partial \boldsymbol{x}_j} &= \left(\frac{\partial^2\theta}{\partial \boldsymbol{x}_j\partial \boldsymbol{x}_l}\right)^\top\\
\frac{\partial^2\theta}{\partial \boldsymbol{x}_l\partial \boldsymbol{x}_k} &= \left(\frac{\partial^2\theta}{\partial \boldsymbol{x}_k\partial \boldsymbol{x}_l}\right)^\top
\end{aligned}
\tag{30}
$$

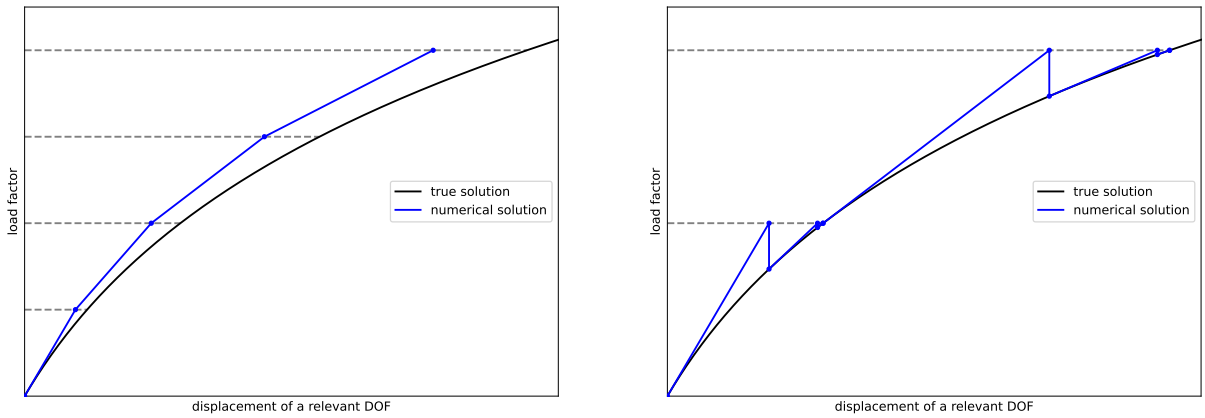
A.5 Introduction to path-following techniques

Generally, there are two types of path-following strategies: load-control and displacement-control. Here we will only discuss the load-control method, which itself can be split into two categories: purely incremental methods and incremental-iterative methods [47].

The simpler of the two is purely incremental methods, illustrated in figure 31a and algorithm 5. In these methods, we start by setting a target load factor for the first step, evaluate the tangent stiffness matrix and compute the displacements. That gives a point on the numerical equilibrium path; that point is not on the true equilibrium path, because of the nonlinearity. In the next step, we set a new target load factor, evaluate the tangent stiffness at the newfound point and compute the new displacements, giving us a new point on the path,

most likely even further from the true solution. We can continue this until we reach a load that is satisfactory for the specific analysis goals.

The "drifting away" behaviour of the purely incremental methods are solved by adding iterations within the load factor increments. These improved procedures are called incremental-iterative methods, illustrated in figure 31b and algorithm 6. In those methods, we calculate the residual forces, i.e. the difference between the external forces and the internal forces, to see if the computed equilibrium point is close to the true equilibrium path. If it is not, the tangent stiffness matrix is evaluated at the most recently discovered point and displacement values that are attributable to the residual forces are computed. This process repeats until the residual forces almost equal zero, after which the next increment can begin with a similar cycle. Newton's method is the easiest variant of incremental-iterative algorithms and that is the method that is used in figure 31b and algorithm 6.



(a) The result of a purely incremental solution procedure (b) The result of an incremental-iterative solution procedure

Figure 31: Two categories of path-following methods to solve nonlinear problems

Algorithm 5 Pseudo-code of a purely incremental procedure

$t = 0$ # *increment counter*
$\boldsymbol{U_t} = \mathbf{0}$ # *initial displacement is zero*
$\lambda_t = 0$ # *initial load factor is zero*
$d\lambda = 0.01$ # *load factor step*

while $\lambda < 1$ # *until we reach a load factor of 1*
  calculate $\boldsymbol{K_t}(\boldsymbol{U_t})$ # *stiffness matrix at last force-displacement point*

  $t = t + 1$ # *increase increment counter*
  $\lambda_t = \lambda_{t-1} + d\lambda$ # *increase load factor*
  $\boldsymbol{F_t^{ext}} = \lambda_t \cdot \boldsymbol{F}$ # *setting new load*

  $\boldsymbol{\Delta U_{t-1}} = \boldsymbol{K_t^{-1}}(\boldsymbol{F_t^{ext}} - \boldsymbol{F_{t-1}^{ext}})$ # *calculating displacement step*
  $\boldsymbol{U_t} = \boldsymbol{U_{t-1}} + \boldsymbol{\Delta U_{t-1}}$ # *updating displacement with displacement step*

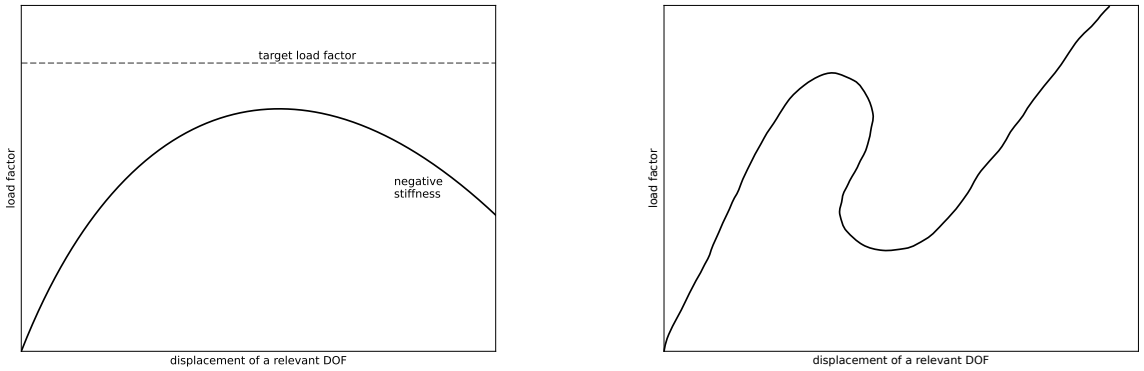
Algorithm 6 Pseudo-code of an incremental-iterative procedure (Newton's method)

```
t = 0                                   # increment counter
j = 0                                   # iteration counter
U_{t,0} = 0                             # initial displacement is zero
λ_t = 0                                 # initial load factor is zero
dλ = 0.01                               # load factor step
tol = 0.00001                           # tolerance of residual loads

while λ_t < 1                           # until we reach a load factor of 1
    t = t + 1                           # increase increment counter
    U_{t,0} = U_{t-1,j}                 # set first displacement of this incr. to the final value of the previous incr.

    j = 0                               # reset iteration counter
    λ_t = λ_{t-1} + dλ                  # increase load factor
    F_t^{ext} = λ_t · F                 # setting new load

    calculate F_{t,j}^{int}(U_{t,j})    # internal forces
    R_{t,j} = F_t^{ext} − F_{t,j}^{int} # residual load
    err = 1000                          # any large number

    while err > tol · ‖(F_t^{ext}‖      # until the residuals are almost zero
        calculate K_{t,j}(U_{t,j})      # stiffness matrix at last force-displacement point
        dU_{t,j} = K_{t,j}^{-1} R_{t,j} # calculating displacement step

        j = j + 1                       # increase iteration counter
        U_{t,j} = U_{t,j-1} + dU_{t,j-1} # updating displacement with displacement step

        calculate F_{t,j}^{int}(U_{t,j}) # internal forces at new force-displacement point
        R_{t,j} = F_t^{ext} − F_{t,j}^{int} # residual loads at new force-displacement point
        err = ‖R_{t,j}‖                 # magnitude of the residual load
```

There are several extensions and variations on incremental-iterative algorithms that will be discussed now, most are also applicable to purely incremental algorithms. Some are also discussed in section 2. First, there is the choice of initial load factor step, which can influence the equilibrium path the method will take, particularly in more advanced incremental-iterative methods such as the arc-length method. Secondly, there is the possibility to change the load factor step while the algorithm is running; when there are very many iterations within an increment, that means that convergence is slow or even not happening at all, which is a sign that the load step is too large. We can set a maximum to the number of iterations, and if this maximum is exceeded, we can reduce the load factor step and reset to the beginning of the increment to try again. And there is the opposite case: if it takes very few (i.e. one or two) iterations to converge, we are wasting computing power and we should increase the load factor step as we go to the next increment. This mid-execution changing of the load factor step can bring problems. Suppose we are not just looking for the general equilibrium path, but are instead interested in the displacement field at a specific load magnitude. We are unlikely to reach this specific load if the load factor step is unpredictable. The problem is that we might overshoot this specific load magnitude and thus never get to know the displacements at that specific load. A solution to that problem is to take the increment that overshot, calculate the fraction of the load factor step that overshoots the target load factor, and finally, subtract an equal fraction of the final displacement step from the displacement vector to come to the displacement vector at the target load factor. On a different note, another choice we can make is the value for the tolerance on the residual forces to reach zero. That choice will control the number of iterations during each step, which in turn influences the speed and accuracy of the solution. We can also choose to add unloading to the procedure, in which the load factor is reduced every step instead of increased. In fully elastic models, the loading and unloading equilibrium paths will overlap. Finally, the calculations of the tangent stiffness matrix

and the internal forces share a lot of computationally expensive steps, and it is thus beneficial to combine the two calculations. This requires some reordering of the algorithm, which reduces understandability but increases the speed.

The simplest incremental-iterative implementations (such as Newton's method) will run into problems if there is an extraordinary equilibrium path that must be followed: a path that includes negative stiffness (figure 32a) or even snap-back behaviour (figure 32b) will lead to failure of the procedure.



(a) An equilibrium path with negative stiffness

(b) Snap-back behaviour

Figure 32: Two equilibrium paths that can not be followed using Newton's method

A.6 Alternative method for calculating the center of mass

Instead of calculating the center of mass of the nodes that make up the bar-and-hinge model, we can also calculate the center of mass of the origami design using

$$\boldsymbol{c} = \frac{1}{A_{\text{total}}} \sum_{i=0}^{N_{\text{triangles}}-1} \boldsymbol{c_i} \cdot A_i \tag{31}$$

where we take the average of the centers of mass of the triangles in the bar-and-hinge model. The center of mass of a single triangle can be computed by

$$\boldsymbol{c_i} = \frac{1}{3}(\boldsymbol{n_0} + \boldsymbol{n_1} + \boldsymbol{n_2}) \tag{32}$$

where $\boldsymbol{n_0}$, $\boldsymbol{n_1}$, and $\boldsymbol{n_2}$ are the position vectors of the corner nodes of the triangle. And the area of a single triangle can be computed by

$$A_i = \frac{1}{2} \left\| \boldsymbol{v_{01}} \times \boldsymbol{v_{02}} \right\|. \tag{33}$$

where $\boldsymbol{v_{01}}$ and $\boldsymbol{v_{02}}$ are the vectors from $\boldsymbol{n_0}$ to $\boldsymbol{n_1}$ and $\boldsymbol{n_2}$, respectively.

A.7 Alternative method for calculating the inertia tensor

Instead of calculating a pseudo-inertia tensor, we can also calculate the inertia tensor with a continuum approach by summing the individual inertia contributions of the triangular surfaces that can be extracted from the bar-and-hinge model. The inertia of a surface is expressed by the symmetric tensor

$$\boldsymbol{I} = \begin{bmatrix} I_{xx} & I_{xy} & I_{xz} \\ I_{yx} & I_{yy} & I_{yz} \\ I_{zx} & I_{zy} & I_{zz} \end{bmatrix} = \begin{bmatrix} \rho \int_A (y^2 + z^2) dA & \rho \int_A (-x \cdot y) dA & \rho \int_A (-x \cdot z) dA \\ \rho \int_A (-y \cdot x) dA & \rho \int_A (x^2 + z^2) dA & \rho \int_A (-y \cdot z) dA \\ \rho \int_A (-z \cdot x) dA & \rho \int_A (-z \cdot y) dA & \rho \int_A (x^2 + y^2) dA \end{bmatrix} \tag{34}$$

We can also set the surface density ρ to 1 and reach

$$\boldsymbol{I} = \begin{bmatrix} \int_A (y^2+z^2)dA & -\int_A (x\cdot y)dA & -\int_A (x\cdot z)dA \\ -\int_A (y\cdot x)dA & \int_A (x^2+z^2)dA & -\int_A (y\cdot z)dA \\ -\int_A (z\cdot x)dA & -\int_A (z\cdot y)dA & \int_A (x^2+y^2)dA \end{bmatrix} \tag{35}$$

The triangle is defined by three points, $\boldsymbol{n_0}$, $\boldsymbol{n_1}$, and $\boldsymbol{n_2}$, and the end points of the longest side of the triangle must be $\boldsymbol{n_0}$ and $\boldsymbol{n_1}$. These integrals over the triangle are difficult to compute, so we transform the triangle to a position where these integrals become easier, see figure 33 for a schematic of the transformation process. The first step in this transformation is making the triangular surface parallel to the xy-plane using algorithm 7, which uses the rotation matrix $\boldsymbol{R_1}$ to obtain the new node vectors $\boldsymbol{n_0}^{\text{flat}}$, $\boldsymbol{n_1}^{\text{flat}}$, and $\boldsymbol{n_2}^{\text{flat}}$.

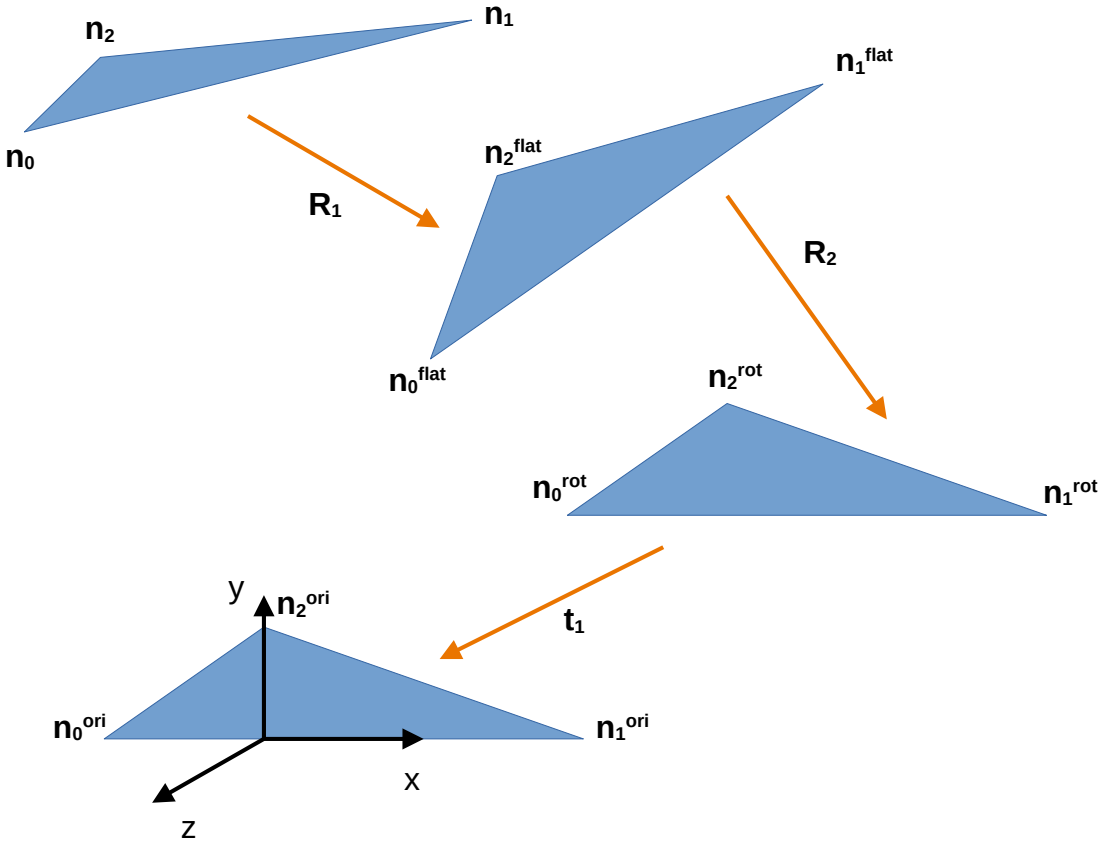


Figure 33: A schematic of the transformation of a triangle

Algorithm 7 Rotating a triangular surface such that it is parallel to the xy-plane

$\boldsymbol{v_{01}} = (\boldsymbol{n_1} - \boldsymbol{n_0}) / \|\boldsymbol{n_1} - \boldsymbol{n_0}\|$
$\boldsymbol{v_{02}} = (\boldsymbol{n_2} - \boldsymbol{n_0}) / \|\boldsymbol{n_2} - \boldsymbol{n_0}\|$
$\boldsymbol{n} = (\boldsymbol{v_{01}} \times \boldsymbol{v_{02}}) / \|\boldsymbol{v_{01}} \times \boldsymbol{v_{02}}\|$
$\boldsymbol{z} = \begin{bmatrix} 0 & 0 & 1 \end{bmatrix}^\top$
$\boldsymbol{a} = \begin{bmatrix} a_x \\ a_y \\ a_z \end{bmatrix} = (\boldsymbol{n} \times \boldsymbol{z}) / \|\boldsymbol{n} \times \boldsymbol{z}\|$
$\theta = \arccos(\boldsymbol{n} \cdot \boldsymbol{z})$
$\boldsymbol{R_1} = \begin{bmatrix} \cos\theta + a_x^2(1-\cos\theta) & a_x a_y(1-\cos\theta) - a_z\sin\theta & a_x a_z(1-\cos\theta) + a_y\sin\theta \\ a_y a_x(1-\cos\theta) + a_z\sin\theta & \cos\theta + a_y^2(1-\cos\theta) & a_y a_z(1-\cos\theta) - a_x\sin\theta \\ a_z a_x(1-\cos\theta) - a_y\sin\theta & a_z a_y(1-\cos\theta) + a_x\sin\theta & \cos\theta + a_z^2(1-\cos\theta) \end{bmatrix}$
$\boldsymbol{n_i}^{\text{flat}} = \boldsymbol{R_1}\, \boldsymbol{n_i} \quad \forall\, i \in \{0, 1, 2\}$

The second step in the transformation is another rotation such that the longest side of the triangle is parallel to the x-axis, using algorithm 8.

Algorithm 8 Rotating a triangular surface such that the longest side is parallel to the x-axis

$$\boldsymbol{v_{01}} = (\boldsymbol{n_1^{\text{flat}}} - \boldsymbol{n_0^{\text{flat}}}) / \left\| \boldsymbol{n_1^{\text{flat}}} - \boldsymbol{n_0^{\text{flat}}} \right\|$$
$$\boldsymbol{x} = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix}^\top$$
$$\theta = \arccos(\boldsymbol{v_{01}} \cdot \boldsymbol{x})$$
$$\boldsymbol{R_2} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$
$$\boldsymbol{n_i^{\text{rot}}} = \boldsymbol{R_2}\, \boldsymbol{n_i^{\text{flat}}} \quad \forall\, i \in \{0, 1, 2\}$$

And finally, the third step is a translation such that the triangle is positioned on the xy-plane, its longest side is collinear with the x-axis and node $\boldsymbol{n_2^{\text{ori}}}$ is on the y-axis, see algorithm 9 and figure 33.

Algorithm 9 Translating a triangular surface towards the origin

$$\boldsymbol{t_1} = \begin{bmatrix} -n_{2,x}^{\text{rot}} & -n_{0,y}^{\text{rot}} & -n_{0,z}^{\text{rot}} \end{bmatrix}^\top$$
$$\boldsymbol{n_i^{\text{ori}}} = \boldsymbol{n_i^{\text{rot}}} + \boldsymbol{t_1} \quad \forall\, i \in \{0, 1, 2\}$$

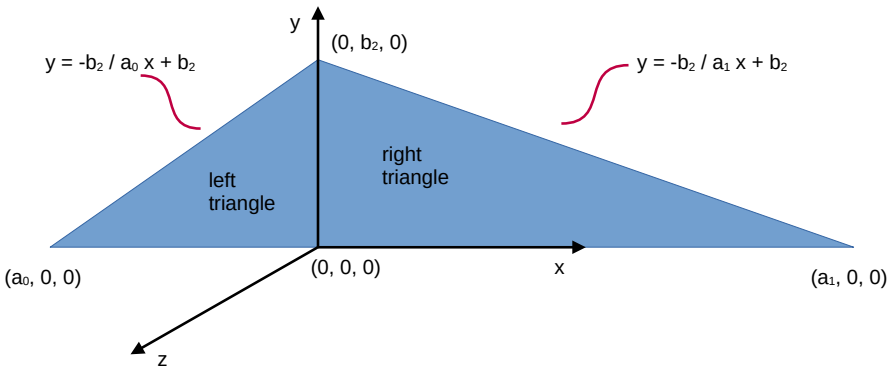


Figure 34: A triangle for which we can compute the inertia

This conveniently-placed triangle is split into a left triangle and a right triangle, for which we can compute the inertia tensors. Looking at equation 35, we can see that $I_{xz} = I_{zx} = I_{yz} = I_{zy} = 0$, since the triangle has no z-component. By also looking at figure 34, where a_0, a_1, and b_2 are defined, we can derive the integration bounds and reach

$$\boldsymbol{I_{left}^{\text{ori}}} = \begin{bmatrix} \int_{a_0}^{0} \int_{0}^{-\frac{b_2}{a_0}x+b_2} y^2 \, dydx & -\int_{a_0}^{0} \int_{0}^{-\frac{b_2}{a_0}x+b_2} x \cdot y \, dydx & 0 \\ -\int_{a_0}^{0} \int_{0}^{-\frac{b_2}{a_0}x+b_2} x \cdot y \, dydx & \int_{a_0}^{0} \int_{0}^{-\frac{b_2}{a_0}x+b_2} x^2 \, dydx & 0 \\ 0 & 0 & \int_{a_0}^{0} \int_{0}^{-\frac{b_2}{a_0}x+b_2} (x^2+y^2) \, dydx \end{bmatrix} \tag{36}$$

and

$$\boldsymbol{I_{right}^{\text{ori}}} = \begin{bmatrix} \int_{0}^{a_1} \int_{0}^{\frac{b_2}{a_1}x+b_2} y^2 \, dydx & -\int_{0}^{a_1} \int_{0}^{\frac{b_2}{a_1}x+b_2} x \cdot y \, dydx & 0 \\ -\int_{0}^{a_1} \int_{0}^{\frac{b_2}{a_1}x+b_2} x \cdot y \, dydx & \int_{0}^{a_1} \int_{0}^{\frac{b_2}{a_1}x+b_2} x^2 \, dydx & 0 \\ 0 & 0 & \int_{0}^{a_1} \int_{0}^{\frac{b_2}{a_1}x+b_2} (x^2+y^2) \, dydx \end{bmatrix} \tag{37}$$

which can be simplified to

$$\boldsymbol{I}^{\mathbf{ori}}_{\boldsymbol{left}} = \begin{bmatrix} -\frac{1}{12}a_0 b_2^3 & \frac{1}{24}a_0^2 b_2^2 & 0 \\ \frac{1}{24}a_0^2 b_2^2 & -\frac{1}{12}b_2 a_0^3 & 0 \\ 0 & 0 & -\frac{1}{12}a_0 b_2^3 - \frac{1}{12}b_2 a_0^3 \end{bmatrix} \tag{38}$$

and

$$\boldsymbol{I}^{\mathbf{ori}}_{\boldsymbol{right}} = \begin{bmatrix} \frac{1}{12}a_1 b_2^3 & -\frac{1}{24}a_1^2 b_2^2 & 0 \\ -\frac{1}{24}a_1^2 b_2^2 & \frac{1}{12}b_2 a_1^3 & 0 \\ 0 & 0 & \frac{1}{12}a_1 b_2^3 + \frac{1}{12}b_2 a_1^3 \end{bmatrix} \tag{39}$$

We sum these tensors to get the complete tensor of the left and right triangles combined: $\boldsymbol{I}^{\mathbf{ori}} = \boldsymbol{I}^{\mathbf{ori}}_{\boldsymbol{left}} + \boldsymbol{I}^{\mathbf{ori}}_{\boldsymbol{right}}$. This inertia tensor must now be transformed back, such that it represents the triangle we started with. The higher level generalized parallel axis theorem [52] provides a method for accounting for the translation $\boldsymbol{t_1}$:

$$\boldsymbol{I}^{\mathbf{rot}} = \boldsymbol{I}^{\mathbf{ori}} + A \begin{bmatrix} t_y^2 + t_z^2 - 2t_y c_y - 2t_z c_z & -t_x t_y + t_x c_y + t_y c_x & -t_x t_z + t_x c_z + t_z c_x \\ -t_y t_x + t_y c_x + t_x c_y & t_x^2 + t_z^2 - 2t_x c_x - 2t_z c_z & -t_y t_z + t_y c_z + t_z c_y \\ -t_z t_x + t_z c_x + t_x c_z & -t_z t_y + t_z c_y + t_y c_z & t_x^2 + t_y^2 - 2t_x c_x - 2t_y c_y \end{bmatrix} \tag{40}$$

where A is the area of the triangle, which is $\frac{1}{2}(-a_0 b_2 + a_1 b_2)$. t_x, t_y, and t_z are the components of $\boldsymbol{t_1}$ and c_x, c_y, and c_z are the components of the center of mass of the triangle, $\boldsymbol{c}$. This $\boldsymbol{c}$ can be computed by $\frac{1}{3}(\boldsymbol{n_0}^{\mathrm{ori}} + \boldsymbol{n_1}^{\mathrm{ori}} + \boldsymbol{n_2}^{\mathrm{ori}})$. From $\boldsymbol{I}^{\mathbf{rot}}$ we work back to $\boldsymbol{I}^{\mathbf{flat}}$ by

$$\boldsymbol{I}^{\mathbf{flat}} = \boldsymbol{R_2}^{\top} \boldsymbol{I}^{\mathbf{rot}} \boldsymbol{R_2} \tag{41}$$

and finally to $\boldsymbol{I}$ with

$$\boldsymbol{I} = \boldsymbol{R_1}^{\top} \boldsymbol{I}^{\mathbf{flat}} \boldsymbol{R_1} \tag{42}$$

which is the inertia tensor of the triangle with corner nodes $\boldsymbol{n_0}$, $\boldsymbol{n_1}$, and $\boldsymbol{n_2}$. As discussed, the total inertia tensor of an origami can be constructed by summing the inertia tensors of the individual triangles:

$$\boldsymbol{I}_{\mathbf{total}} = \sum_{i=0}^{N_{\mathrm{triangles}}-1} \boldsymbol{I_i} \tag{43}$$

This procedure was tested by comparing the result with another approach to calculating the inertia tensor. This approach involves meshing the triangles and summing the inertia contributions of the small elements, which are considered point masses, located at their centroid:

$$\boldsymbol{I}^{\mathbf{mesh}}_{\mathbf{total}} = \sum_{i=0}^{N_{\mathrm{triangles}}-1} \boldsymbol{I}^{\mathbf{mesh}}_{\boldsymbol{i}} = \sum_{i=0}^{N_{\mathrm{triangles}}-1} \sum_{j=0}^{N_{\mathrm{elems}}-1} A_j \begin{bmatrix} c_{j,y}^2 + c_{j,z}^2 & -c_{j,x}c_{j,y} & -c_{j,x}c_{j,z} \\ -c_{j,y}c_{j,x} & c_{j,x}^2 + c_{j,z}^2 & -c_{j,y}c_{j,z} \\ -c_{j,z}c_{j,y} & -c_{j,z}c_{j,x} & c_{j,x}^2 + c_{j,y}^2 \end{bmatrix} \tag{44}$$

As the mesh was refined, the error between $\boldsymbol{I}^{\mathbf{mesh}}_{\mathbf{total}}$ and the analytically expressed $\boldsymbol{I}_{\mathbf{total}}$ was reduced, which indicates that the expression for $\boldsymbol{I}_{\mathbf{total}}$ is correct. The test was performed for a single triangle and a complete origami saddle.

A.8 Alternative method for orienting an origami design such that it aligns with the target shape

In section 2.2.2, we assumed that the origami design's orientation was already close to the target surface's orientation before we aligned it. Here we will discuss a more robust method of orienting the origami such that the process also works when this assumption does not hold.

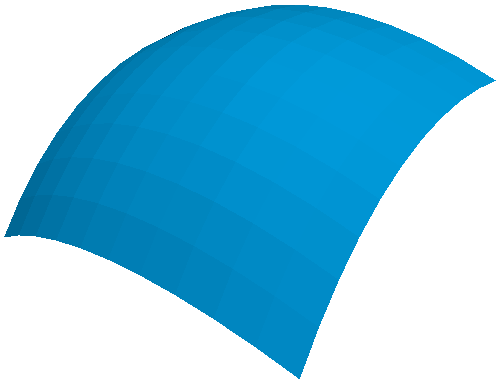
First, we start by calculating the inertia tensor of the target shape—either using the method in section 2.2.2 or the method in appendix A.7—and name it $\boldsymbol{I}_{\text{target}}$. From this inertia tensor we compute the pairs of principle moments of inertia and principle axes of inertia: $\{\lambda_0, \boldsymbol{v_0}\}$, $\{\lambda_1, \boldsymbol{v_1}\}$, and $\{\lambda_2, \boldsymbol{v_2}\}$, ordered from smallest to largest moment of inertia. We place the principle axes in a matrix $\boldsymbol{V}_{\text{target}} = \begin{bmatrix} \boldsymbol{v_0} & \boldsymbol{v_1} & \boldsymbol{v_2} \end{bmatrix}$ and when there are negative values on the diagonal, we multiply the affected column by -1, which removes unwanted reflections that would otherwise occur at a later stage. We repeat this process for the origami design and retrieve $\boldsymbol{V}_{\text{design}}$. With these matrices, we compute a rotation matrix

$$\boldsymbol{R} = \boldsymbol{V}_{\text{target}} \boldsymbol{V}_{\text{design}}^{\top} \tag{45}$$

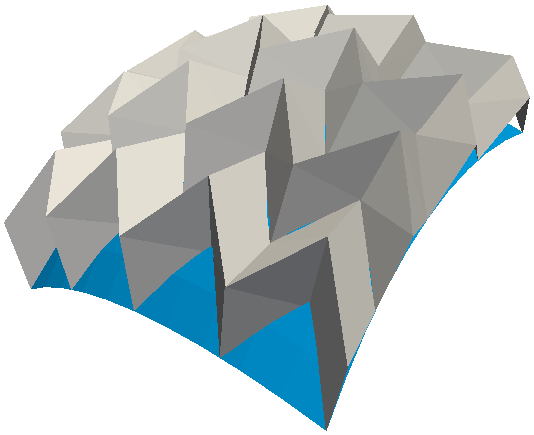
which is used to transform the displaced nodes of the origami design, in the same manner as described in section 2.2.2, whereafter the design is correctly oriented with respect to the target surface.

A.9 Rigid origami optimization results for the dome and arch shapes

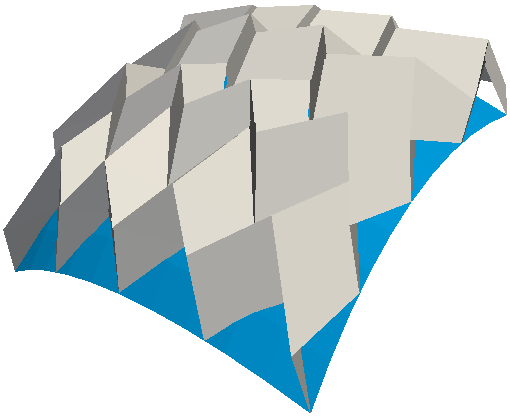
Dome



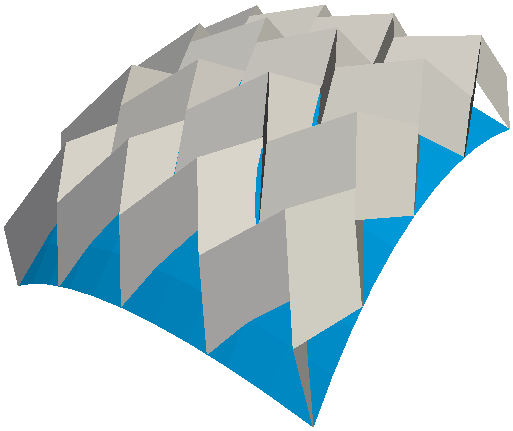
(a) Target shape



(b) Design made by placing Miura-ori-like unit cells (not foldable)

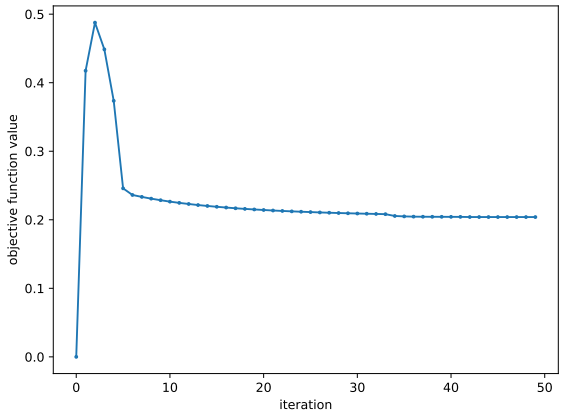


(c) Intermediate design during optimization

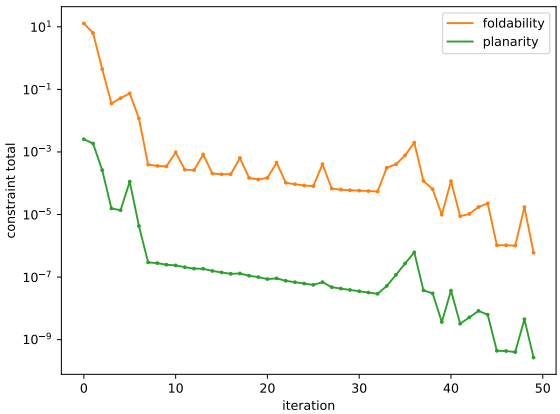


(d) Optimized design which is foldable

Figure 35: Sequence of steps for optimizing a rigid origami dome design such that it is foldable



(a) Objective function during optimization



(b) Sum of constraints during optimization (logarithmic vertical axis)

Figure 36

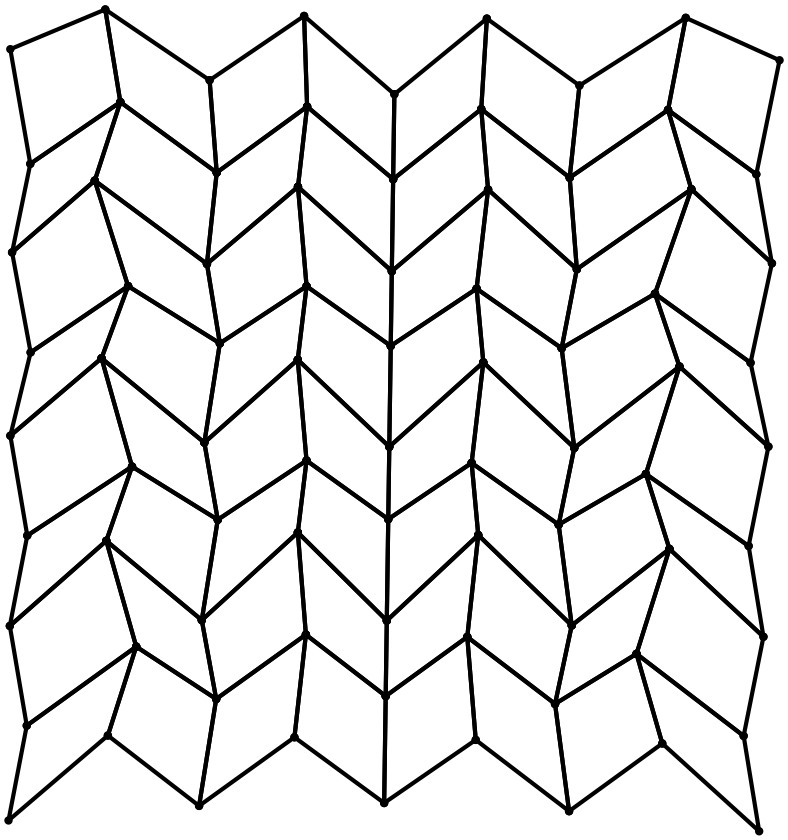
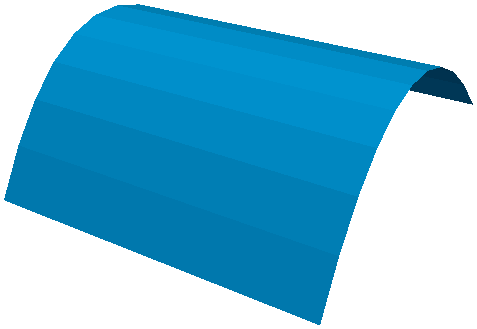
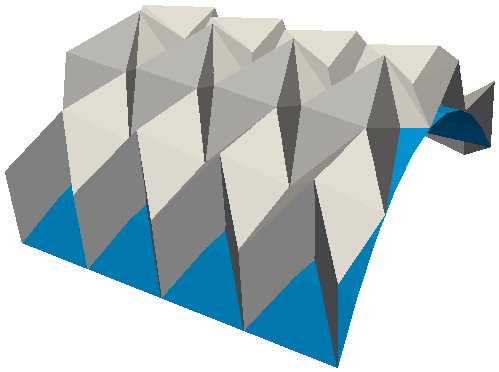


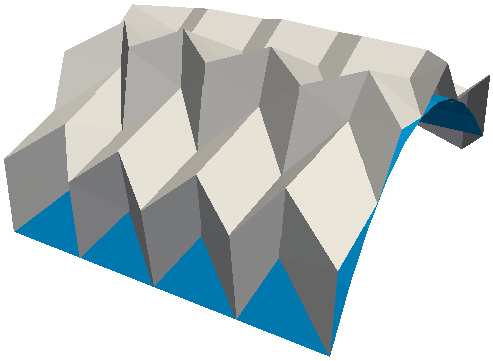
Figure 37: Flattened pattern of the optimized dome design



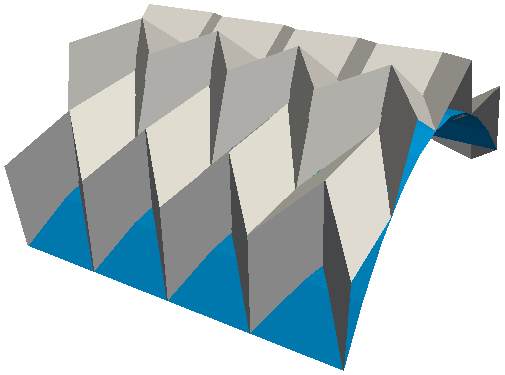
(a) Target shape



(b) Design made by placing Miura-ori-like unit cells (not foldable)

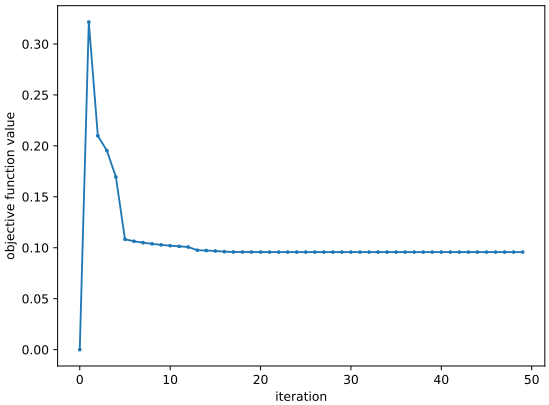


(c) Intermediate design during optimization

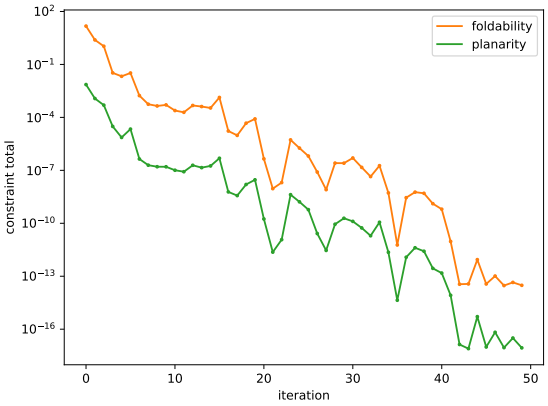


(d) Optimized design which is foldable

Figure 38: Sequence of steps for optimizing a rigid origami arch design such that it is foldable



(a) Objective function during optimization



(b) Sum of constraints during optimization (logarithmic vertical axis)

Figure 39

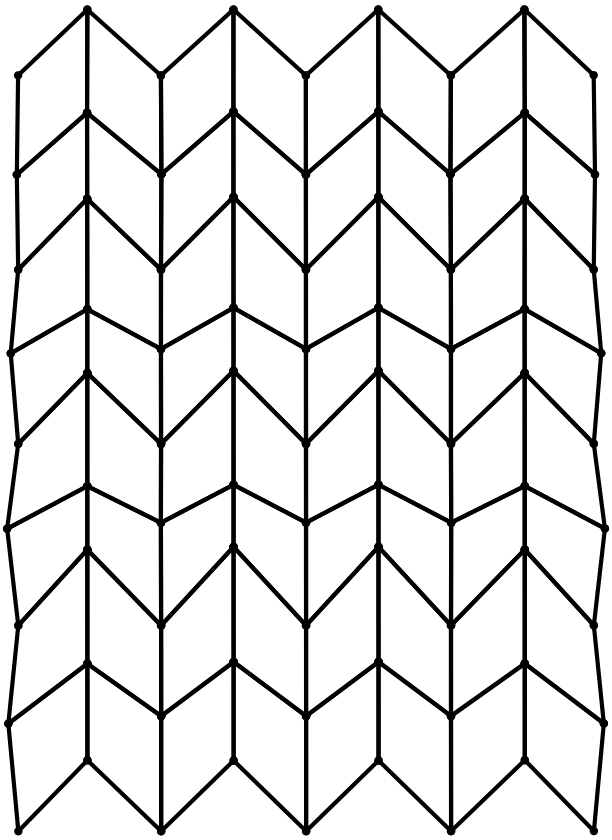
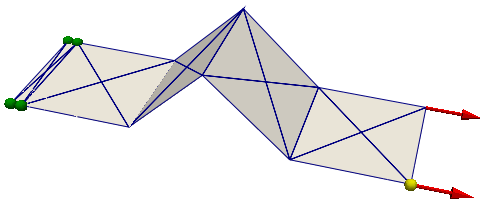


Figure 40: Flattened pattern of the optimized arch design

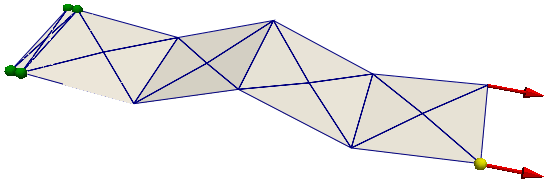
A.10 Bar-and-hinge model folding capabilities

Below are the mechanical analyses of the "bump", the "curl", the unfolding of a Miura-ori sheet, and the compression of a Miura-ori sheet. These display the capabilities of the bar-and-hinge model and the arc-length method. To the left are the physical configurations and deformed states of the origami, with the constrained nodes in green (note that not necessarily all three components are constrained), the forces in red, and the relevant DOF for measuring displacement in yellow. On the right, the figure depicts the development of the load factor over the arc-length iterations, and the force-displacement curve.

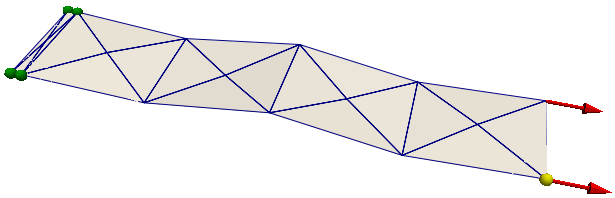
A.10.1 Bump



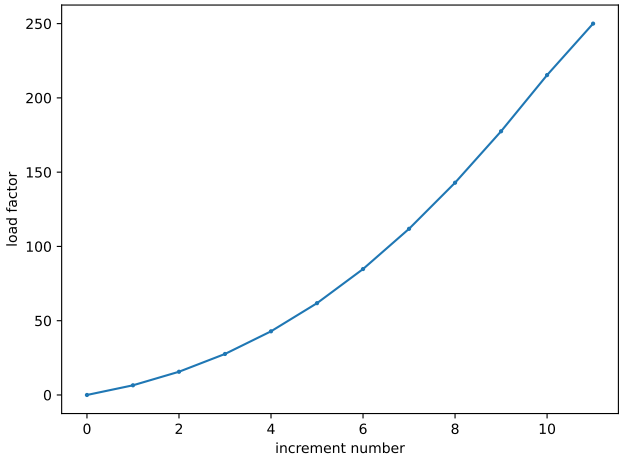
(a) Initial state of the origami



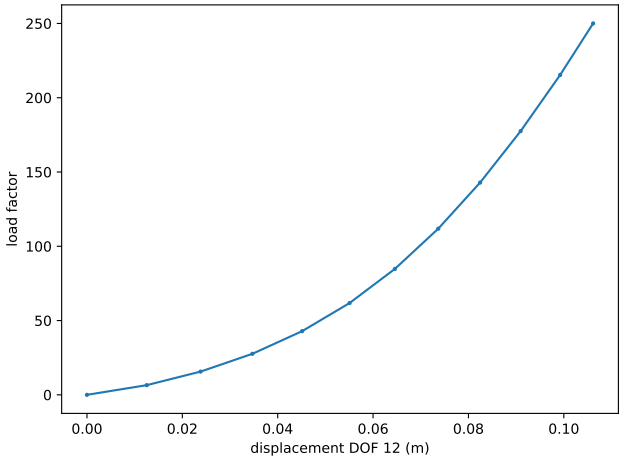
(b) Intermediate state of the origami



(c) Final state of the origami



(d) Load factor over the increments of the arc-length method

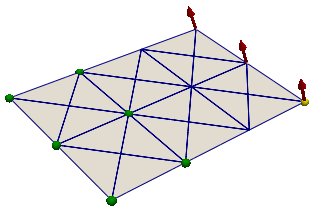


(e) Force-displacement graph

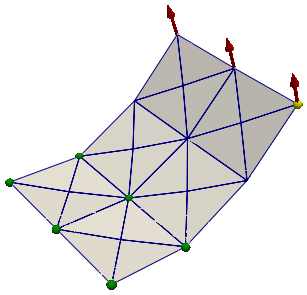
Figure 41: Results of the mechanical simulation of the "bump" origami

Watch the animation here: https://www.youtube.com/watch?v=uZnq9s4437U

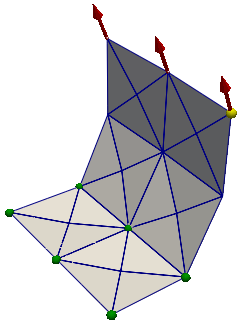
A.10.2 Curl



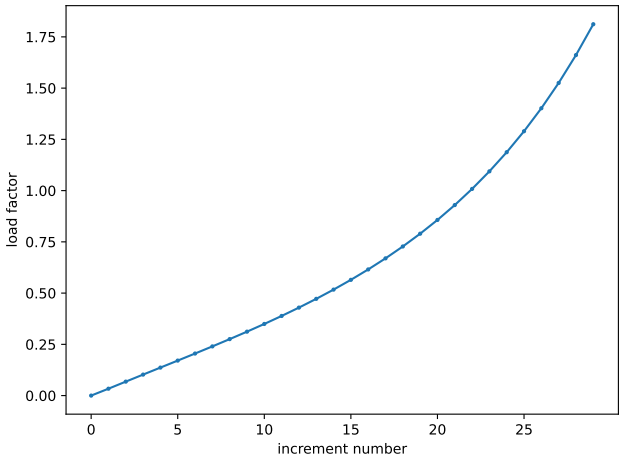
(a) Initial state of the origami



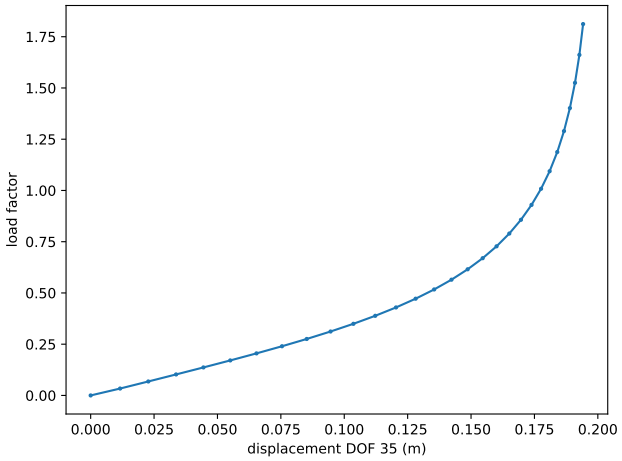
(b) Intermediate state of the origami



(c) Final state of the origami



(d) Load factor over the increments of the arc-length method

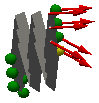


(e) Force-displacement graph

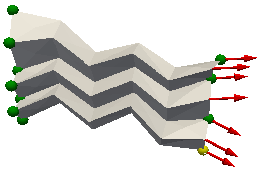
Figure 42: Results of the mechanical simulation of the “curl” origami

Watch the animation here: https://www.youtube.com/watch?v=xnzffUaizFs

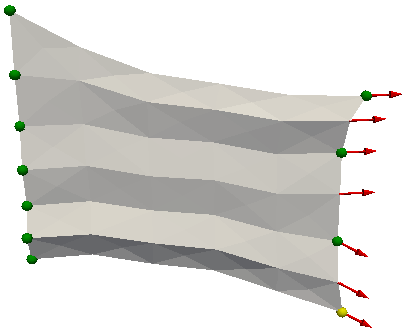
A.10.3 Unfolding of a Miura-ori sheet



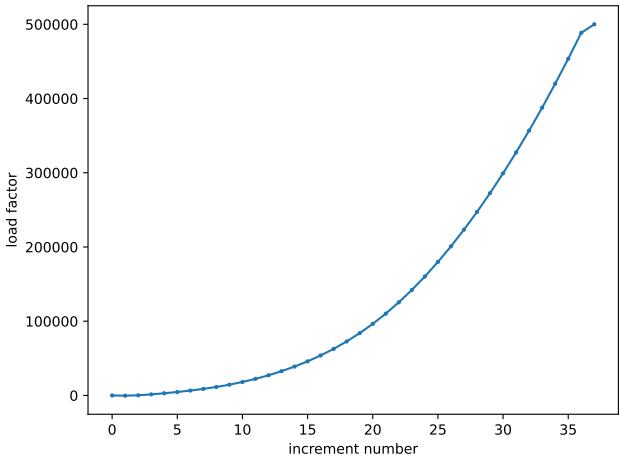
(a) Initial state of the origami



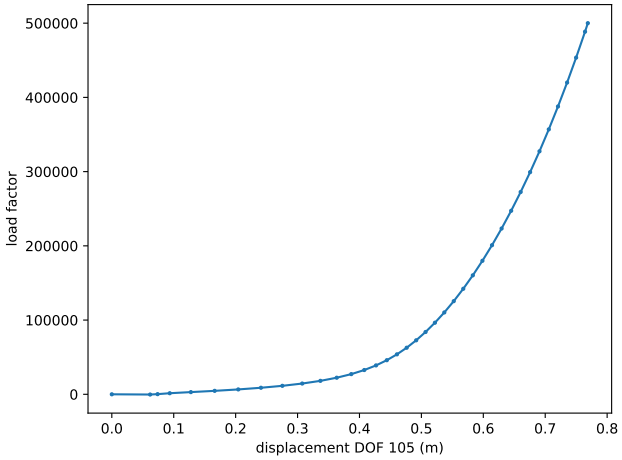
(b) Intermediate state of the origami



(c) Final state of the origami



(d) Load factor over the increments of the arc-length method

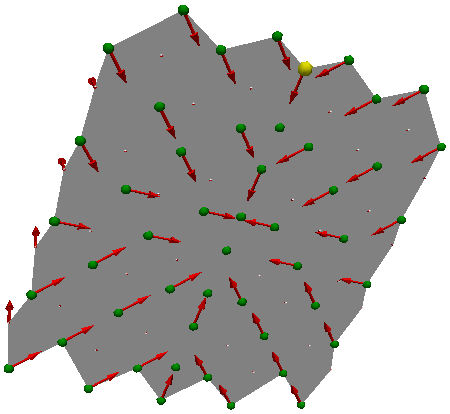


(e) Force-displacement graph

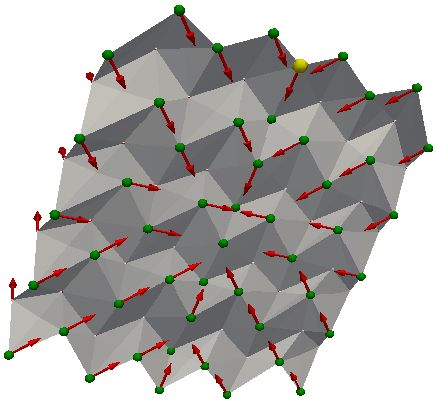
Figure 43: Results of the mechanical simulation of the unfolding of a Miura-ori sheet

Watch the animation here: https://www.youtube.com/watch?v=J7mLWyzeweE

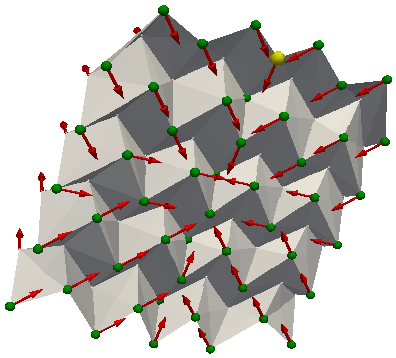
A.10.4 Compression of a Miura-ori sheet



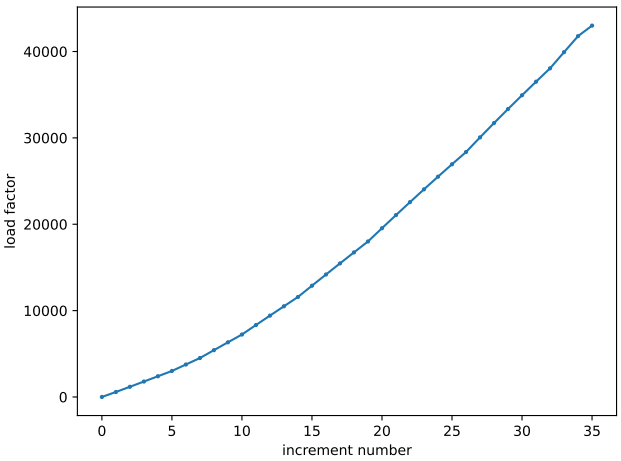
(a) Initial state of the origami



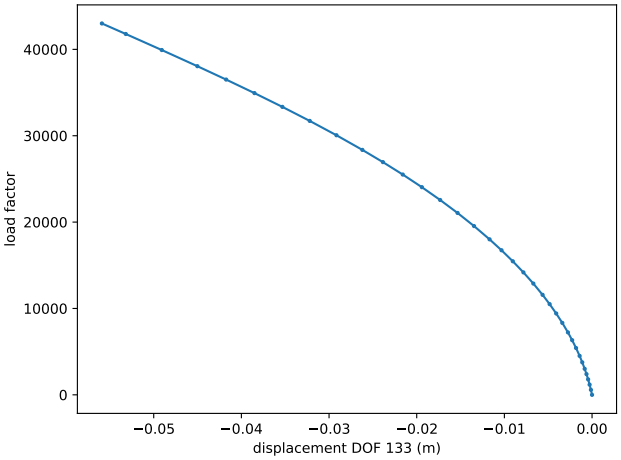
(b) Intermediate state of the origami



(c) Final state of the origami



(d) Load factor over the increments of the arc-length method



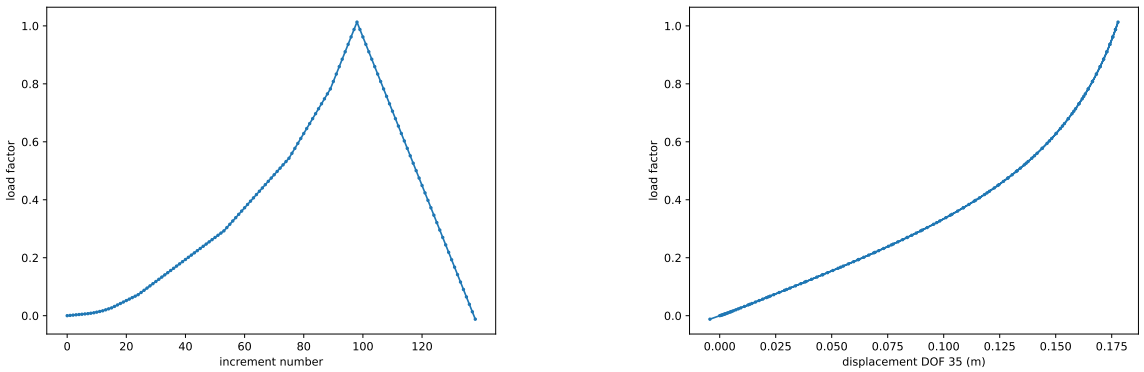
(e) Force-displacement graph

Figure 44: Results of the mechanical simulation of the compression of a flat Miura-ori sheet

Watch the animation here: https://www.youtube.com/watch?v=GdJv5cU6e9Y

A.11 Elasticity of the bar-and-hinge model

The bar-and-hinge model is an elastic model and that was verified. Using Newton's method as the path-following technique, an origami design was loaded and unloaded. In other words: the forces on the origami were incrementally increased to a maximum and then incrementally decreased to zero. The force-displacement graph of the loading and unloading sequences overlapped, see figure 45b.



(a) Load factor over the increments of Newton's method

(b) Force-displacement graph

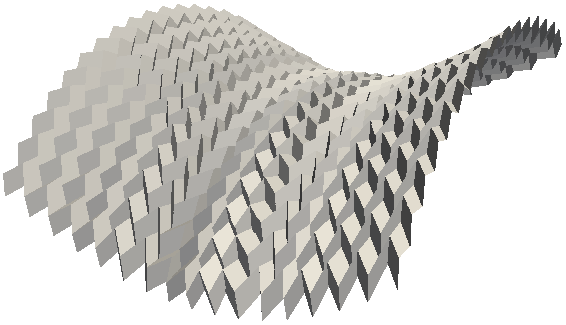
Figure 45: Graphs showing the elasticity of the bar-and-hinge model

A.12 Reducing the computation time of the bar-and-hinge model

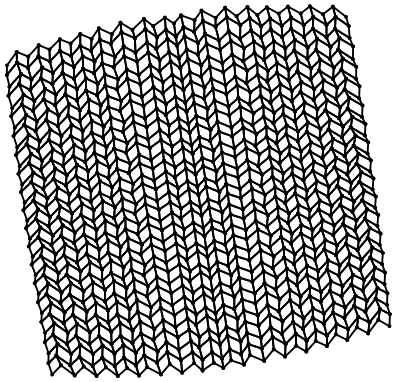
The computation time of the model was reduced fourfold by the use of sparse arrays from SciPy's library. Different kinds of sparse arrays were used in the proper situations: diagonal arrays (`dia_array`) for diagonal matrices, coordinate format arrays (`coo_array`) for constructing matrices, and compressed sparse column or row arrays (`csc_array`, `csr_array`) for matrix arithmetic. The method can calculate the equilibrium paths of the smaller models such as the "bump" and "curl" practically instantly, while the models with more DOFs take around 5 seconds to complete.

A.13 Rigid origami optimization with more Miura-ori unit cells

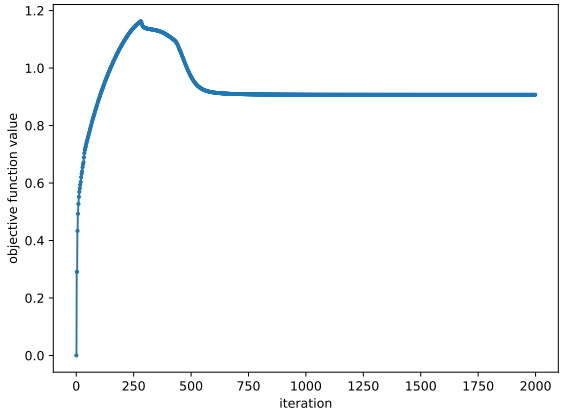
While this work's designs consist of a 4-by-4 grid of Miura-ori unit cells, the rigid origami optimization code is capable of producing much more detailed tessellations: figure 46 shows an origami design with a grid of 16-by-16 Miura-ori unit cells, along with the graphs of the objective function and constraints that summarize the optimization process, which is much more computationally expensive than with the 4-by-4 grid—it took 10 hours as opposed to 30 seconds.



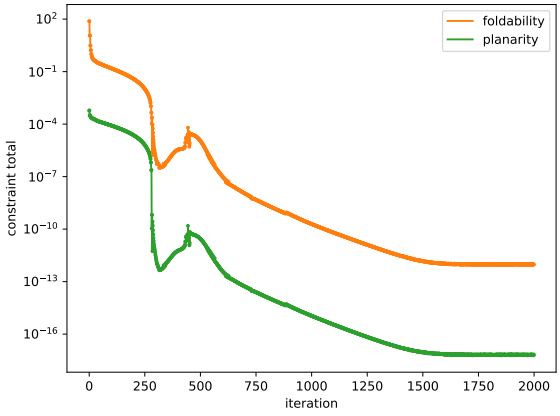
(a) Origami design that approximates a saddle



(b) Flattened origami design



(c) Objective function during the rigid optimization of the 16-by-16 Miura-ori unit cell design



(d) Sum of the constraints during the rigid optimization of the 16-by-16 Miura-ori unit cell design

Figure 46: Results of using the rigid origami optimization with a 16-by-16 Miura-ori unit cell design

A.14 Real objective function results in non-rigid origami shape matching

Looking at the graph for the smoothened objective function in figure 47, we see monotonically decreasing functions for all values of p. The smooth maximum distance, however, is not the function we are truly interested in; we are interested in the real maximum distance, which is plotted in figure 20.

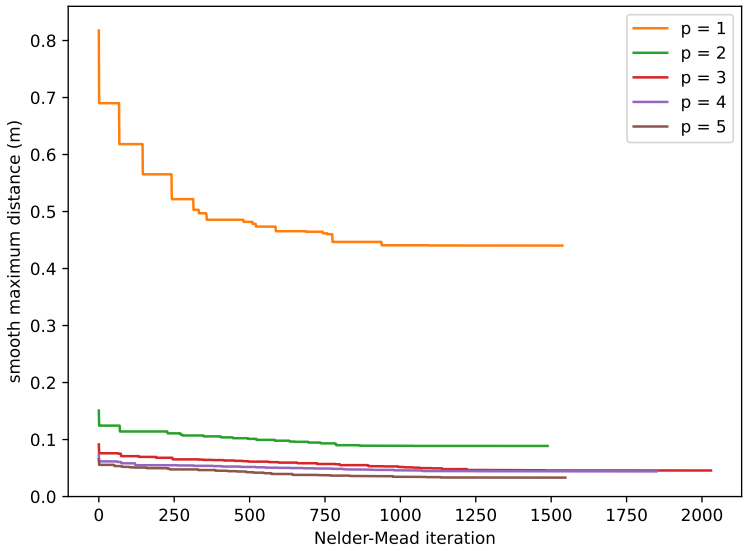


Figure 47: Smooth maximum distance function for different values of p

A.15 Sequential quadratic programming and relative design variables

In principle, the SQP optimization algorithm with BFGS updates from `scipy.optimize.minimize(method='BFGS')` calculates the expected optimal design by fitting a multi-dimensional quadratic function to the objective function and taking its minimum. In our case, this approximation to the minimum is already unphysical after the first iteration, see figure 48. The changes in the design are too large, and the algorithm did not allow to put bounds on the design variables.

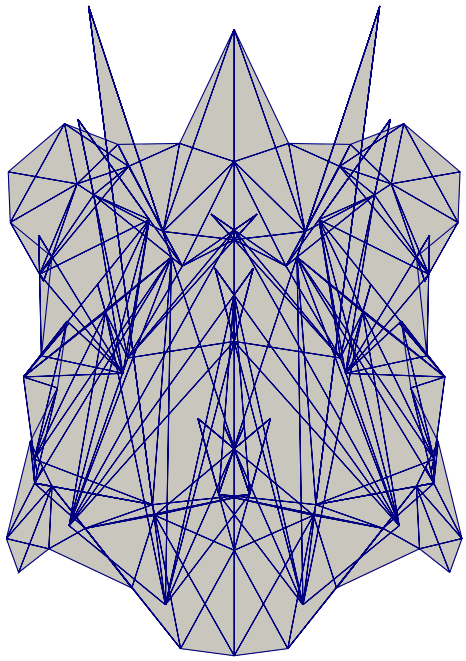
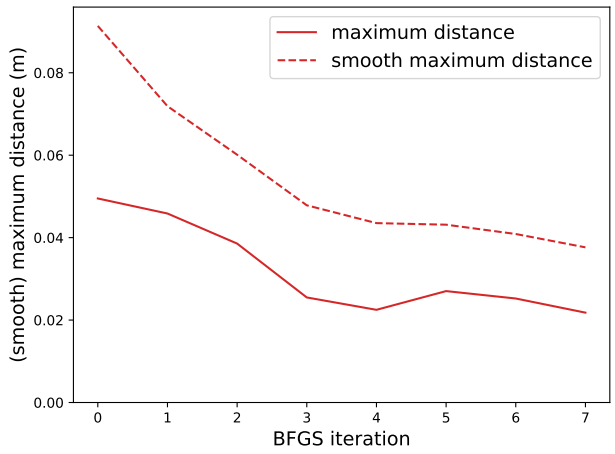


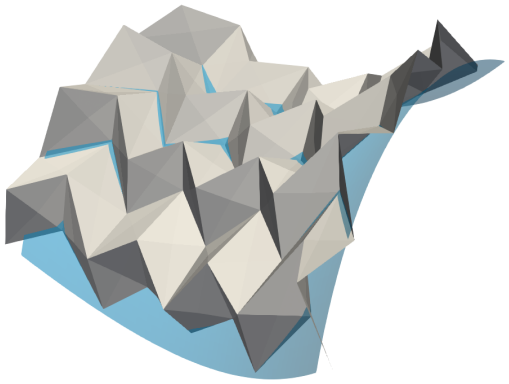
Figure 48: The design after one iteration of SQP with BFGS updates

We explored a different way of defining the design variables to see if it would impact the performance of the SQP optimizer. The new design variables use the relative coordinates, instead of the absolute coordinates—they define the deviation of the coordinate from the initial design. Figure 49 shows the results for the saddle case.

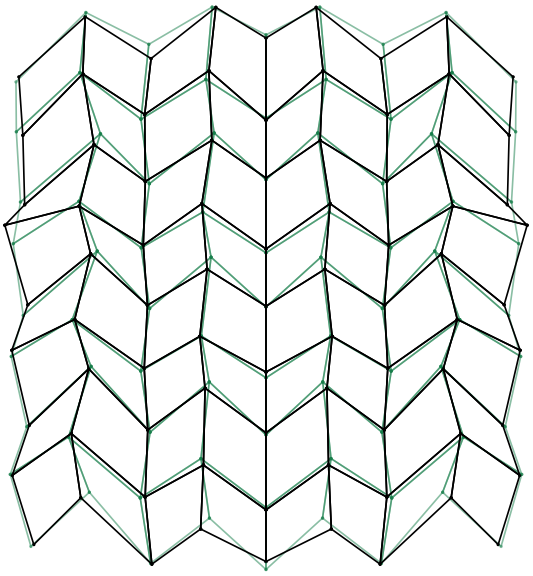
The maximum distance is 0.0218 m, which is lower than the result we obtained with the Nelder-Mead optimizer, which was 0.0243 m. The final design is similar to the previously obtained design, but the computational effort is around double—it took 32 hours, whereas the Nelder-Mead process took 16 hours.



(a) (Smooth) maximum distance over the optimization iterations



(b) Folded saddle on the target surface



(c) Flat origami design; initial design in green, final design in black

Figure 49: Non-rigid shape matching results for the saddle, using the SQP optimizer with BFGS updates