Development of the new origami designing algorithm, the Angle-Ray method

Through the approach of experimental mathematics

Toma K. Sydnes

Contents

Abstract

In recent years origami has gained significant interest among engineers applying the ancient art of paper folding to aerospace, robotics, biomedical devices, and more. As the number of applications has increased and areas widened, the need for a thorough understanding of the mathematics of origami has become crucial for developing new applications. One application that documents the mathematical properties of origami well is the Circle-River method, an origami designing algorithm known to be the most successful. This and other algorithms are mathematically complex and not easily accessible without an extensive mathematical background. To combat this accessibility issue and out of personal curiosity I decided to develop a new origami designing algorithm, the Angle-Ray method, which uses trigonometry and a circle property to develop new origami structures according to user specifications. This report will present the experimental approach behind the development of the Angle-Ray method.

Glossary

In this report, some origami-specific terminology will be used. Whenever uncertain, please use the glossary below explaining the terms.

Closed-ended flaps – flaps that share both nodes with at least another flap.

Closed-ended flap section – a section of closed-ended flaps separating two collections of open-ended flaps.

Closed-ended point – a node where the open-ended flap and the closed-ended flap intersect.

Connectivity – a measure of how well two flaps are connected. The greater the connectivity the lower the mobility between the flaps and the less surface area between them, and vice versa.

Crease line – representation of each fold in a crease pattern and is a simple straight line.

Crease pattern – the pattern made of crease lines that emerge when unfolding an origami structure and the pattern can be used to fold the origami structure.

Efficiency – how much of the paper is being used in comparison to the total. The more paper within the crease pattern the less is wasted and the more efficient it becomes.

Flap – a part of a paper that is partially disconnected from the rest with a degree of free movement. More theoretically a flap is a mountain fold with valley folds on both sides.

Mountain fold – a fold that folds outwards in a crest and is represented with a blue crease line on crease patterns (Figure 1).

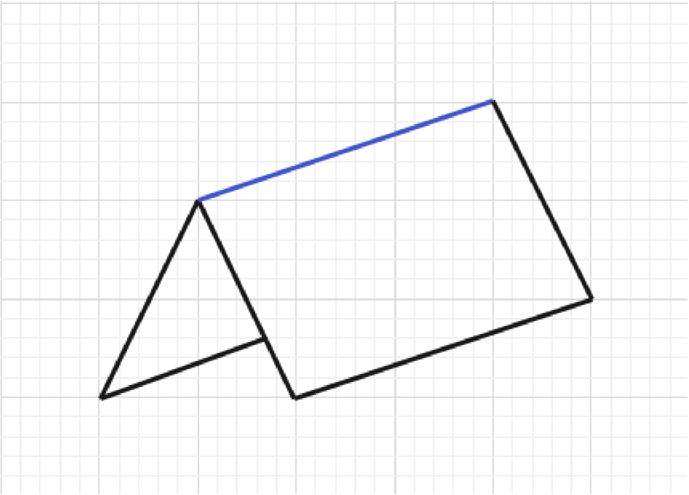


Figure 1. Mountain fold

Open-ended flaps – flaps that are connected to another flap only on one end.

Open-ended point – the outer edge of an open-ended flap.

Origami base – the simplest structure used to create a desired final product with the required number of flaps and connectivities.

Shaping – the process of adding details to an origami base to make it resemble the desired structure.

Shaping folds – types of folds used in the shaping process to add details.

Tree diagram – a diagram simplifying the final desired structure into its essential required flaps both closed-ended and open-ended. In the case of humans, it would be a common stick figure (Figure 2).

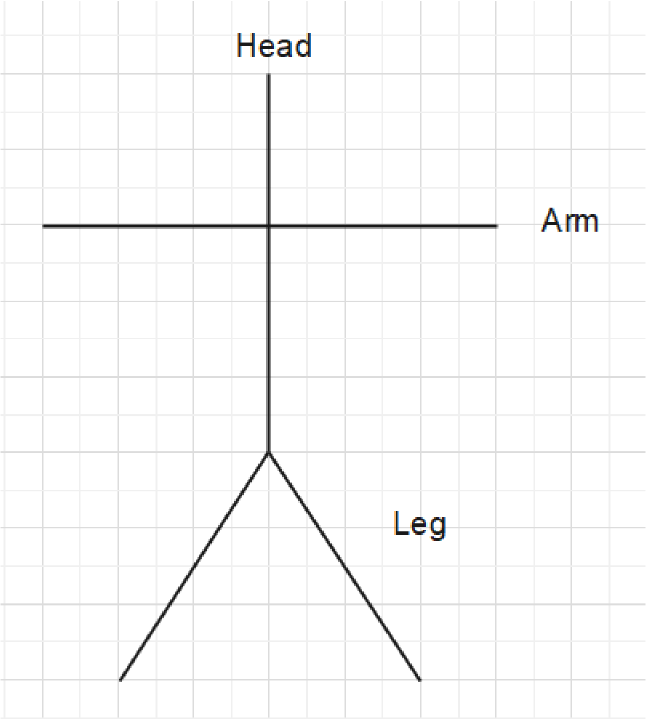


Figure 2. A tree diagram of a human

Total flap length – the length from an open-ended point to the center point of the crease pattern.

Valley fold – a fold which folds inwards and is represented by a red crease line on crease patterns (Figure 3).

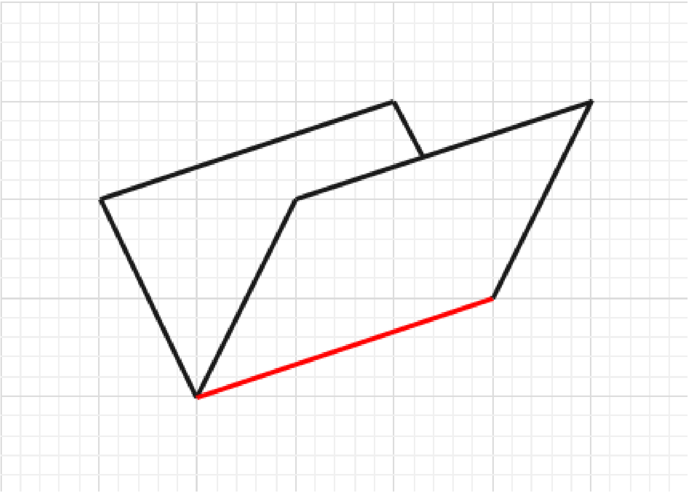


Figure 3. Valley fold

Introduction

Origami (折り紙), the Japanese art of folding (折り) paper (紙) has a history spanning at least six centuries (Robinson, N/A). It is an art form where you fold structures from uncut paper either as decorative elements or depicting various shapes such as birds, and people. In more recent years origami designing has seen major leaps after mathematicians found interest in the art, discovering mathematical properties that aid artists when making new designs. One of the greatest leaps is the use of mathematics to design desired origami bases. An origami base is a simple structure with the number of desired flaps, types, lengths, and connectivities, that can be shaped out to the structure by folding in details. Therefore, many traditional and modern origami designs start with an origami base. However, developing the desired origami base is a tedious non-artistic process of trial and error, especially if you are inexperienced. To aid this process of developing an origami base, mathematicians have made several algorithms. The most successful of them is the circle-river method (Lang, 2009) accomplished through complex mathematics requiring a high degree of mathematical knowledge. This complexity restricts the method's accessibility, and I felt a desire to develop a method using simpler mathematical concepts that would be familiar to a 16-year-old in the Middle Year Program (MYP) taught by the International Baccalaureate (IB). This would give a wider audience the ability to design their own origami. Additionally, I gained a renewed interest in origami when discovering origami applications in real life. I was fascinated when I discovered origami’s application to the design of the James Webb Space Telescope to fold and unfold the primary mirror (NASA, 2022) and its applications in robotics and medical devices (Muller, 2019). Researching origami’s potential applications is still ongoing as origami's ability to reshape three-dimensional structures and transfer mechanical energy is highly valuable within the field of engineering.

Origami basics

When developing a new origami design there are three distinct processes. Firstly, the artist starts by drawing a simple tree diagram noting desired lengths and connectivities. Secondly, the artist uses this information at the first step (number of flaps, lengths, types, positions, and connectivities) to generate an origami base with these requirements. Thirdly the artist adds details using shaping folds to make the origami base resemble the desired structure. In brief, the first step is planning, the second step is making an origami base, and the third step is shaping. My project is focused on developing a new algorithm to design origami bases based on user specifications, as the process of making an origami base is tedious, non-artistic, and challenging compared to the other two steps.

Approach to the Problem

The process behind the development of my algorithm can be reduced to a repetitive cycle of observations, hypothesis, experimentation, analysis, and conclusion. At any time, I made observations through sketching, folding, mind simulations, and computer programs, seeking relationships and features that could be manipulated. Based on these observations I developed hypotheses, at times small and at times an entire method, which were experimentally tested. The data from the tests came as folded shapes and were analyzed and compared to expectations. Therefore, most of the data will be presented as pictures in the appendices, but its meaning will be summarized.

When approaching a huge problem such as developing a new fully functional origami-designing algorithm there is a need to subdivide the problem into smaller more comprehendible parts, as seen below. Further, the problem was not just subdivided but allocated different degrees of importance (level 1 being the most important), which influenced the order I approached the problem.

Level 1

- Must allow input and output of open-ended flaps.
- Must allow input and output of several flaps.
- Must allow input and output of the length of flaps.

Level 2

- Must allow input and output of closed-ended flaps.
- Must allow input and output of any number of flaps.
- Must allow input and output of connectivity.

Level 3

- Must allow any paper shape.

- Must have an optimization capacity.
- Potentially a computer program accepting variable input and giving the crease pattern as an output.

Development process

Developing a new origami designing algorithm was a long endeavor starting as part of personal project at the IB's Middle Year Program in the summer of 2022 (Sydnes, 2022), but the research was not continuous. The development process is subdivided into four stages as each stage of development has a fundamentally different goal and approach, where each stage has its own product.

As a note whenever a crease pattern is drawn, folded, and measured there will be minor errors due to measurement tools and folding inaccuracy which are inevitable. The ruler used can only measure to the nearest mm and the protractor used can only measure to the closest whole degree.

The first stage

The start of my endeavor was the personal project, a research project done at the end of the MYP. As part of my personal project, I had set a goal to develop an origami designing algorithm that could be understood by 16-year-old MYP students (Sydnes, 2022). Hence, I restricted myself to mathematical concepts taught in MYP and I made a booklet with concise instructions on how to apply the method. This initial restriction was also followed for the final method of this report. When approaching the problem of developing a new origami designing algorithm at this stage I decided to simplify and focus only on the most important elements of origami (level 1), which means that this stage only investigated open-ended flaps.

Initial observation

Before I started to develop an algorithm, I had to find patterns and features of origami that could be manipulated. Hence, I folded and unfolded several traditional origami structures observing their crease patterns (Appendix 1). My observations can be summarized in the three points below.

1. Most of the time valley folds and mountain folds alternated.

   a. This is called Maekawa’s theorem and was discovered by Maekawa to be the criteria for flat foldability, the ability of one origami crease pattern to be folded flat (Wikipedia, 2022).
   b. This observation also allowed me to understand the minimum theoretical criteria of a flap, to be a mountain fold with two neighboring valley folds.
2. Conditions for axes of symmetry for n number of flaps.

   If $\frac{n}{4} = \mathbb{Z}$ there are two diagonal and two perpendicular axes of symmetry intersecting at the center of the paper.

   If $\frac{n}{2} = \mathbb{Z}$ and $\frac{n}{4} \neq \mathbb{Z}$ there are two perpendicular axes of symmetry intersecting at the center of the paper.

3. All shapes within the crease patterns were right-angle triangles.
   a. I later found the statement to be false however this was a crucial misunderstanding, which simplified the problem of designing origami bases.

Experiment 1

Based on my observations I saw that all traditional structures were built on right angle triangles inspiring me to investigate how I could use trigonometry.

Hypothesis: Trigonometry can be used to outline the edges of a new origami structure.

This approach was never properly developed, but to test it I developed a rough method using the concept.

Method: Using trigonometry one could find the angle necessary to create a line of desired side length which could outline the crease pattern as a polygon. The crease pattern could then be found by drawing lines from the center of the paper to the edges of the polygon. Each side length would then be a measure of connectivity and the number of flaps could be adjusted through the choice of polygon.

Data: Images from the tests and calculations can be found in Appendix 2 (tests for five flaps and six flaps with various length inputs).

Results and Analysis: Due to the varied results (Appendix 2) from this method it is difficult to make a clear judgment on the hypothesis, however, I leaned towards rejecting the idea. It was observed that some errors were caused due to rounding, which exacerbates as the number of flaps increase due to the recursive nature of the method. Additionally, there were challenges getting the ending point and starting point to overlap which requires very specific inputs. This could have been overcome using the internal angles of the polygons to adjust.

Conclusion: I decided not to continue the pursuit of this method. Firstly, it required too many calculations and immensely precise measurements. Secondly, due to its recursive nature which added rounding errors. Thirdly, there was no clear path toward the application of the method for more complex structures.

Experiment 2

Following Experiment 1's negative results I went back to my observations looking for any patterns I could manipulate.

Hypothesis: Observed patterns can be used to create rules that can be followed to simplify the process of making an origami base.

Algorithm developed:

1. Decide the number of flaps desired (n)
2. The following steps will vary depending on the number of desired flaps (n)
    a. If $\frac{n}{4} = \mathbb{Z}$ there are 4 axes of symmetry. Two horizontal and two perpendicular axes intersect at the center of the paper.
    b. If $\frac{n}{2} = \mathbb{Z}$ and $\frac{n}{4} \neq \mathbb{Z}$ there are two perpendicular axes of symmetry intersecting at the center of the paper.
    c. If $\frac{n}{2} \neq \mathbb{Z}$ subtract n by 1 and run it through step 2. The flap subtracted will be added between two flaps later.
3. Plot the mountain folds so that they follow the axes of symmetry. For an odd number of flaps add the last one between any two mountain folds. Then draw valley folds cutting the angle between the mountain folds in two to make them a flap.

Data: The images of the crease patterns and bases produced can be found in Appendix 3. In the test, both even and odd values were tested for all values between 4 and 17 flaps.

Result and Analysis: The algorithm simplified the process of designing new origami bases supporting the hypothesis. However, the algorithm only allows for the user to input desired number of flaps, hence very limited in the user's freedom and control.

Conclusion: The algorithm can create a crease pattern with the desired number of flaps supporting the hypothesis. However, a user can only alter the number of flaps. Hence the algorithm is minimal and acts more as a summary of the initial observations that can be built upon in the future.

Experiment 3

Instead of following through Experiment 2’s algorithm which had little capacity to be altered, I returned to the idea of using trigonometry. But instead of using trigonometry to find the outer edges I tried a new approach.

Hypothesis: Trigonometry can be used to get flaps of desired length from the center of the paper when compared to a perpendicular line.

Method: Using a perpendicular line from the edge to the center of the paper users can use Equation 1 to find the angle between the desired flap and the perpendicular line required for the desired flap length. Using these angles one can draw the crease pattern using a protractor. These lines would then become mountain folds and in between every mountain fold one must draw a valley fold bisecting the angle. Other than the angles the flaps can be placed anywhere meaning that for any length there are either eight options or four.

$$cos^{-1}\left(\frac{l}{2d}\right) = \theta$$

d = desired flap length, *l* = side length of the paper.

Equation 1. First equation

Equation 1 explained: The length of the perpendicular line going to the center of the paper will be half of the paper's side length when the paper is a perfect square. The rest is ordinary trigonometry, a concept taught within the MYP following my criteria.

Data: Images from the testing process can be found in Appendix 4.

Result and Analysis: The tests came out positive where the expected number of flaps was generated, the crease pattern was foldable, and came with the desired flap lengths with a 1:1 ratio meeting the requirement for level 1. The method was effective and easy to use and became part of the final algorithm at this stage. When testing there were problems in cases where the crease lines and the edge of the paper created a non-triangular polygon (Figure 4a) as this shape was not foldable without creating an undesired flap. This was resolved by adding a mountain fold between the open-ended points allowing the excess part to be folded away creating a triangle (Figure 4b). This observation and solution were also applied in stages three and four.

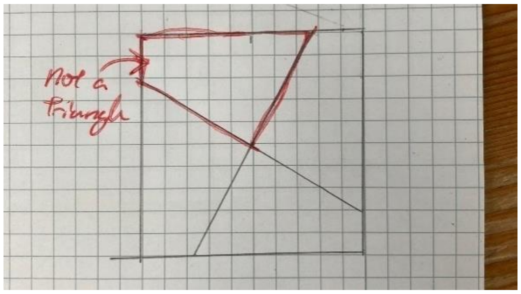


Figure 4a. Non-triangular polygon within crease pattern

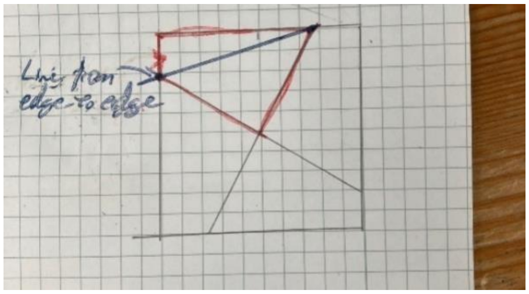


Figure 4b. Figure 4a resolved with an additional mountain fold

Conclusion: The method was successful and easy to apply. Certain minor errors can be expected due to measurement errors and rounding errors for the angles, but it is more than effective for its intended purpose as a simple tool to design origami bases. However, the experiment and theory show that there are limitations to the method. Listed below, the most significant is the limitation on the number of flaps with the same length and the limitation on the maximum and minimum flap length.

- The method does not work if the paper is not a square.
- The method does not work when the length of a flap is less than half of the length of the side.

- The method does not work when the length of a flap is greater than $\sqrt{2 * half\ of\ side\ length^2}$.
- The method does not work when the number of flaps with equal lengths exceeds 8.
- The method does not work when the number of flaps with the maximum length exceeds 4.
- The method does not work when the number of flaps with the minimum length exceeds 4.
- The method cannot create closed-ended flaps.
- All crease lines must merge at the center of the square paper.

Product of the initial stage

Based mainly on Experiments 2 and 3 I developed a final algorithm for this stage. This was made into a booklet as a product of my MYP personal project found in Appendix 5.

Testing the booklet

To ensure the booklet was functional I tested the guide by starting with an intended structure and evaluating the product's resemblance.

Hypothesis: The booklet sufficiently guides an individual in creating an origami design of a desired structure.

Method: I tested the booklet by starting with the intended designs of flower, bird, fish, airplane, lion-face, elephant-face, clamshell (Tridacna), and octopus, and evaluated the outcome.

Data: The data can be found in Appendix 6 where the intended structure is in captions below the image with a personal note of success or failure.

Result and analysis: Most of the origami structures resemble the intended structure. However, some fail due to the limitations of the method. The giraffe failed due to the minimum length and maximum length restrictions, the shrimp failed due to a lack of control over connectivity, and the octopus failed due to difficulty implementing closed-ended flaps.

Conclusion: The method is highly effective for relatively simple designs such as lion-face and birds and meets the requirement for level 1. However, it is difficult to use for structures requiring large length ratios, a high number of flaps, and closed-ended flaps which are a level 2 requirements.

Second stage

After completing the initial stage, I left the project for a while but regained interest in improving the method as I came up with an idea that could potentially allow closed-ended flaps. The idea was to draw circles where the radius would equal the desired flap length and find the intersection between the circles which would become the nodes of the crease pattern. However, the approach produced crease patterns that were difficult to understand and challenging to fold. The structure was difficult as there were no effective ways to ensure the geometry within the crease patterns were triangular. Therefore, I decided to give up on this approach after two months of intensive experimentation. In Appendix 7 you can find some of the experimental sketches made.

Third stage

When developing the method of the second stage I discovered an interesting crease pattern (Figure 5). The crease pattern when folded created two distinct and simple open-ended flaps attached to a single closed-ended flap with three sets of potential variables a, b, and c (Figure 5). Where a influences the length of the closed-ended flap, b influences the length of the open-ended flap, and c influences the connectivity. At an early stage, I confirmed that the input a, b, and c lengths were the same as the output lengths when folded.

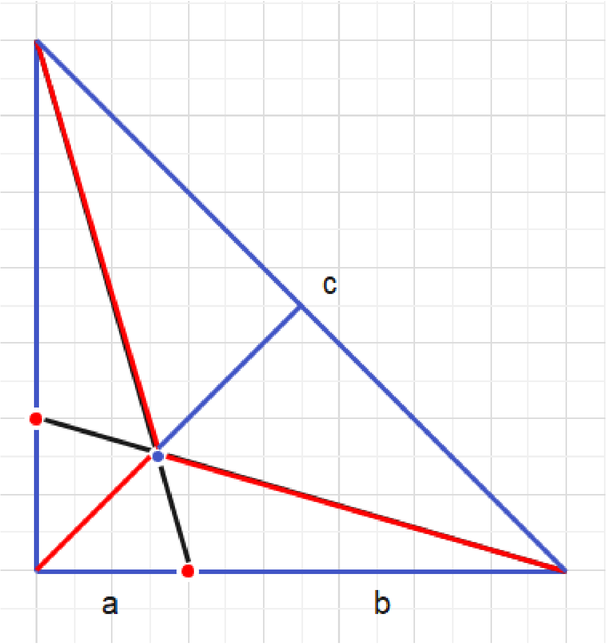


Figure 5. Core crease pattern of the Angle-Ray method

What makes this crease pattern special is how it creates closed-ended flaps. It creates equally as many closed-ended flaps as open-ended flaps but folds them together into one at the folding stage. This allowed the crease pattern to accommodate closed-ended flaps without drastically increasing the complexity, achieving parts of level 2 requirements. However, this approach reduces efficiency as it produces duplicates of the same flap. When weighing the pros and cons it is evidently worth pursuing as the approach is simple and integrates closed-ended flaps and connectivity.

Based on the traits of the crease pattern I developed the start of the new method. When creating a new origami design, you often start by creating a tree diagram of the desired structure (Figure 6, left). Considering the folding approach produces closed-ended flaps by folding together duplicates there is a need to alter the tree diagram. The idea is that one can combine the open-ended flap length and the closed-ended flap length and represent the total flap length as an open-ended flap. The total length will be found by pinpointing a center point in the crease from where you build the altered tree diagram (Figure 6, right).

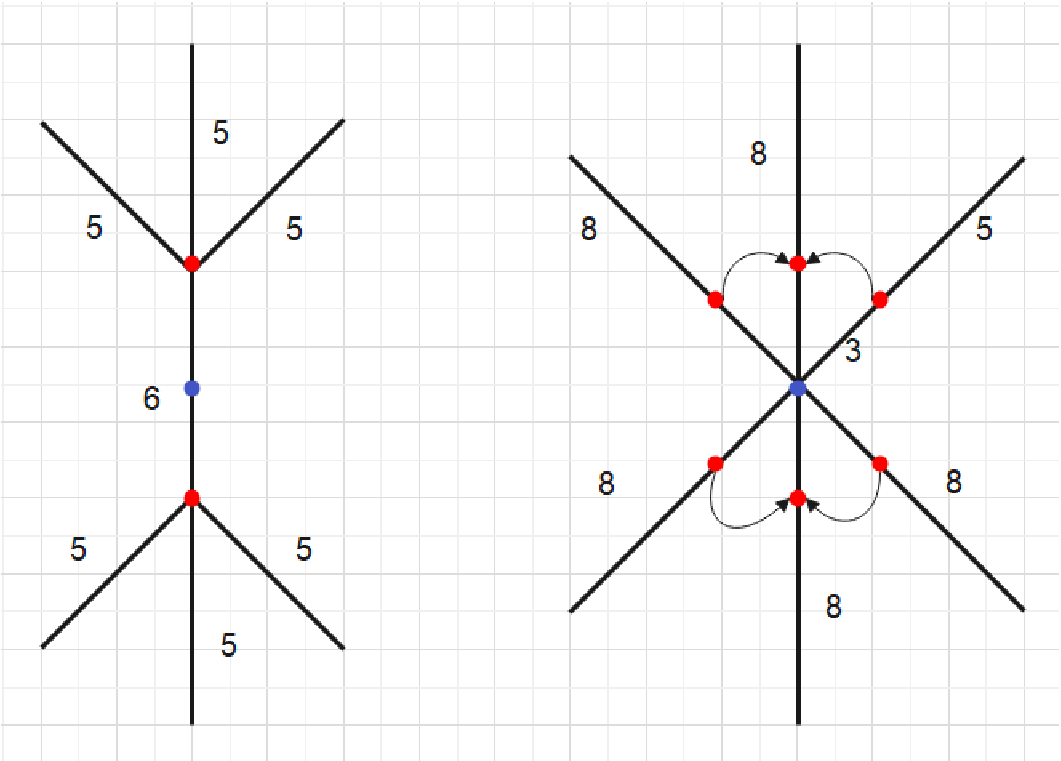


Figure 6. Left: original tree diagram. Right: altered tree diagram

When looking at this newly altered tree diagram (Figure 6, right) I saw strong similarities to the crease pattern produced by the method developed at stage 1. Hence, I decided to opt for a similar approach finding the angle between crease lines using trigonometry and the total flap length. In contrast to the method of stage 1, I used the cosine rule which removed the length and number limitation of the flaps, as it removed reliance on a perpendicular reference line and the need for the crease lines to intersect the edge of the paper as it could be folded away. Therefore, the following equation was developed.

$$\theta = Cos^{-1}\left(\frac{{f_{t1}}^{2} + {f_{t2}}^{2} - l^{2}}{2f_{t1}f_{t2}}\right)$$

$$f_{t1}, f_{t2} \neq 0$$

f_{t1} and f_{t2} is the total flap length drawn as a straight crease line form the center point.

l is the distance between the open-ended points, to an extent a measure of connectivity.

Equation 2. Initial cosine rule based equation

However, Equation 2 had a major problem. Connectivity is a measure that should be ratio-based. This means that if the length ratios between f_{t1} and f_{t2} are constant and the angle is constant then the connectivity should be equal, but the current approach of representing connectivity through the distance between the open-ended points is not constant. Therefore, I defined connectivity below (Equation 3).

$$c = \frac{f_{o1} + f_{o2}}{l}$$

$$l \neq 0$$

c is the connectivity value between the two open-ended flaps.

f_{o1}, f_{o2} is the length of the two neighboring open-ended flaps.

Equation 3. Definition of connectivity

The definition of connectivity can then be rewritten to find l (Equation 4).

$$l = \frac{f_{o1} + f_{o2}}{c}$$

$$c \neq 0$$

Equation 4. l in terms of open-ended flaps and connectivity

From Equation 4 one can understand that when applied to the core crease pattern (Figure 7) the $c \leq 1$ have a 360-degree mobility around the node, on the other hand when $c > 1$ flaps have a decreased mobility. In addition, the lower the value for c is the more surface area there is between the two open-ended flaps, which can be manipulated when shaping.

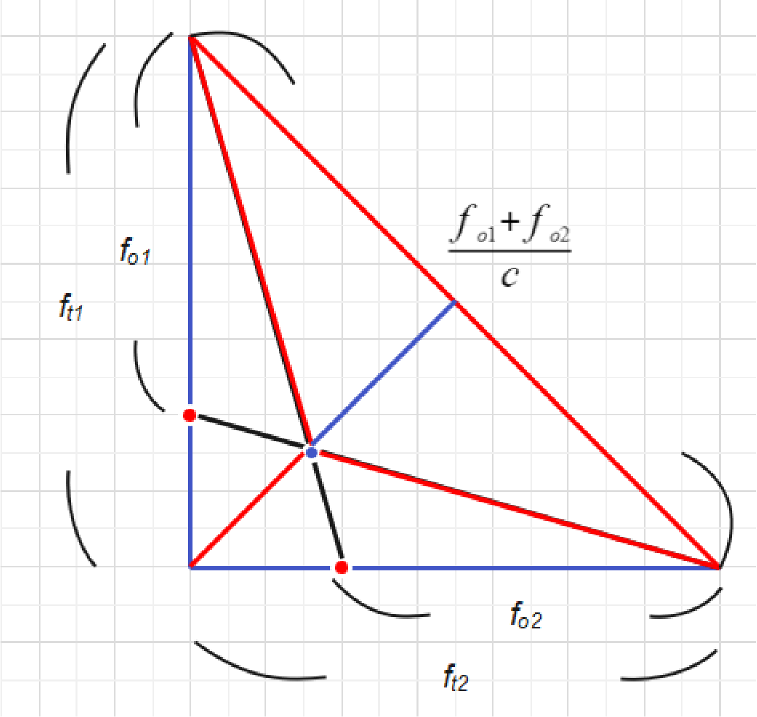


Figure 7. Variables in context of the core crease pattern (Figure 5)

Based on Equation 4 one can rewrite Equation 2. This is the main equation used in the final algorithm, the Angle-Ray method (Equation 5).

$$\theta = Cos^{-1}\left(\frac{{f_{t1}}^2c^2 + {f_{t2}}^2c^2 - {f_{o1}}^2 - {f_{o2}}^2 - 2f_{o1}f_{o2}}{2f_{t1}f_{t2}c^2}\right)$$

$$f_{t1}, f_{t2}, c \neq 0$$

Equation 5. The main equation of the Angle-Ray method

Equation 5 was tested to evaluate its useability and two challenges were observed. Firstly, the connectivity variable is difficult to use due to its definition being inversely proportional to the length between the open-ended points. Secondly the equation is large and takes time to input on a calculator, but this was overcome by developing a simple code to execute Equation 5 by inputting variable values (Appendix 8).

Product of the third stage:

The algorithm that was produced at the third stage is what became the core of the Angle-Ray method.

1. Sketch a tree diagram of the desired structure and decide on all the desired flap lengths and connectivity values (Figure 8).

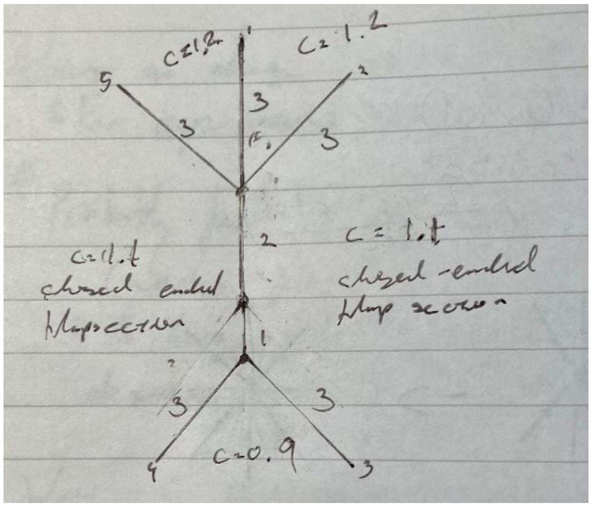
   

   Figure 8. Tree diagram for a 5 flap origami base

2. Then alter the tree diagram to only have one intersection being the center point. This is by deciding a center point from where you would add the length of the closed-ended flap and open-ended flap to gain the total flap length which is drawn as a straight line (Figure 9).

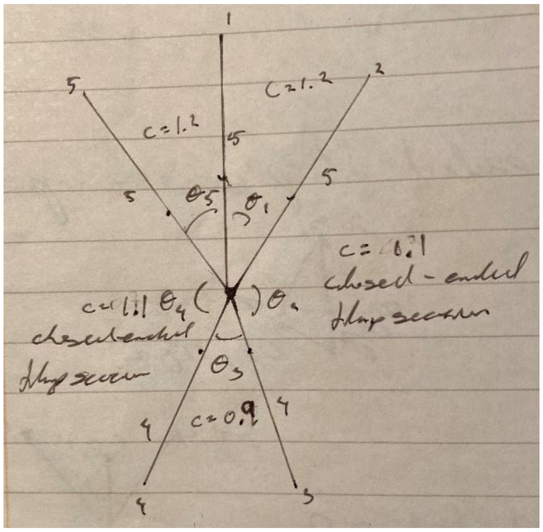


Figure 9. Altered tree diagram of the 5 flap origami base

3. Using the total flap lengths, desired connectivity values, and open-ended flap lengths, to find the angle between the crease lines using Equation 5.

$$\theta = Cos^{-1}\left(\frac{f_{t1}{}^{2}c^{2} + f_{t2}{}^{2}c^{2} - f_{o1}{}^{2} - f_{o1}{}^{2} - 2f_{o1}f_{o2}}{2f_{t1}f_{t2}c^{2}}\right)$$

4. Draw the main mountain folds with their appropriate total length and the angle found at Step 3. The mountain folds are what become the flaps.
   a. If $\sum\theta > 360$, then the values inputted are impossible to fold.
   b. If $\sum\theta < 360$, draw an additional mountain fold with the same length as the first mountain fold drawn, and draw a valley fold between the additional mountain fold and the first mountain fold. In the folding process, the additional flap will be folded together with the first flap removing the excess angle (Figure 10).
   c. If $\sum\theta = 360$, it is of the most optimal design for this method.

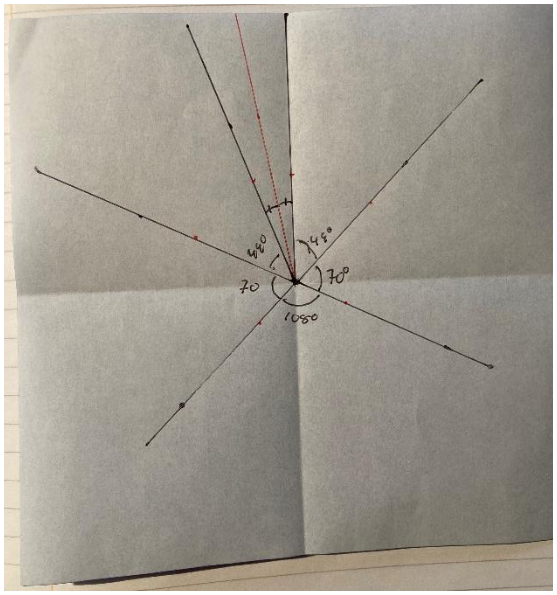


Figure 10. After completion of Step 4 for a 5 flap origami base with a sum theta less than 360

5. Following Step 4 draw additional valley folds between closed-ended points and neighboring open-ended points, and additional valley folds from the center point to the intersection of the two valleys. But not for closed-ended flap sections which will instead have a valley fold bisecting the angle. Also, draw mountain folds between the open-ended points and the rest of the crease lines will naturally follow when folding the flaps together (Figure 11).

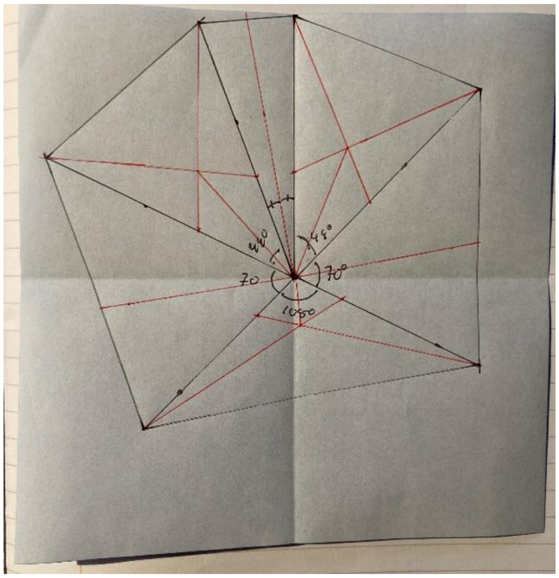


Figure 11, after completion of Step 5 for a 5 flap origami base with two closed-ended flap sections

The method was highly successful at creating closed-ended flaps and open-ended flaps of desired lengths and connectivities, however testing this method was a challenge even with the calculator that ran Equation 5 (Appendix 8) as values had to be input through trial and error for it to be foldable. Additionally, there were challenges drawing the crease pattern precisely with ordinary rulers and protractors. Therefore, for stage four, I decided to develop a computer program that could draw the crease patterns and automatically adjust input for foldability.

Fourth stage

Following the problems of the third stage, I wrote a computer program that could model the crease patterns and make necessary calculations based on user inputs. A detailed explanation of the final computer program is found under *Final computer program explanation*. Here I will present the two biggest problems that arose when coding.

The first problem was difficulty getting a foldable crease pattern commented on in the third stage. A crease pattern output is not foldable when mountain folds cross one another (Figures 12). This occurs when $\sum \theta > 360$ or $\sum \theta < 180$.

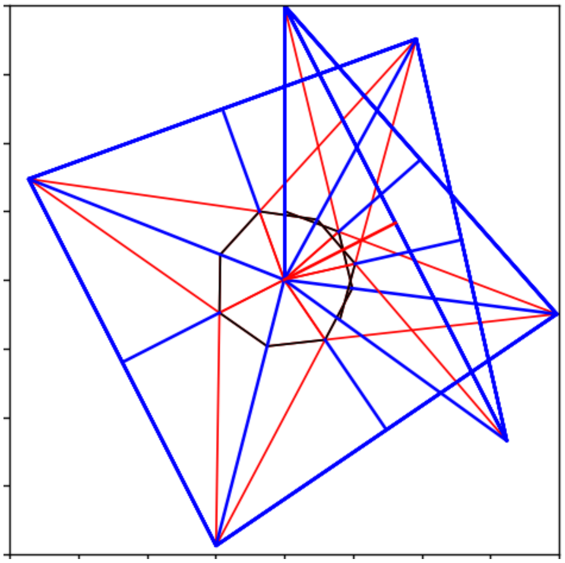


Figure 12a. $\sum \theta > 360$

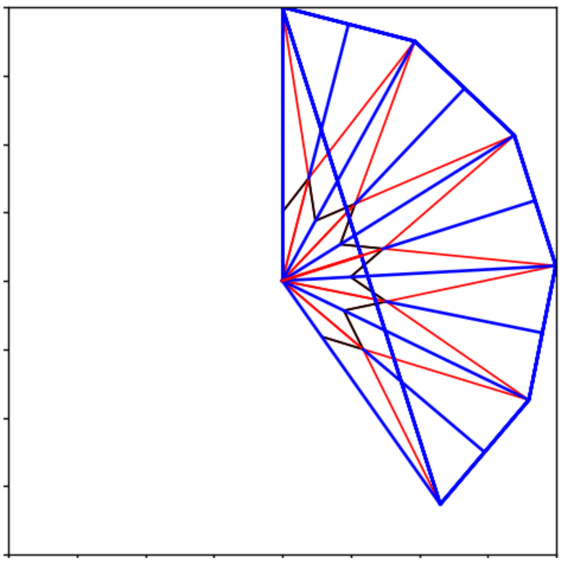


Figure 12b. $\sum \theta < 180$

To solve this issue, I developed Equation 6 where θ is the angles found when using Equation 5.

$$\varphi = \theta * \frac{360}{\sum \theta}$$

θ is the angles found using Equation 5.

φ is the new recalibrated angle.

Equation 6. Equation for angle recalibration

It uses the property of a circle having 360 degrees to find the rate of recalibration required for every angle to get $\sum \theta = 360$. This recalibration makes non-foldable designs (Figure 13a) foldable (Figure 13b) and optimizes any designs that folded away portions of the paper increasing crease pattern efficiency. However, this recalibration alters the connectivity between flaps. When θ increases, the connectivity between flaps would decrease and vice versa. Generally, when designing origami, a decrease in connectivity is not a problem as the connectivity can be increased later through shaping folds. On the other hand, an increase in connectivity is problematic as there are no shaping folds that I am aware of that can decrease connectivity. But the connectivity is only increased when initial inputs give a crease pattern which has a $\sum \theta > 360$, not foldable and any increase in connectivity is better than an un-foldable design.

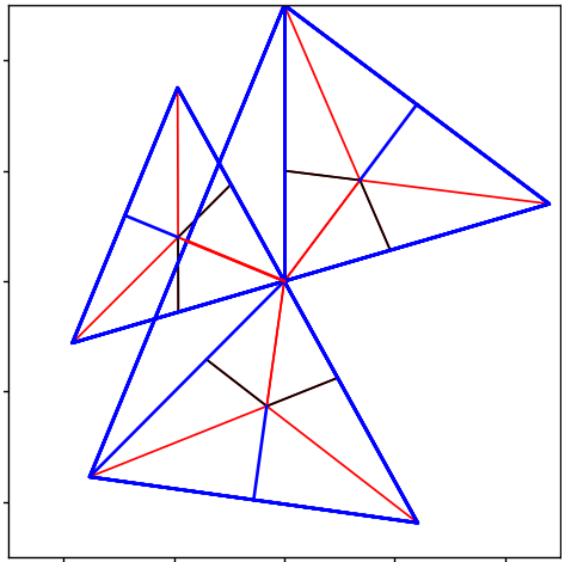


Figure 13a. Before optimization/fixing

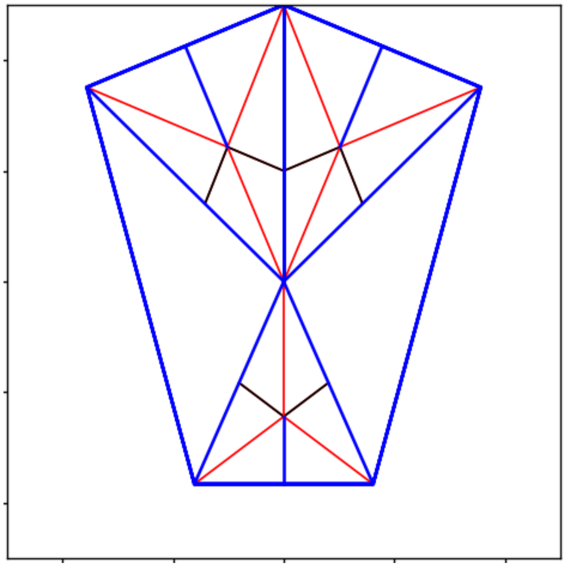


Figure 13b. After optimization/fixing

The second problem I encountered was a missing mountain fold between the flaps, this is required to allow the flaps to fold together. Hence, I investigated all angles produced by the crease lines (Figure 14) which showed equality between the angles represented by the same letter in Figure 15. Based on this observation I hypothesized that the relationship in Figure 15 was true. To test this hypothesis, I first measured the other crease patterns I had folded and got a result aligning with the hypothesis. Following I edited the code and gained crease pattern outputs containing these crease lines (Appendix 9). When folding the crease patterns all of them were successful at being folded together with neighboring flaps supporting the hypothesis. Same results were also achieved in the Final experiment (Appendix 15).

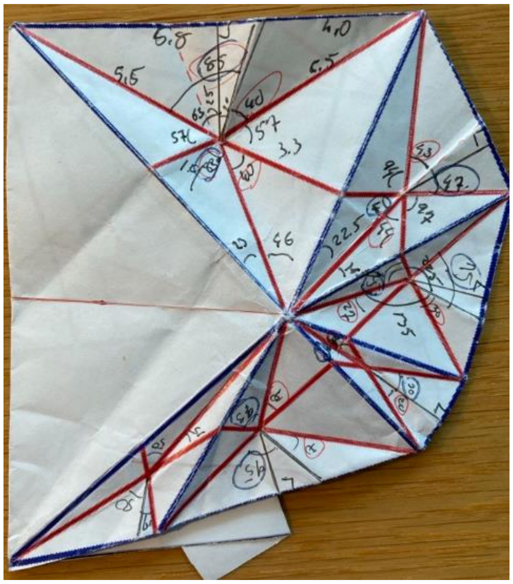


Figure 14. Investigating crease pattern angles

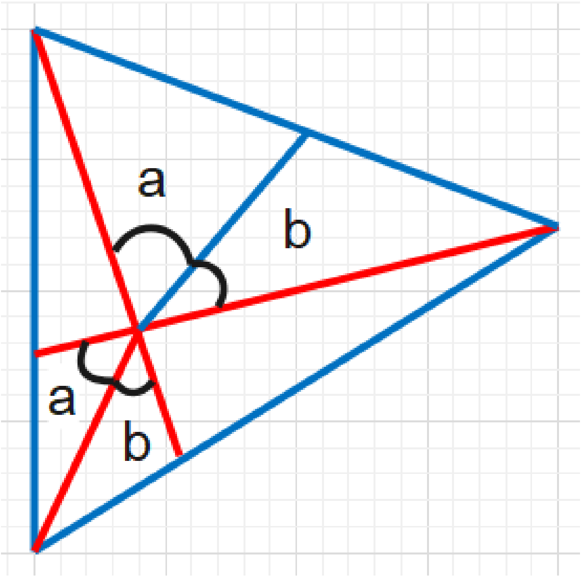


Figure 15. Equality of angles on crease patterns

This is where my development process ends, therefore the final algorithm of this stage is also the final algorithm of the entire project. The written algorithm can be found under *Angle-Ray method*, and the computer program can be found in Appendix 10 which is accompanied by a user guide (Appendix 11) and an explanation of the code (Appendix 12).

Angle-Ray method

1. Sketch a tree diagram and decide on the lengths of all flaps and the connectivity between the flaps (Figure 8).

2. Pick a point on the tree to be the center point from where you alter the initial tree diagrams to have straight lines that only intersect at the center. Then get the total lengths from the center point to each open-ended point which becomes the total flap length (Figure 9).
3. Then input all the values into the equation below (Equation 5).

$$\theta = Cos^{-1}\left(\frac{{f_{t1}}^{2}c^{2} + {f_{t2}}^{2}c^{2} - {f_{o1}}^{2} - {f_{o1}}^{2} - 2f_{o1}f_{o2}}{2f_{t1}f_{t2}c^{2}}\right)$$

$$f_{t1}, f_{t2}, c \neq 0$$

$$f_o \leq f_t$$

θ is the angle between f_{t1} and f_{t2}.

f_{t1} and f_{t2} are the total flap lengths of two neighboring flaps.

f_{o1} *and* f_{o2} are the open-ended flap lengths at the end of f_{t1} and f_{t2}.

c is the connectivity between f_{o1} *and* f_{o2}.

   a) If $c < 1$ there is no change in mobility, however, there is a change in the surface area which can be used during the shaping process.
   b) If $c > 1$ there will be an increase in connectivity as *c* increases.
   c) When dealing with closed-ended flap sections $ft = fo$.

4. If $\sum \theta \neq 360$, run every θ value through the following equation (Equation 6), if not skip this step.

$$\varphi = \theta * \frac{360}{\sum \theta}$$

$$\sum \theta \neq 0$$

θ are the angles found using Equation 5.

φ are the new recalibrated angles.

5. Draw mountain folds from a pre-decided center point with the total flap lengths and the angle found in Steps 3 or 4 between them.
6. Draw mountain folds connecting the neighboring open-ended points.
7. Mark out the closed-ended points and draw a valley fold between the closed-ended points and the open-ended points of neighboring flaps.

8. Then draw a valley fold from the center to the intersection of the valley folds drawn in Step 7.
9. Lastly draw a mountain fold from the intersection between the valleys drawn in Step 7 and the mountain fold drawn in Step 6 following Figure 15's equality of angles.

Reason for the name Angle-Ray method

The name Angle-Ray method is heavily inspired by the Circle-River method (Lang, 2009) which has a name that is highly descriptive of the method's approach to origami designing. Hence, I named my method the Angle-Ray method as the approach is to find the angle between lines that spread out from a central point.

Final experiment

Now that my final product (Appendix 10) was complete it was time to test the algorithm and computer program in its entirety. Testing took place in three parts. Firstly, with values I had selected, and secondly with randomly generated values within a fixed range. For both cases, the crease patterns were folded. The third test was done with random values, but the crease patterns were not folded and was done to investigate the source of error observed in the second test.

First test

In this experiment, I inputted values for logically reasonable origami bases. This means that the value inputs were mainly integers and connectivity values had little variation (Appendix 14). Additionally, closed-ended flap sections were strategically placed to allow ease when shaping. This was done both after and while writing the code therefore some crease patterns may be lacking certain crease lines. However, this does not affect the testing of the algorithm itself as it is independent of the drawing system. This test resulted in all output crease patterns being foldable, however, some were very difficult to fold and questionable on how easily they could be used as an origami base.

Second test

In this experiment, random values were inputted into the computer program using the code in Appendix 15 and the crease patterns (Appendix 16) were folded. Random values were used to avoid human biases when selecting variable values. I generated 10 random crease patterns with the value constraints stated below. Closed-ended flap sections were intended to be included in the randomization, but due to complications with the code was given up.

$$3 \leq total\ number\ of\ flaps < 10, total\ number\ of\ flaps \in \mathbb{Z}^+$$

$$1 \leq total\ flap\ length < 10, total\ flap\ length \in \mathbb{R}^+$$

$$1 \leq open\ ended\ flap\ length < total\ flap\ length, open\ ended\ flap\ length \in \mathbb{R}^+$$

$$0 \leq connectivity < 1, connectivity \in \mathbb{R}^+$$

From the crease patterns only Appendix 16.1, 16.6, 16.8, 16.10 was foldable. The other had one of two kinds of problems. The first kind is a case where you can connect the two neighboring open-ended points with a straight line (Figure 16, green line) without intersecting the flap in between (Figure 16, yellow circle). These structures are very difficult to fold and are not intended to be compatible with this method and computer program.

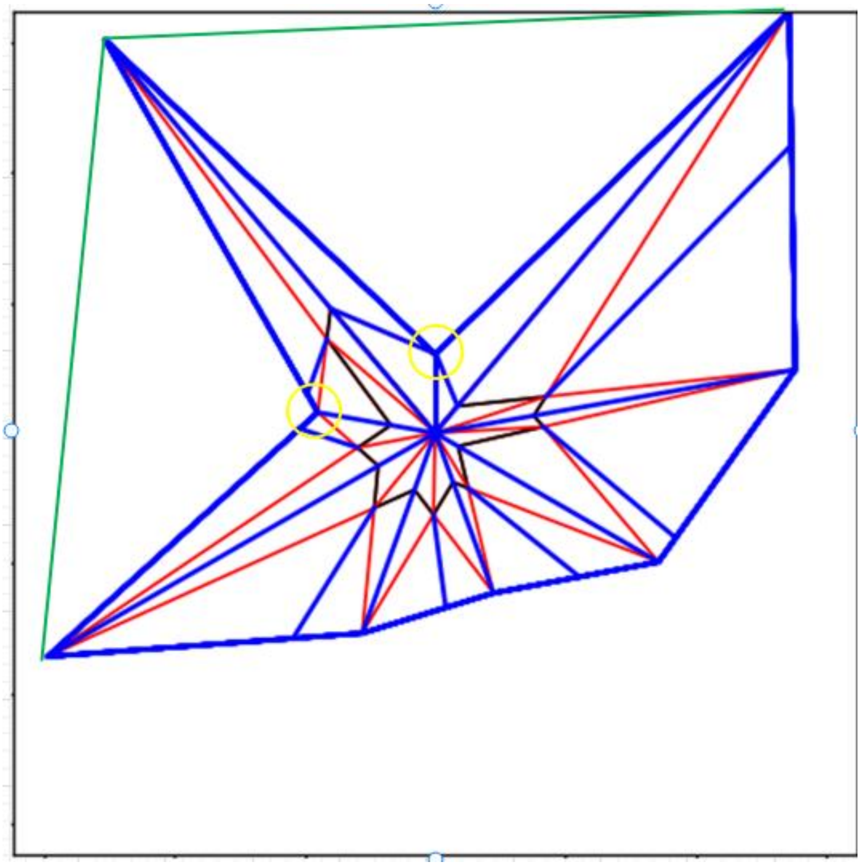


Figure 16. Problem with outputted crease pattern due to flap lengths

The second kind is a problem where the computer program draws two mountain folds without drawing a valley fold in between (Figure 17). When looking into the values used (Appendix 15) the problem is evident. The values generated had open-ended flap lengths which were too close to the total flap length resulting in an overlap of lines when the computer drew the crease pattern. Theoretically, if the crease pattern were printed on a larger paper with higher resolution the problem would be solved, but realistically speaking this kind of input provides no value.

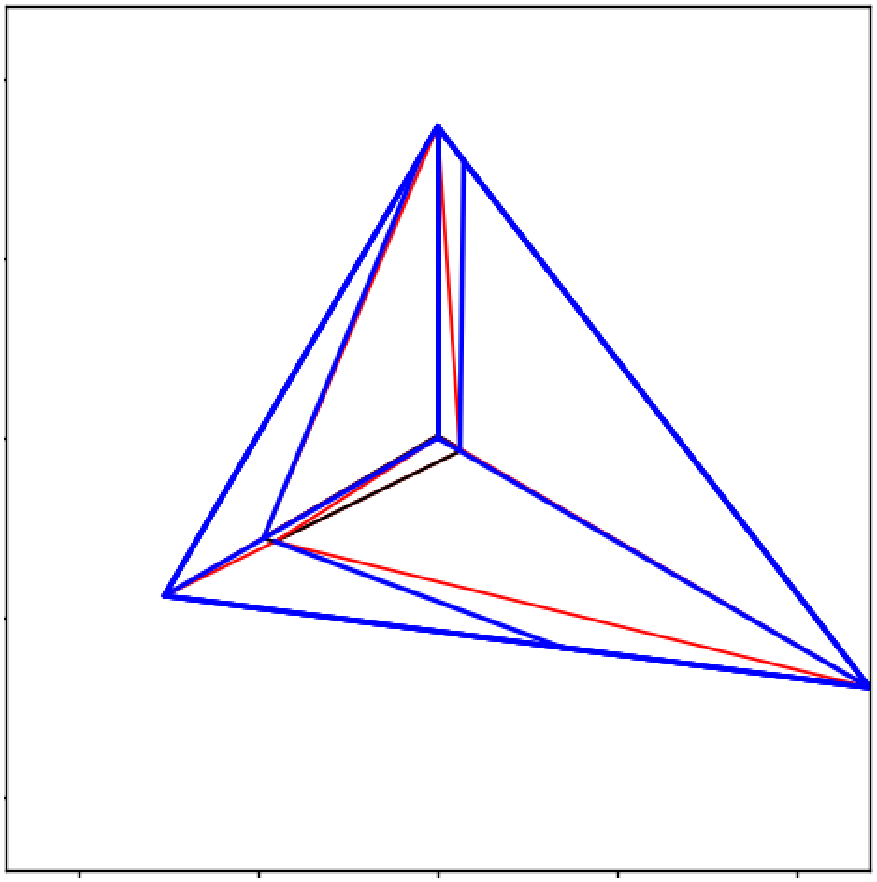


Figure 17. Problem with outputted crease pattern due to mountain folds without valley folds in between

The first problem occurs due to a lack of value limitation. Further investigation into this problem would allow for an effective way to warn users when such values are inputted. However, this will be implemented in a future update. The second problem is a problem due to the drawing program, which occurred because of the extreme similarities in values being inputted by randomization but are unrealistic inputs when considering practical applications. This was also the reason why I opted to test with random values as they would allow the discovery of unforeseen problems.

Additionally, when testing I observed several error outputs hinting at a need to find the error conditions done in the third test.

Third test

For the final test, I generated 500 sets of random variable inputs using the code in Appendix 15 and got the computer to record the types of errors given by the computer program and which lines gave the error. The value constraints were the same as for the second test except for an additional constraint limiting the number of decimal places to two.

$$3 \leq total\ number\ of\ flaps < 10, total\ number\ of\ flaps \in \mathbb{Z}^+$$

$$1 \leq total\ flap\ length < 10, total\ flap\ length \in \mathbb{R}^+$$

$$1 \leq open\ ended\ flap\ length < total\ flap\ length, open\ ended\ flap\ length \in \mathbb{R}^+$$

$$0 \leq connectivity < 1, connectivity \in \mathbb{R}^+$$

$$All\ values\ were\ limited\ to\ two\ decimal\ places$$

I decided to conduct this experiment as I observed several unexpected errors in the second test and suspected the source of the errors to be the code drawing the crease pattern rather than the algorithm itself. Hence, I hypothesized that the lines causing errors in the computer program were one of the lines modeling the crease pattern. The data can be found in Appendix 17.

Analysis and Evaluation

A summary of the data (Appendix 17) can be found in Table 1 showing a success rate of 41.6%. In Table 1 the percentage of each error can be found together with the line causing the error, showing that all errors occur in lines 162, 216, 224, 338, 391. All these lines are found within lines 139 to 246 and 315 to 419, which are the sections where the crease pattern is being modeled. Suggesting problems with the approach taken to model the crease patterns and a need to recode the computer program. However, it must not be understood that the goal is to eliminate all errors as certain inputs are not foldable and not intended for this method, and must be caught. At the same time, Table 1 shows no errors in the lines of code running the Angle-Ray algorithm itself, evidencing the algorithm's high functionality and reliability for any input values.

Error	Count	Percentage
NoError	208	41.60%
(<class 'ZeroDivisionError'>, 162)	154	30.80%
(<class 'ValueError'>, 216)	103	20.60%
(<class 'ZeroDivisionError'>, 224)	16	3.20%
(<class 'ZeroDivisionError'>, 216)	14	2.80%
(<class 'ZeroDivisionError'>, 338)	4	0.80%
(<class 'ZeroDivisionError'>, 391)	1	0.20%
Total	**500**	100.00%

Table 1. Summary of the data collected for the third test (Appendix 16)

Conclusion

Overall, this project was a success, developing a new origami designing algorithm called the Angle-Ray method which uses clever manipulation of trigonometry and a simple repeated folding pattern. The algorithm was successful at only using mathematical concepts taught in the middle year program, making origami designing more accessible. Additionally, the algorithm allows users to input desired flap lengths and connectivity with an option to optimize designs granting users a high degree of freedom and meeting level 3 criteria. However, the number of variables comes with the inconvenience of a large equation. To combat this the Angle-Ray computer program was developed to execute both the calculations and modeling of the crease patterns. The program was deemed well-functioning for reasonable origami designs but faces challenges when inputting random values causing errors in the modeling system. Fixing this issue requires a complete recoding of the program and a new modeling approach, a task for the future.

This report has opened the doors to a new viable alternative origami designing algorithm. But still lacks reliability and needs subtle modifications and improvements. Therefore, any future study on the Angle-Ray method should focus on scrutinizing and inspecting its limits and applicability to both origami designing and engineering.

Acknowledgment

Special thanks to Dr. Kim Damstra, a biology teacher at British International School of Stavanger, for his excellent guidance in report writing and for his supervision of my IB personal project, the starting point of this research project.

References

Lang Robert J., 2009. *Mathematical Methods in Origami Design*. [pdf] Available at: < http://m.archive.bridgesmathart.org/2009/bridges2009-11.pdf > [Accessed 18. June 2022]

Muller D., 2019. *Engineering with Origami.* [video online] Available at: <https://www.youtube.com/watch?v=ThwuT3_AG6w > [Accessed 15. May 2022]

Nasa, 2022. *Webb and Origami.* [online] Available at: <https://webb.nasa.gov/content/features/origami.html> [Accessed 10. December 2022]

Robinson N., N/A. *History of origami.* [online] Encyclopedia Britannica. Available at: <https://www.britannica.com/art/origami/History-of-origami> [Accessed 19. February 2024]

Sydnes T. K., 2022. *Personal Project – Creating Origami Designs*. [Unpublished report]

Wikipedia, 2022. *Maekawa's theorem*. [online] Available at: <https://en.wikipedia.org/wiki/Maekawa%27s_theorem#:~:text=Maekawa's%20theorem%20is%20a%20theorem,by%20two%20in%20either%20direction> [Accessed 4. September 2022]

Appendix

Appendix 1. Initial observations, traditional origami bases crease pattern

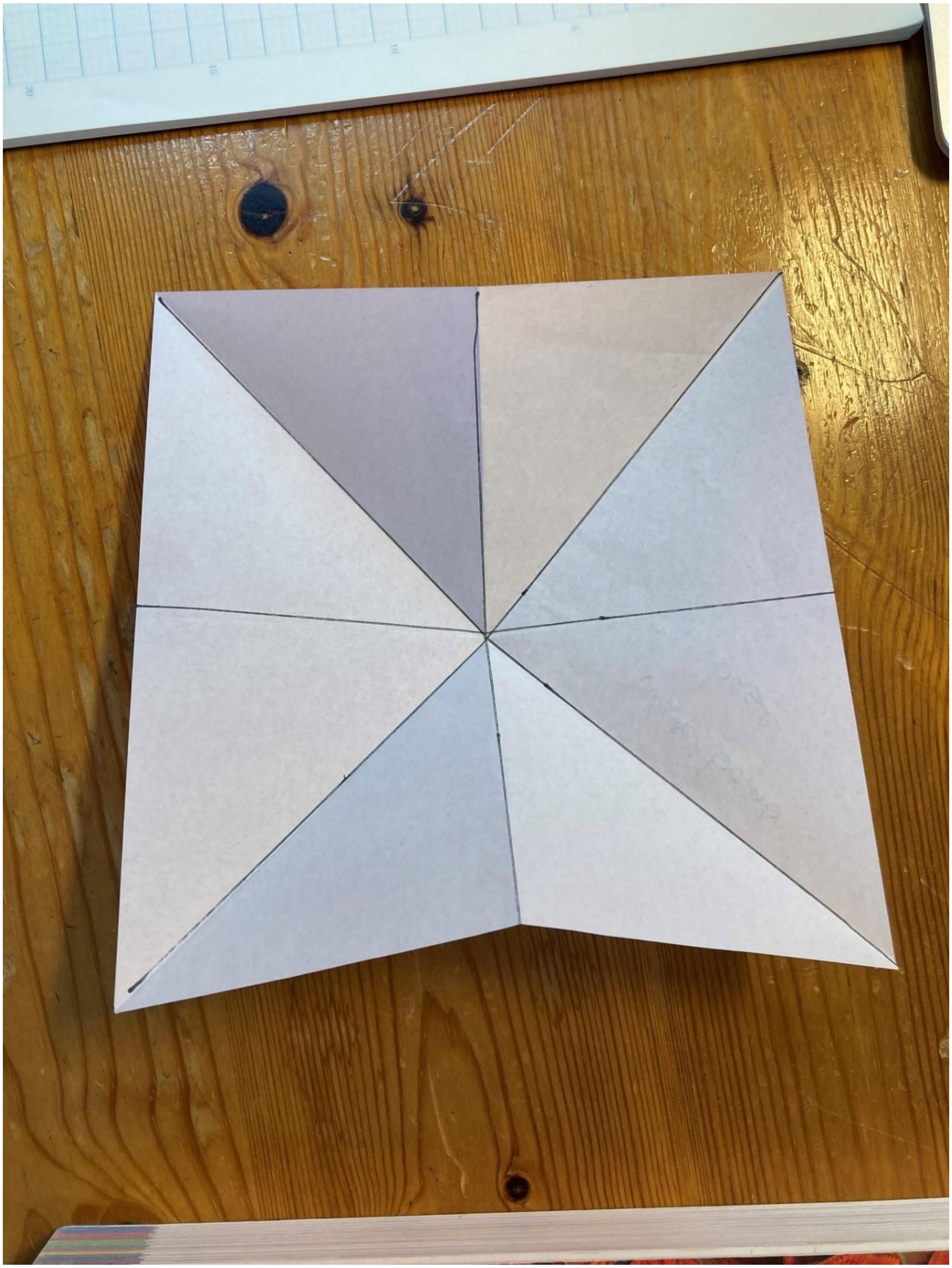


Figure 1813. Water bomb base

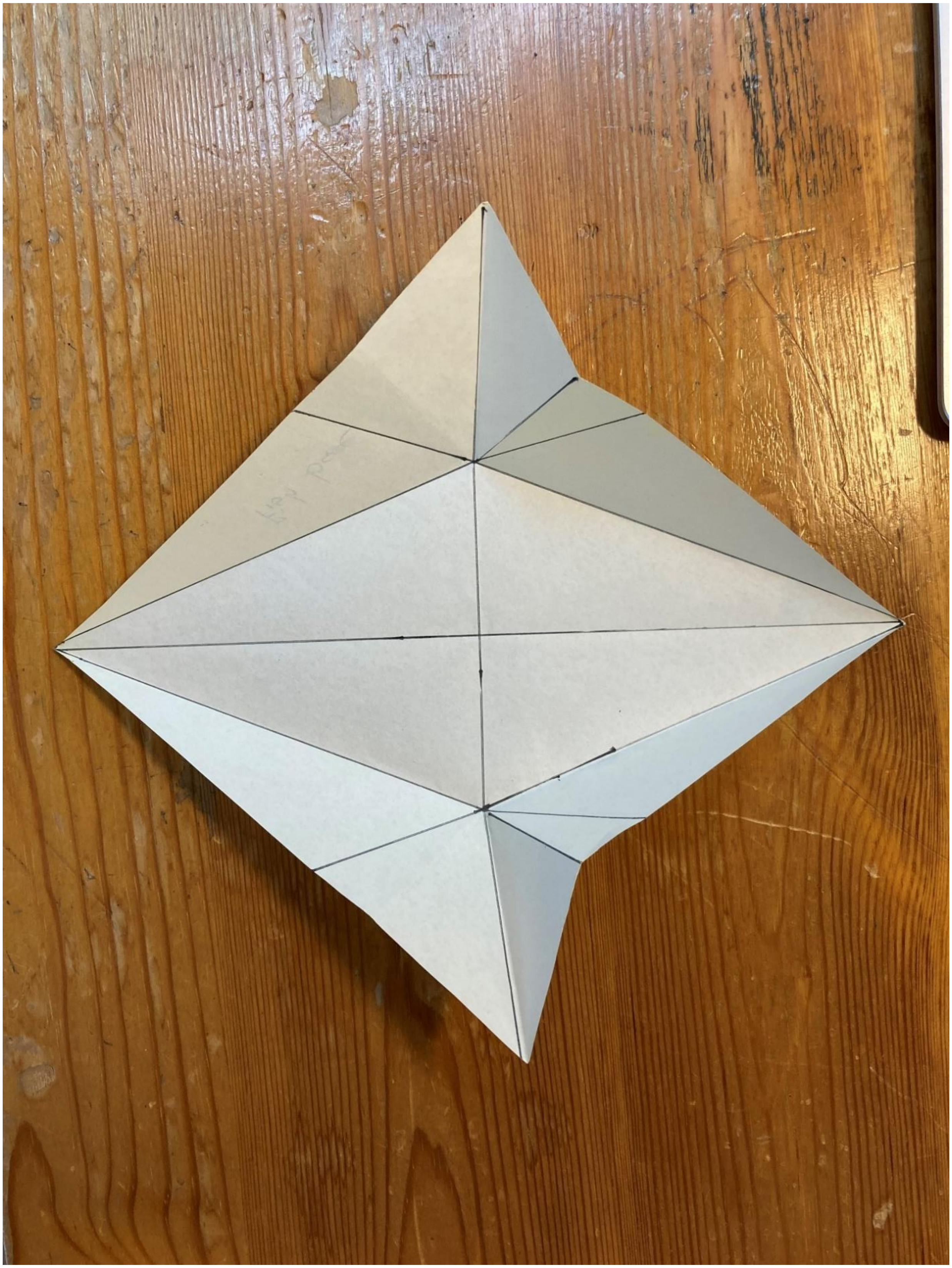


Figure 19. Fish base

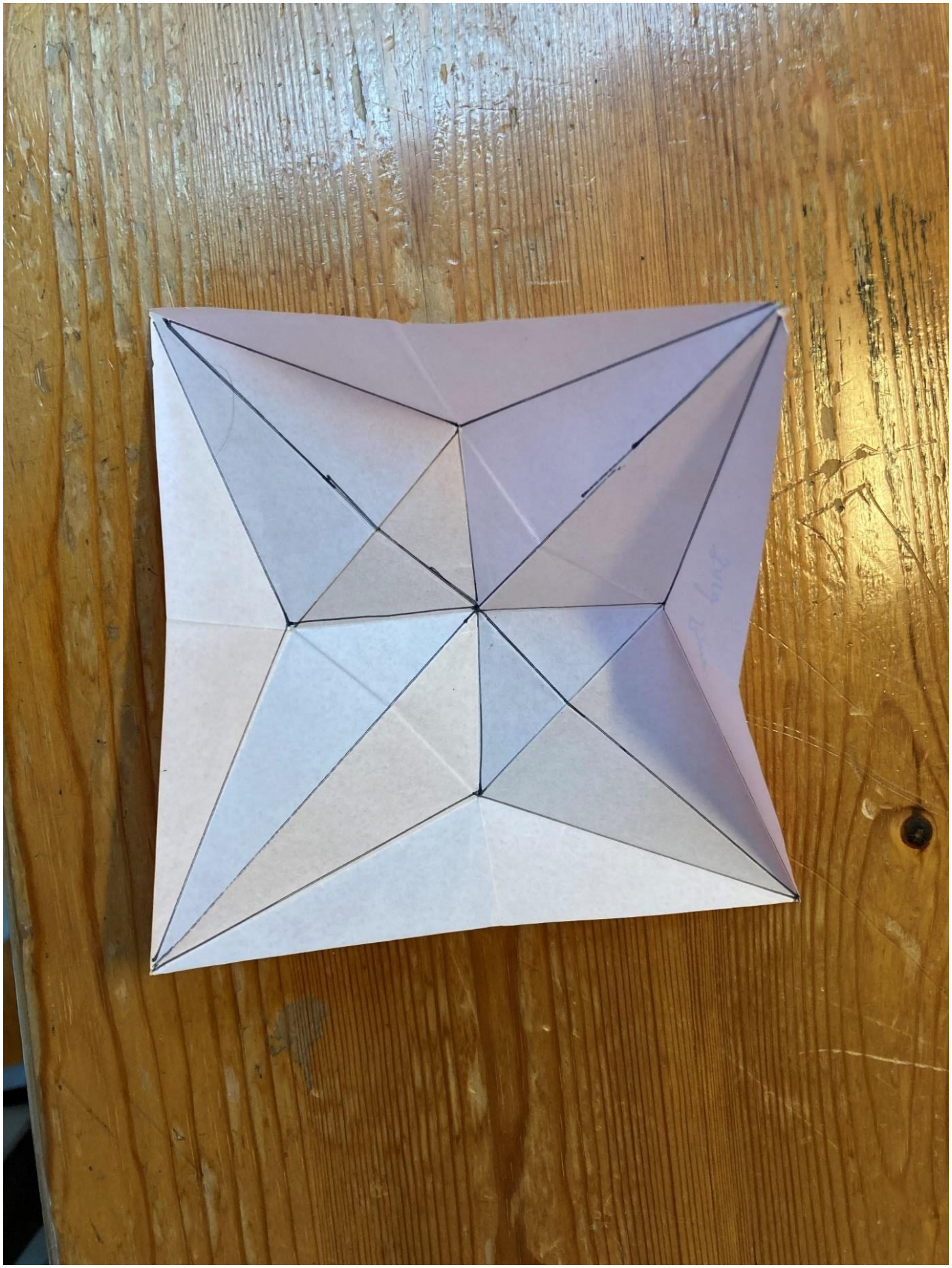


Figure 20. Bird base

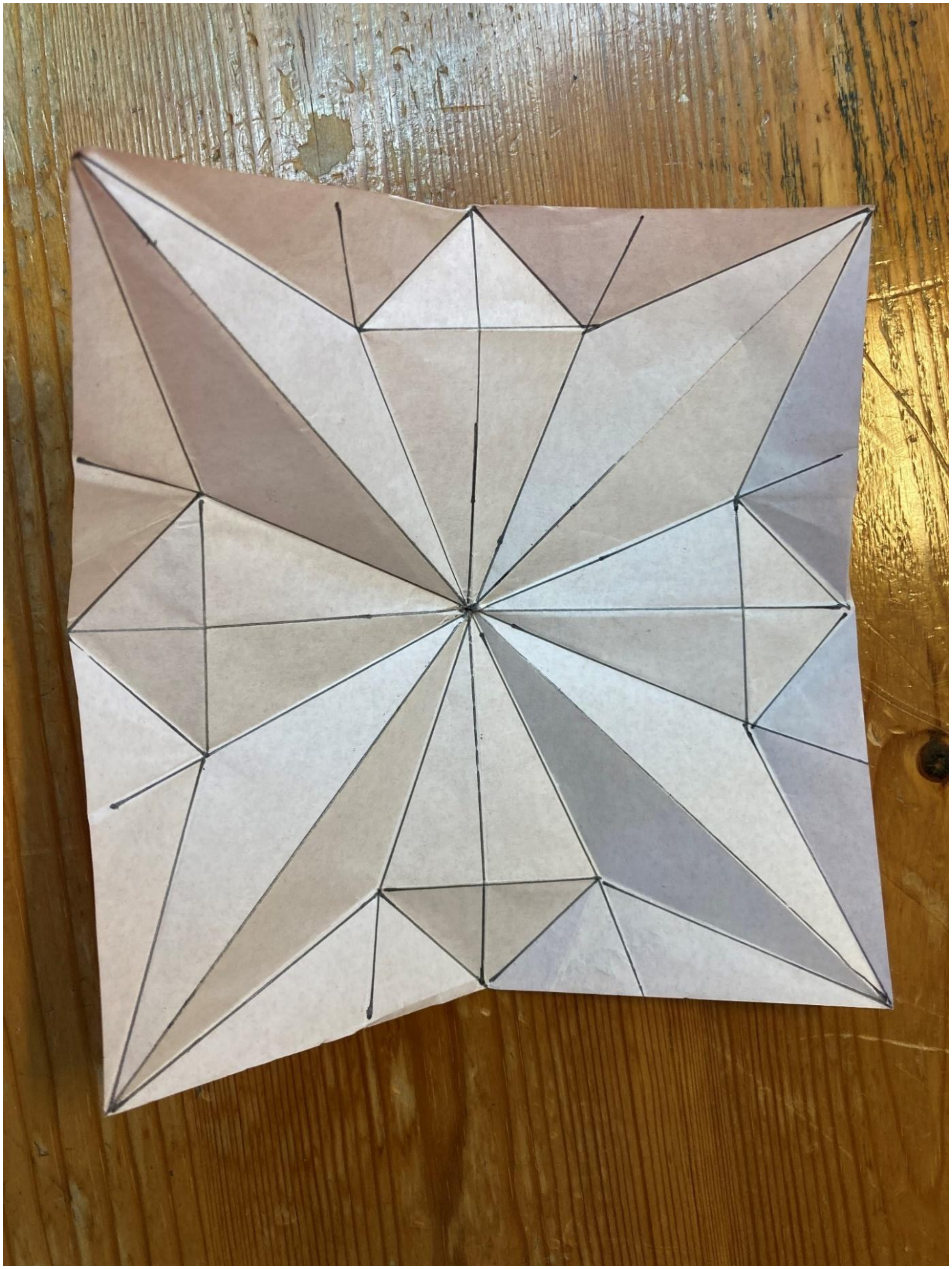


Figure 21. Frog base

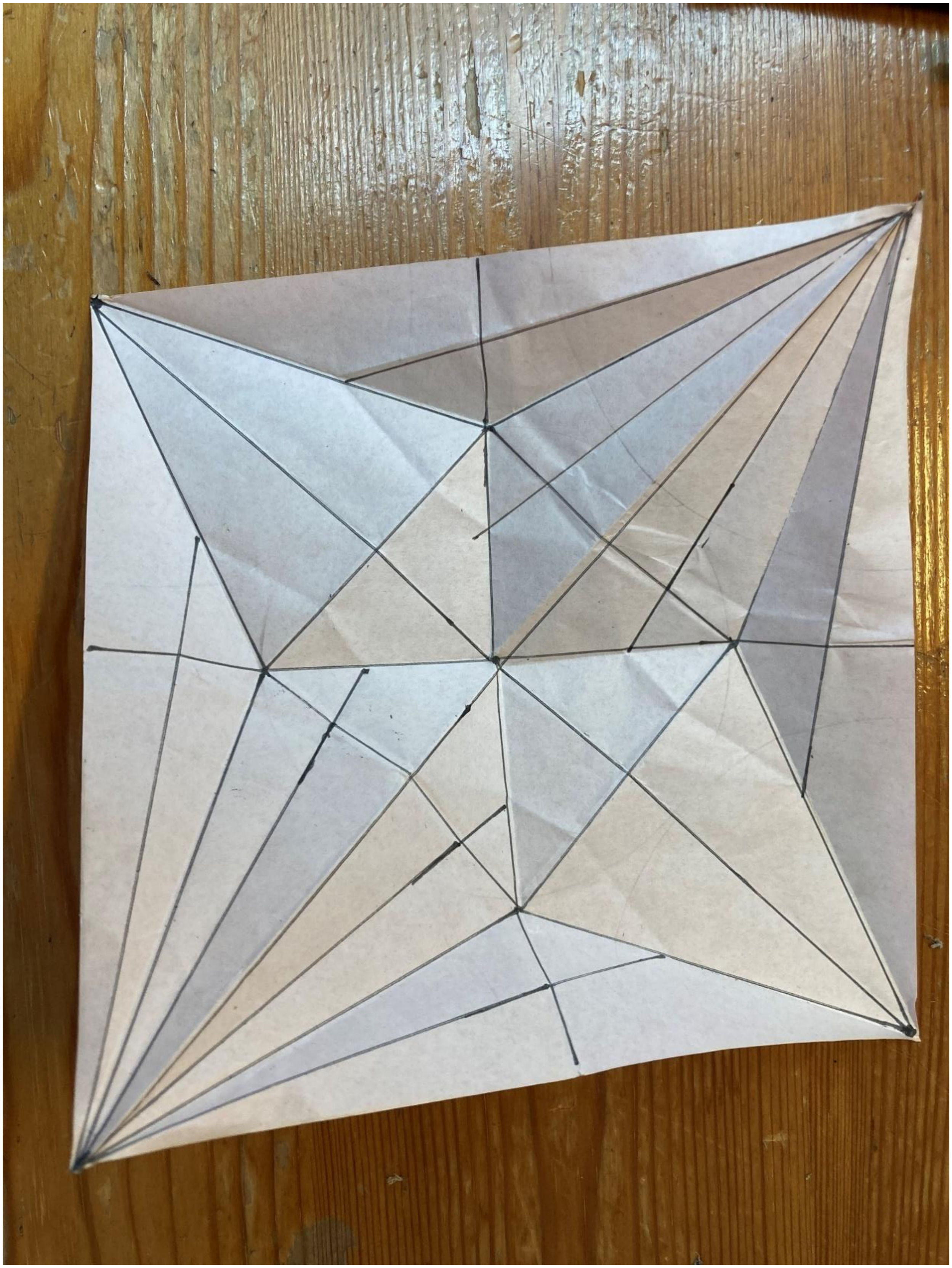


Figure 22. Bird base all crease lines

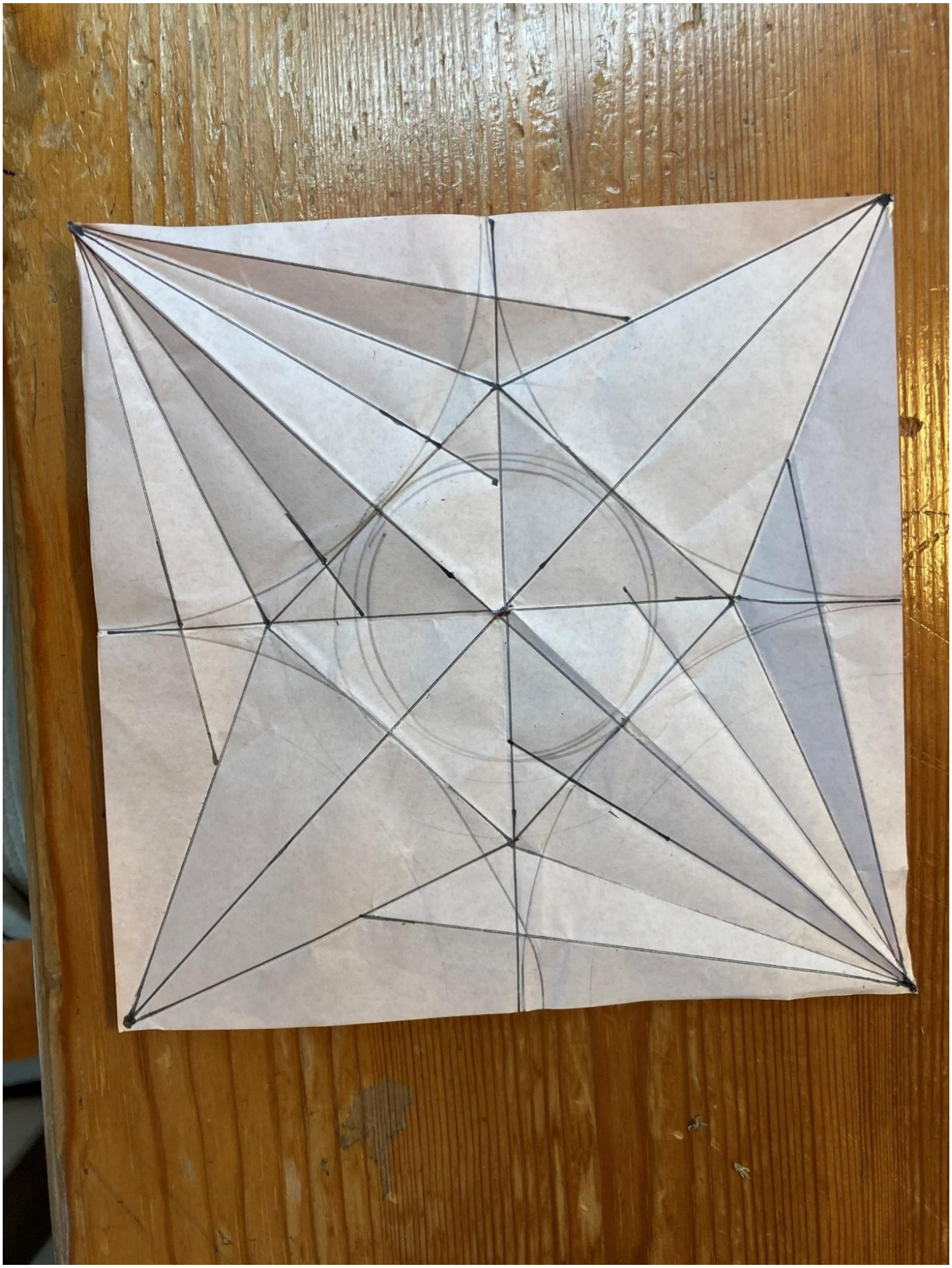


Figure 23. Bird base investigation with circles

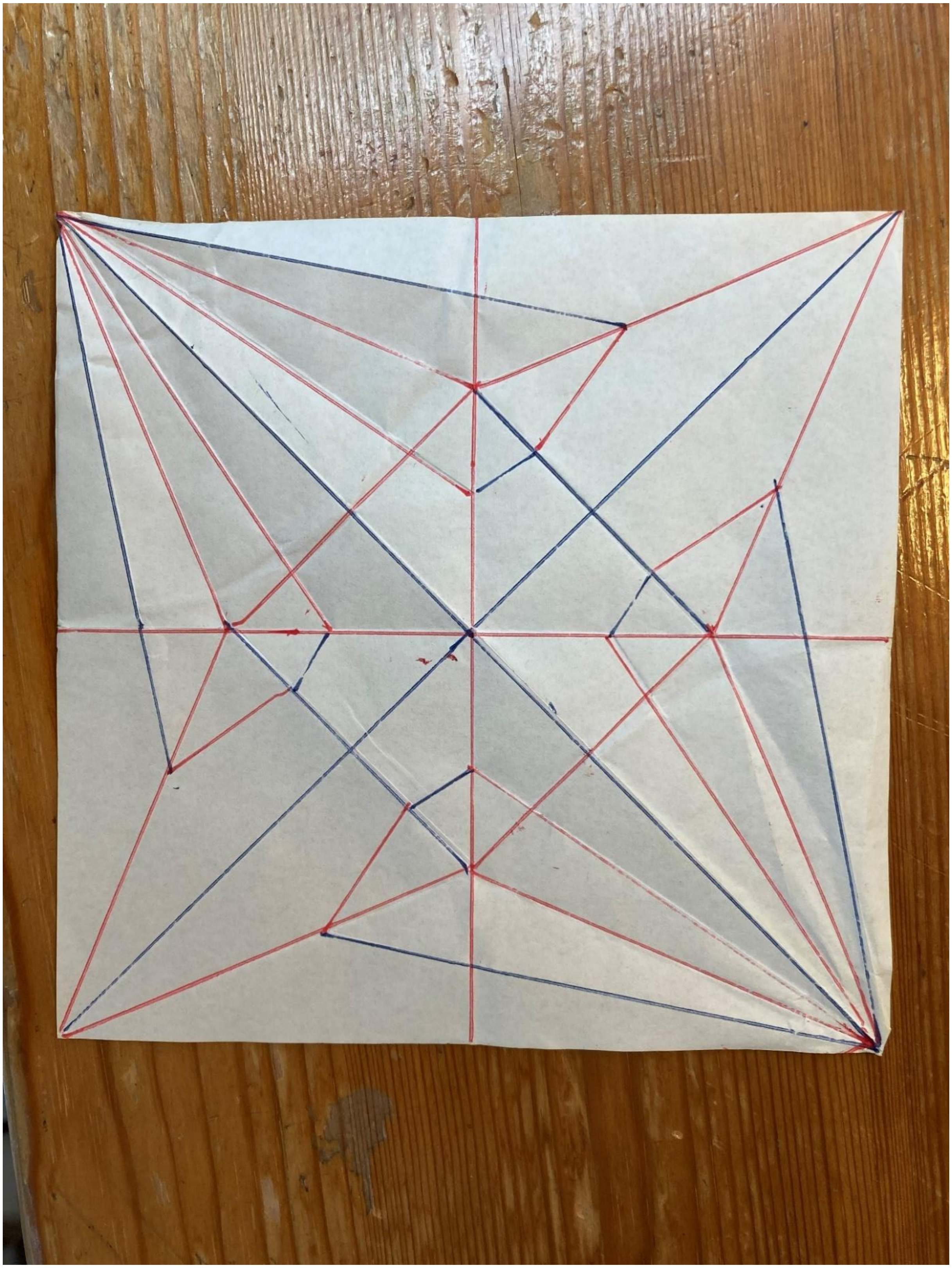
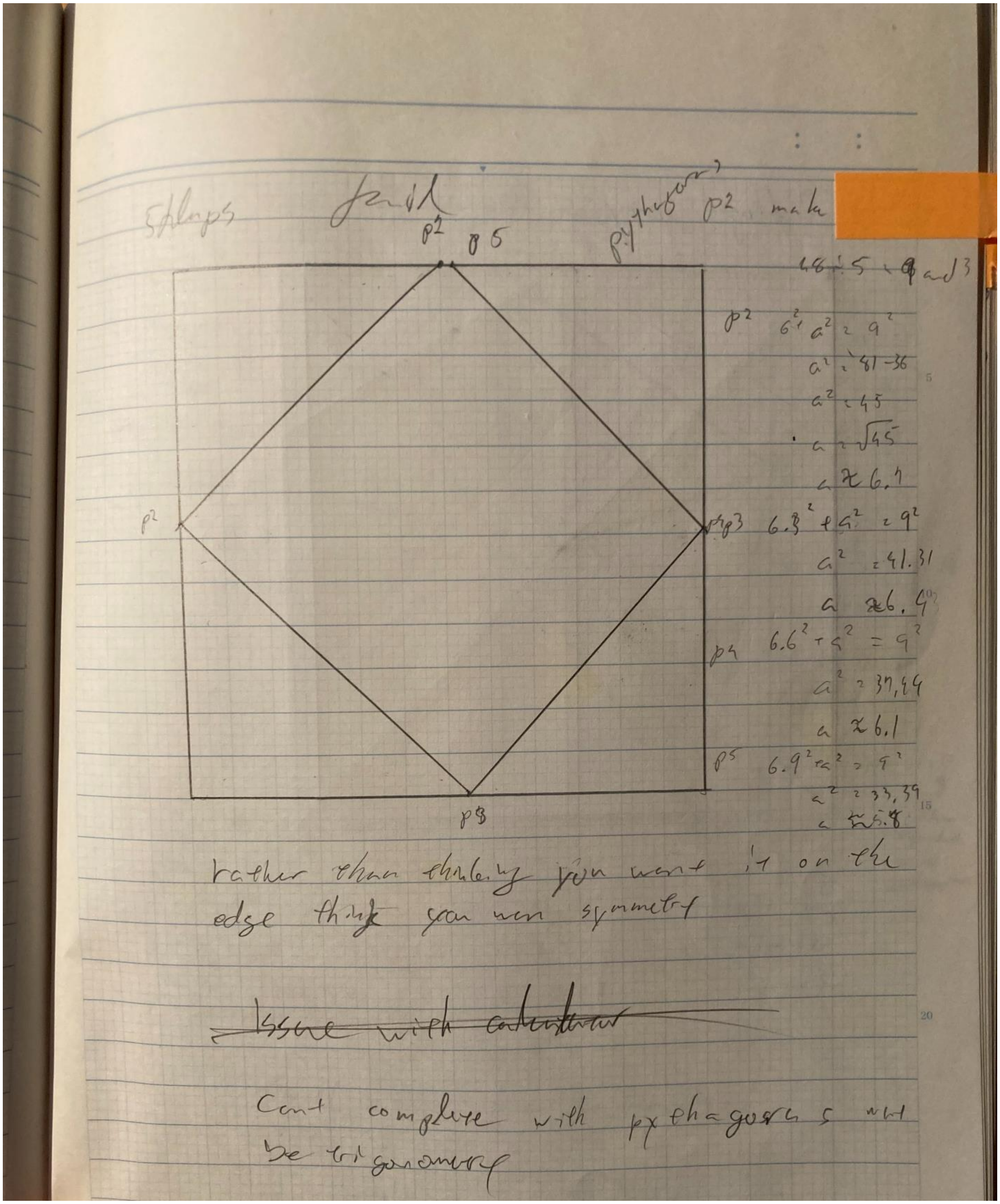


Figure 24. Bird base mountain and valley fold crease pattern investigation

Appendix 2. Experiment 1 data


p1
p5
p2
p3
p4

5 flaps [illegible] pythagoras? p2 make

48÷5 ≈ 9 and 3

p2 $6^2+a^2=9^2$

$a^2=81-36$

$a^2=45$

$a=\sqrt{45}$

$a\approx6.7$

p3 $6.3^2+a^2=9^2$

$a^2=41.31$

$a\approx6.4$

p4 $6.6^2+a^2=9^2$

$a^2=37.44$

$a\approx6.1$

p5 $6.9^2+a^2=9^2$

$a^2=33.39$

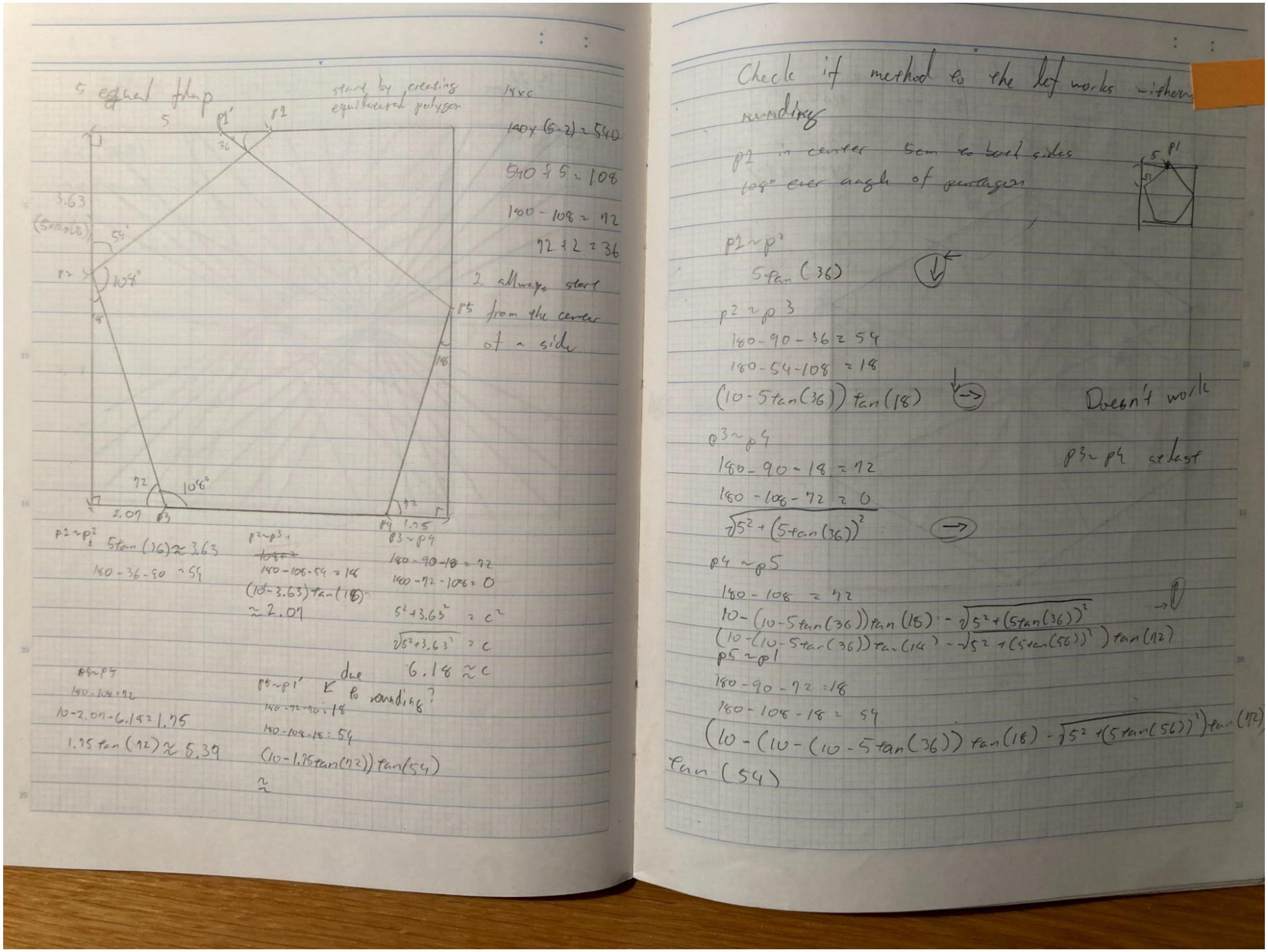
$a\approx5.8$

rather than thinking you want it on the edge think you want symmetry

~~Issue with calculator~~

Can't complete with pythagoras will be trigonometry

Figure 14. Testing the method of stage 1 experiment 1 for 5 flaps


5
p1'
p2
36
3.63
(5×tan(36))
54°
p2
108°
18
72
108°
2.07
p3
p4
1.75
72
18
p5
5
p1
5

5 equal flap

start by creating equilateral polygon

1ㄨ6

$140 \times (5-2) = 540$

$540 \div 5 = 108$

$180 - 108 = 72$

$72 \div 2 = 36$

2. allways start from the center of a side

$p1 \sim p^2_1$ $5\tan(36) \approx 3.63$

$180 - 36 - 90 = 54$

$p^2 \sim p^3$

~~108~~

$180 - 108 - 54 = 18$

$(10 - 3.63)\tan(18)$

≈ 2.07

p3 ~ p4

$180 - 90 - 18 = 72$

$180 - 72 - 108 = 0$

$5^2 + 3.63^2 = c^2$

$\sqrt{5^2 + 3.63^2} = c$

$6.18 \approx c$

p4 ~ p5

$180 - 108 = 72$

$10 - 2.07 - 6.18 = 1.75$

$1.75\tan(72) \approx 5.39$

p5 ~ p1' ← due to rounding?

$180 - 72 - 90 = 18$

$180 - 108 - 18 = 54$

$(10 - 1.75\tan(72))\tan(54)$

$\approx$

Check if method to the left works without rounding

~~p1 in center 5cm to both sides~~

~~108° ever angle of pentagon~~

$p1 \sim p^2$

$5\tan(36)$ ↓

$p2 \sim p3$

$180 - 90 - 36 = 54$

$180 - 54 - 108 = 18$

$(10 - 5\tan(36))\tan(18)$ →

Doesn't work

$p3 \sim p4$

p3 ~ p4 at last

$180 - 90 - 18 = 72$

$180 - 108 - 72 = 0$

$\sqrt{5^2 + (5\tan(36))^2}$ →

$p4 \sim p5$

$180 - 108 = 72$

$10 - (10 - 5\tan(36))\tan(18) - \sqrt{5^2 + (5\tan(36))^2}$ → 0

$(10 - (10 - 5\tan(36))\tan(18) - \sqrt{5^2 + (5\tan(56))^2})\tan(72)$

$p5 \sim p1$

$180 - 90 - 72 = 18$

$180 - 108 - 18 = 54$

$(10 - (10 - (10 - 5\tan(36))\tan(18) - \sqrt{5^2 + (5\tan(56))^2})\tan(72)$

$\tan(54)$

Figure 15. Testing with the method of stage 1 experiment 1 for 5 flaps

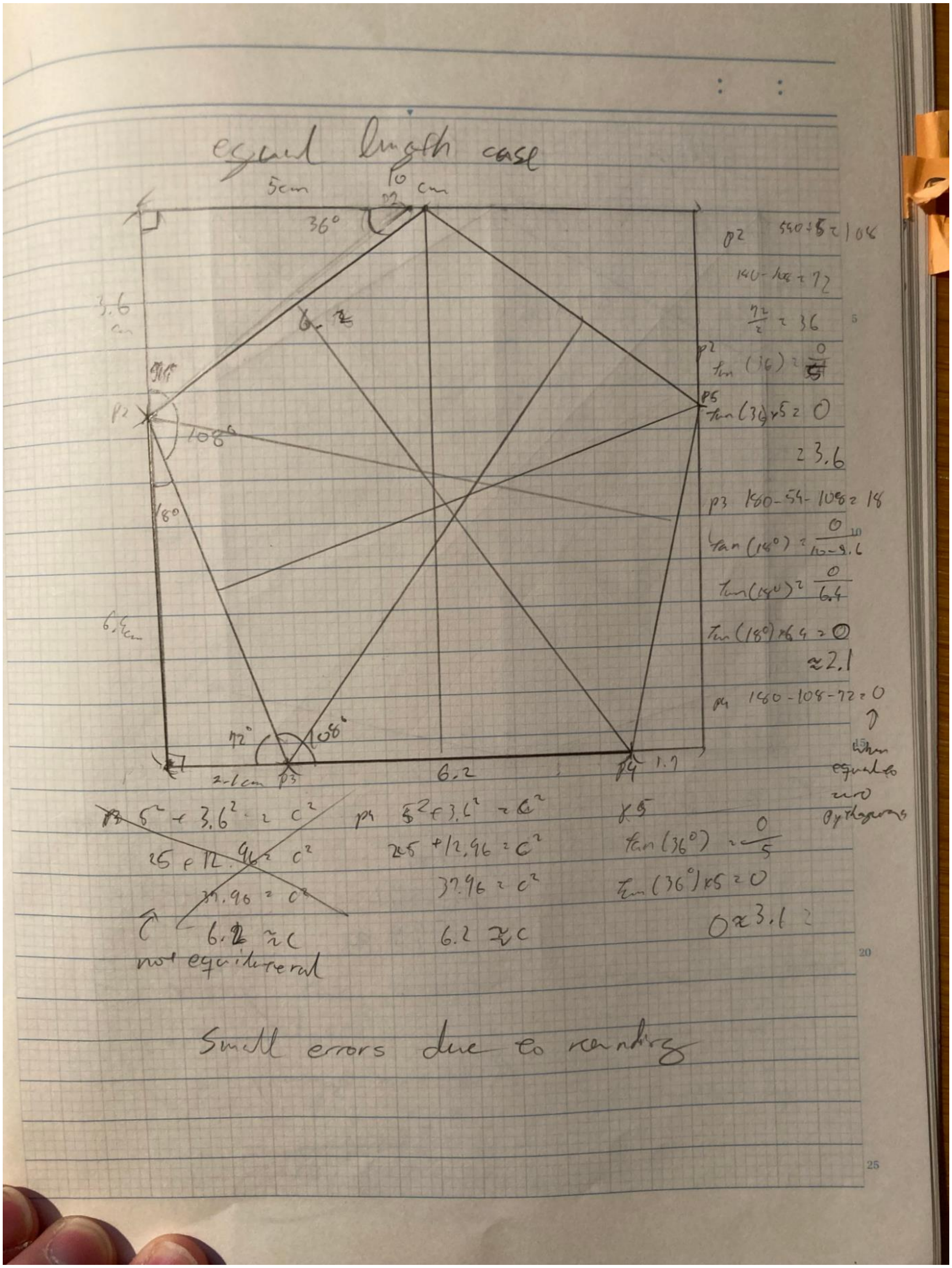
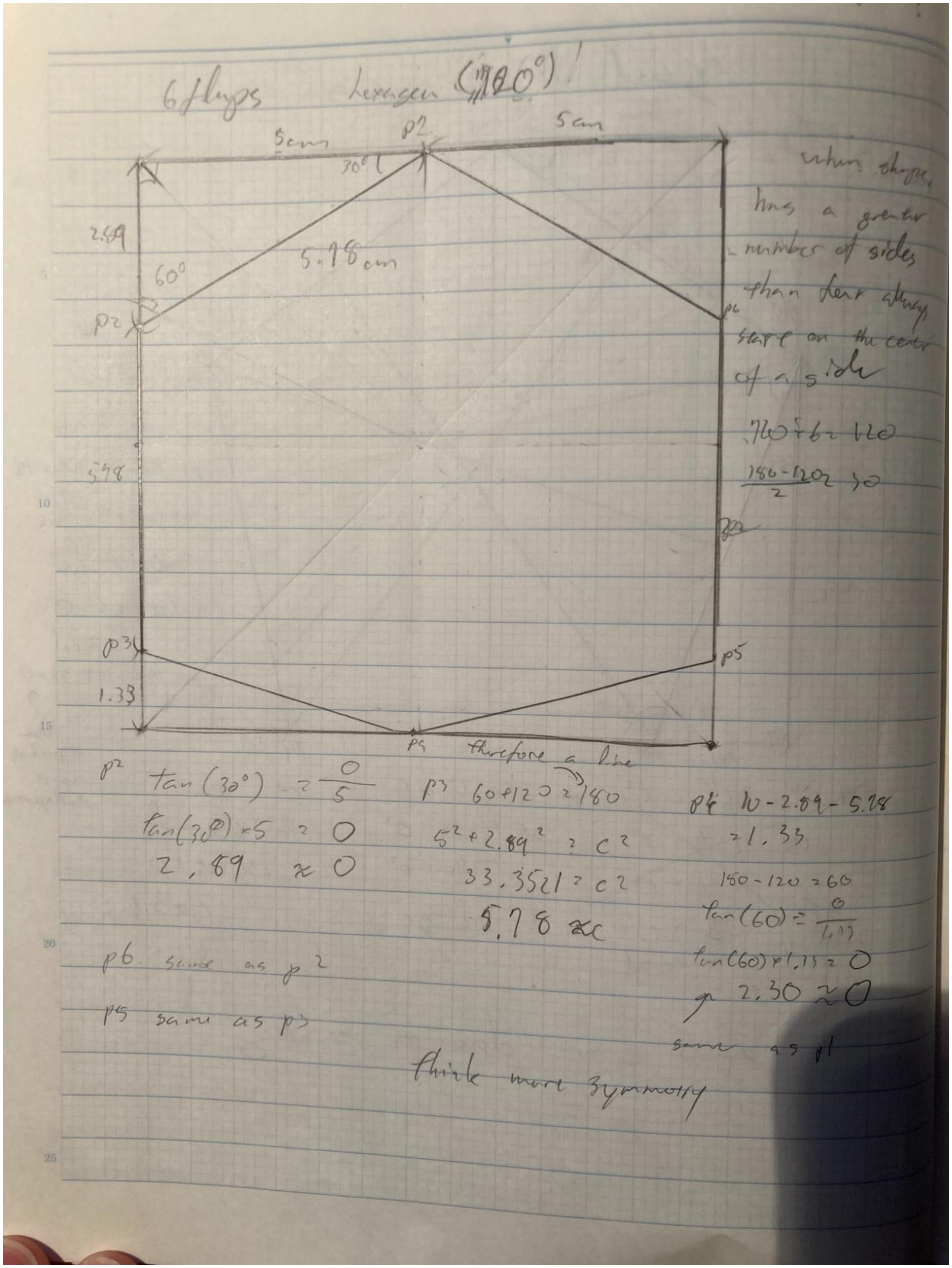


Figure 16. Testing method of stage 1 experiment 1 for 5 flaps



6 flaps hexagon (720°)!

when shapes has a greater number of sides than four always start on the center of a side

720 ÷ 6 = 120

$\frac{180-120}{2} = 30$

therefore a line

p2 $\tan(30°) = \frac{O}{5}$

$\tan(30°) \times 5 = O$

$2,89 \approx O$

p3 $60+120 = 180$

$5^2 + 2.89^2 = c^2$

$33.3521 = c^2$

$5.78 \approx c$

p4 $10 - 2.89 - 5.78$

$= 1.33$

$180 - 120 = 60$

$\tan(60) = \frac{O}{1.33}$

$\tan(60) \times 1.33 = O$

$2.30 \approx O$

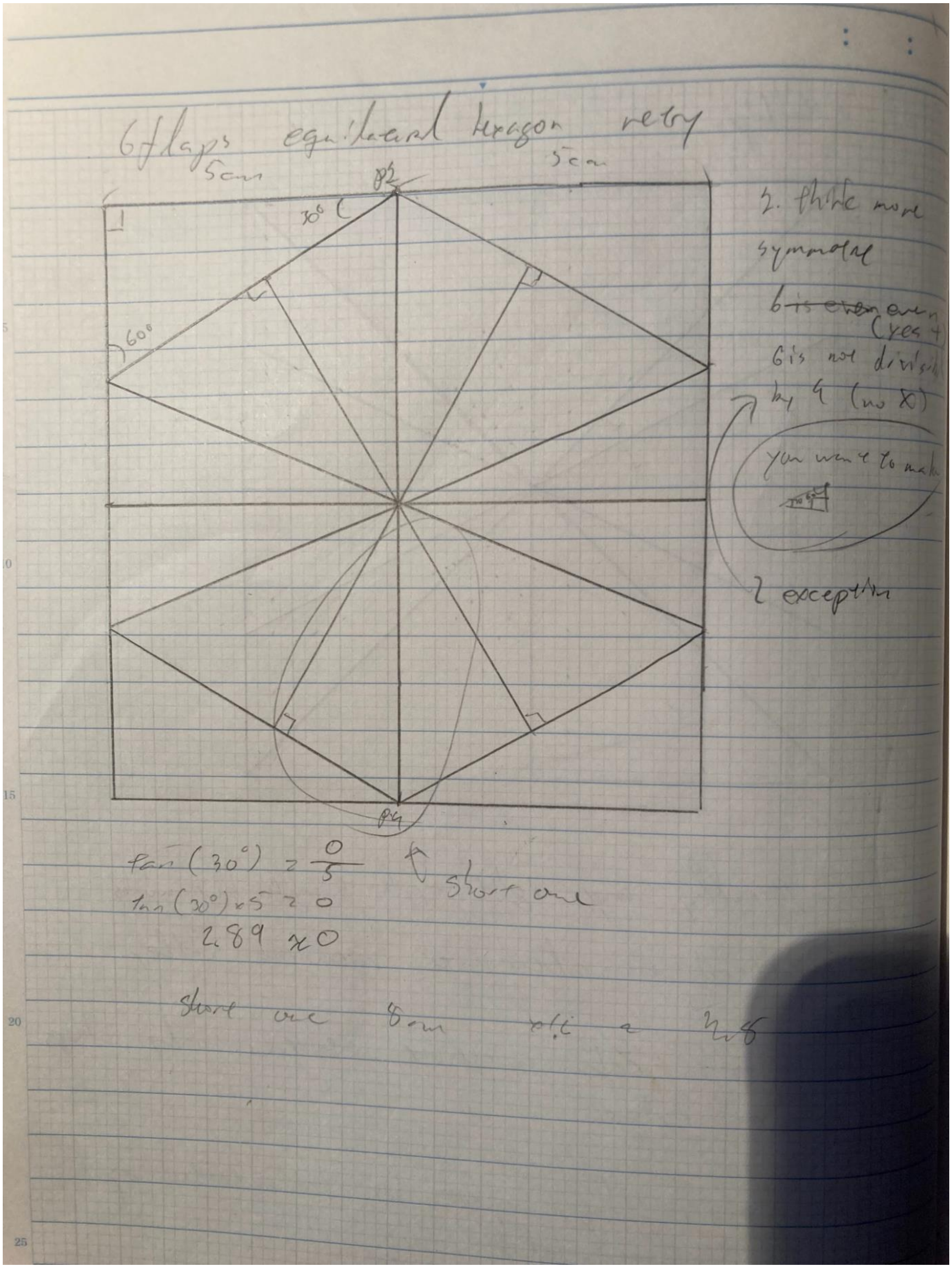
same as p1

p6 same as p2

p5 same as p3

think more symmetry

Figure 17Testing method of stage 1 experiment 1 for 6 flaps



6 flaps equilateral hexagon retry

2. think more symmetrical

6 ~~is even~~ even (yes +)
6 is not divisible by 4 (no X)

you want to make

2 exception

$\tan(30°) = \frac{O}{5}$ ↑ short one

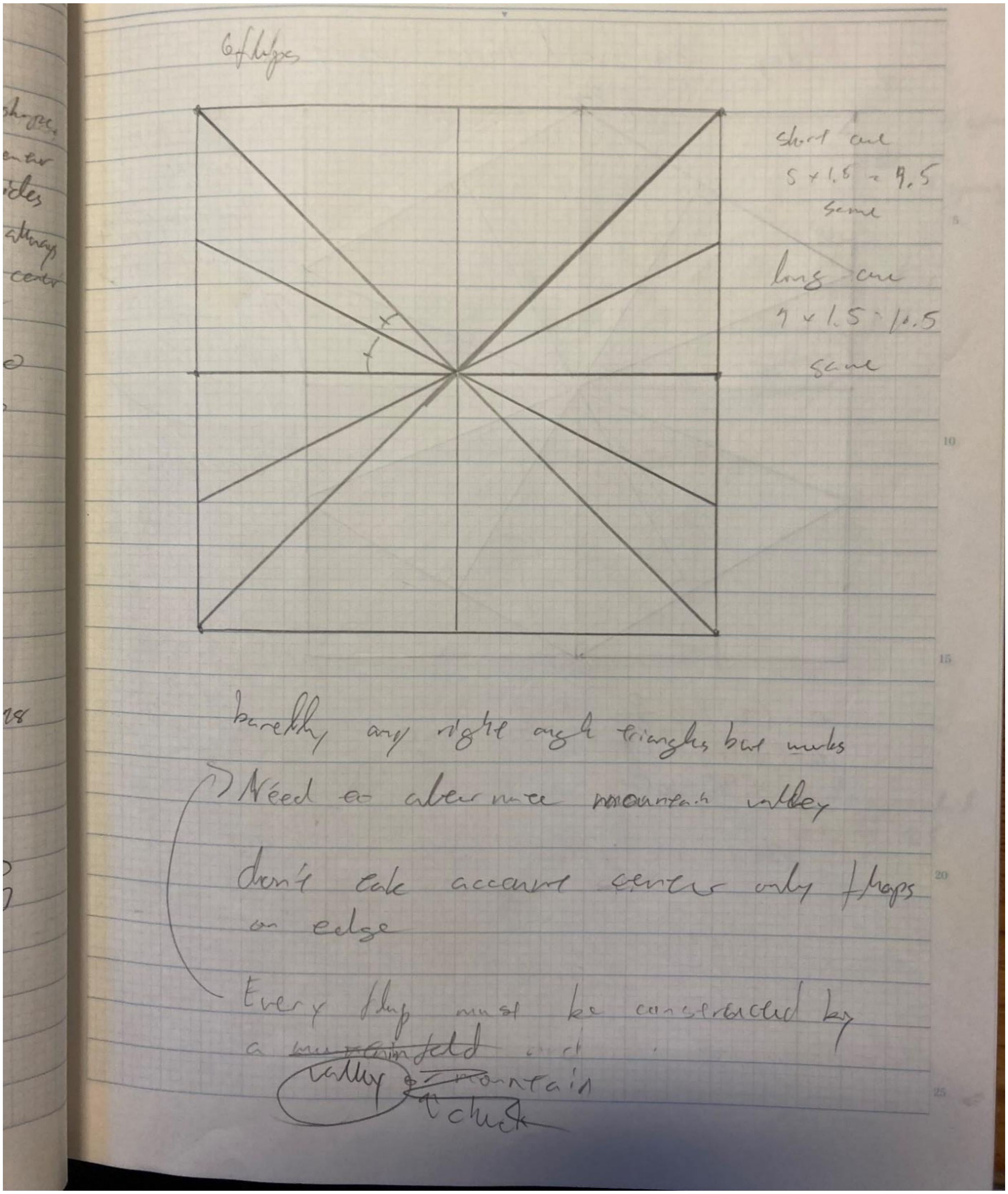
$\tan(30°) \times 5 = O$

$2.89 \approx O$

short one 8cm x.6 = 4.8

Figure 18. Testing method of stage 1 experiment 1 for 6 flaps

Appendix 3. Experiment 2 data, pattern-based algorithm



6 flaps

short one
5 × 1.5 = 7.5
same

long one
7 × 1.5 = 10.5
same

barely any right angle triangles but works

Need to alternate mountain valley

don't take account center only flaps on edge

Every flap must be constructed by a ~~mountain~~ fold and valley ~~& mountain~~ ~~+ check~~

Figure 19. Designed crease pattern based on the method of stage 1 experiment 2 for 6 flaps

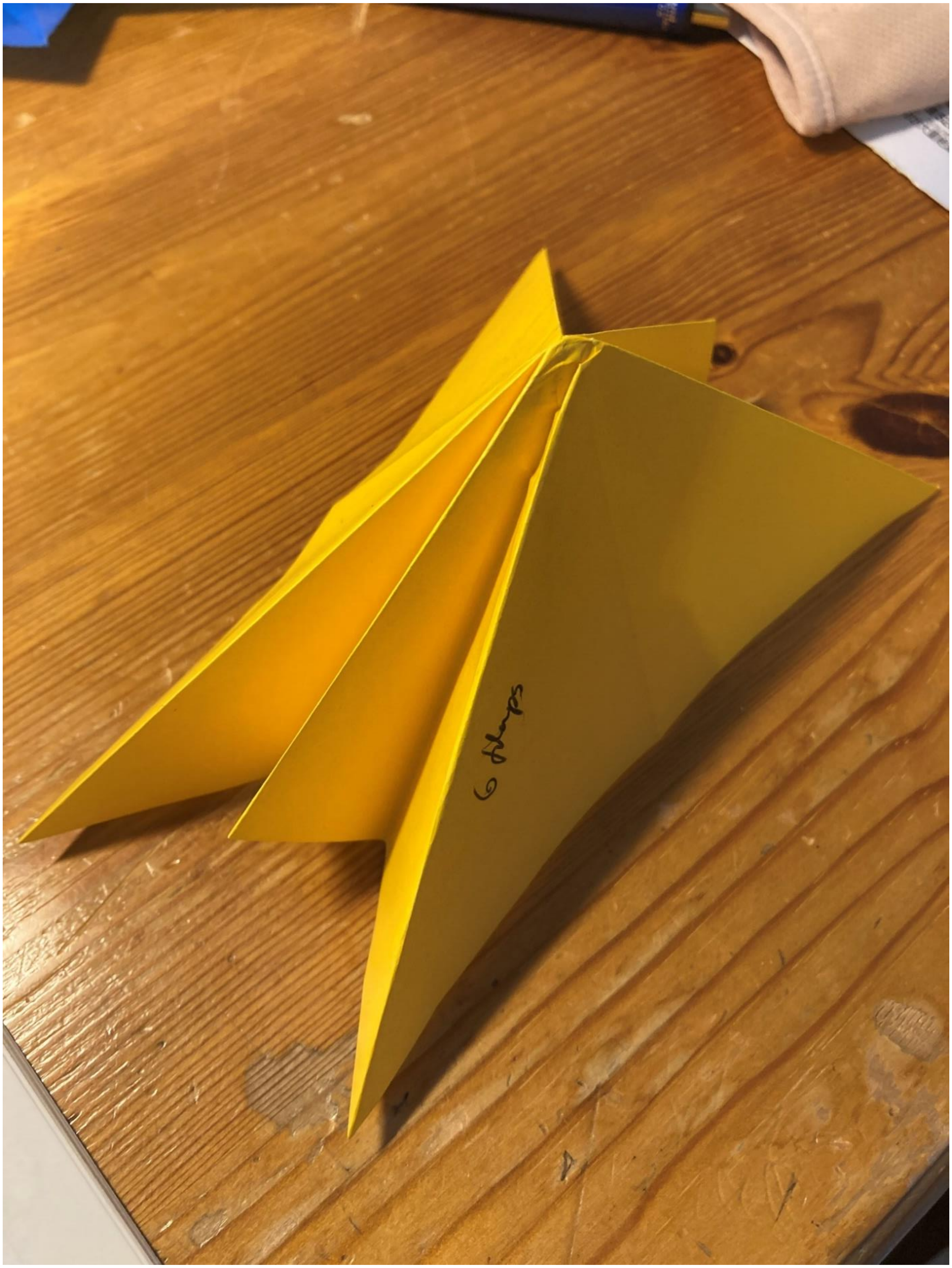
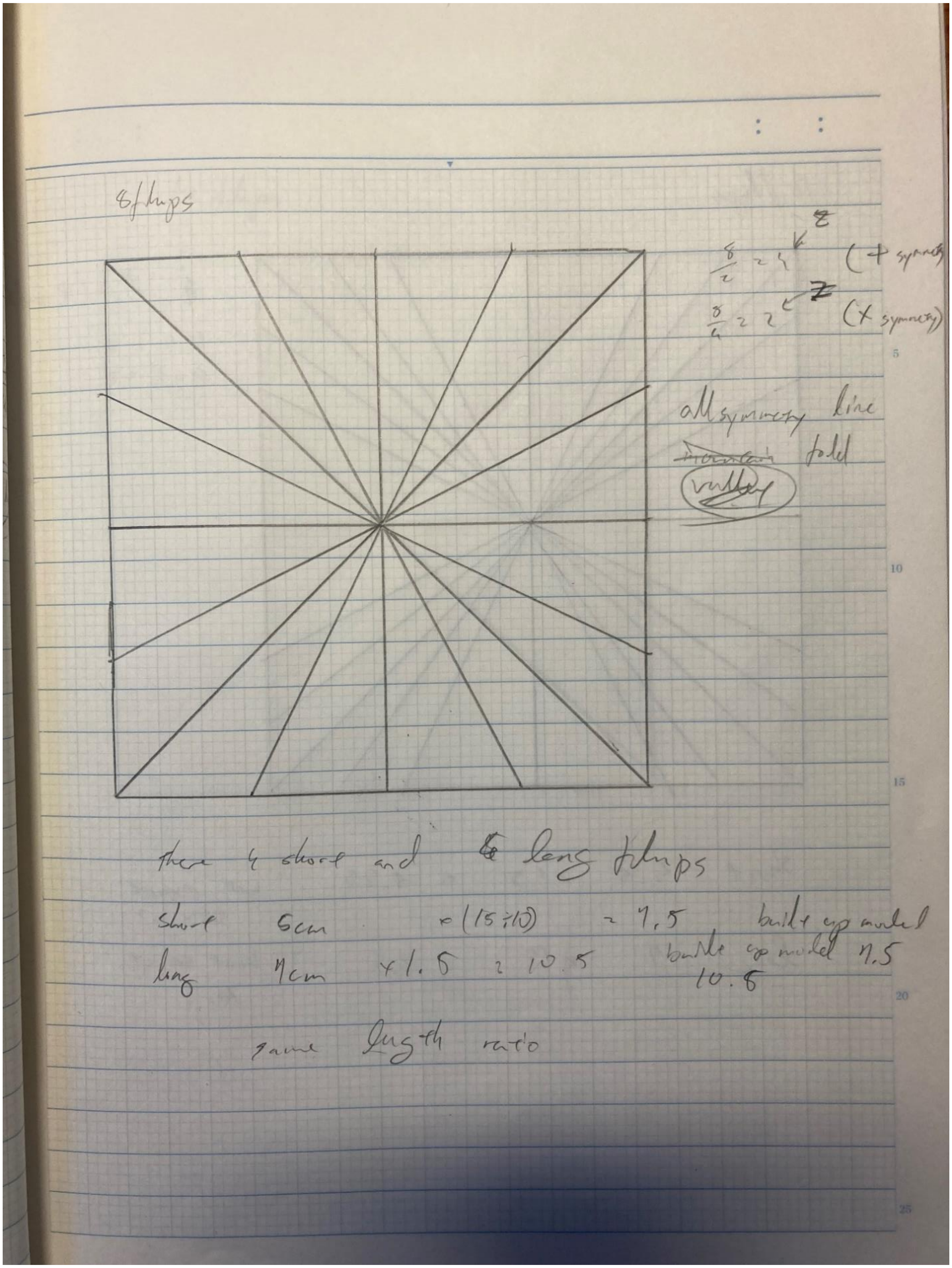


Figure 20. The folded structure of the crease pattern for 6 flaps above



8 flaps

$\frac{8}{2} = 4$ (+ symmetry)

$\frac{8}{4} = 2$ (× symmetry)

all symmetry line ~~mountain~~ fold valley

there 4 short and 4 long flaps

short 5cm ×(15÷10) = 7.5 built up model

long 7cm ×1.5 = 10.5 built up model 7.5 10.5

same length ratio

Figure 21. Designed crease pattern based on the method of stage 1 experiment 2 for 8 flaps

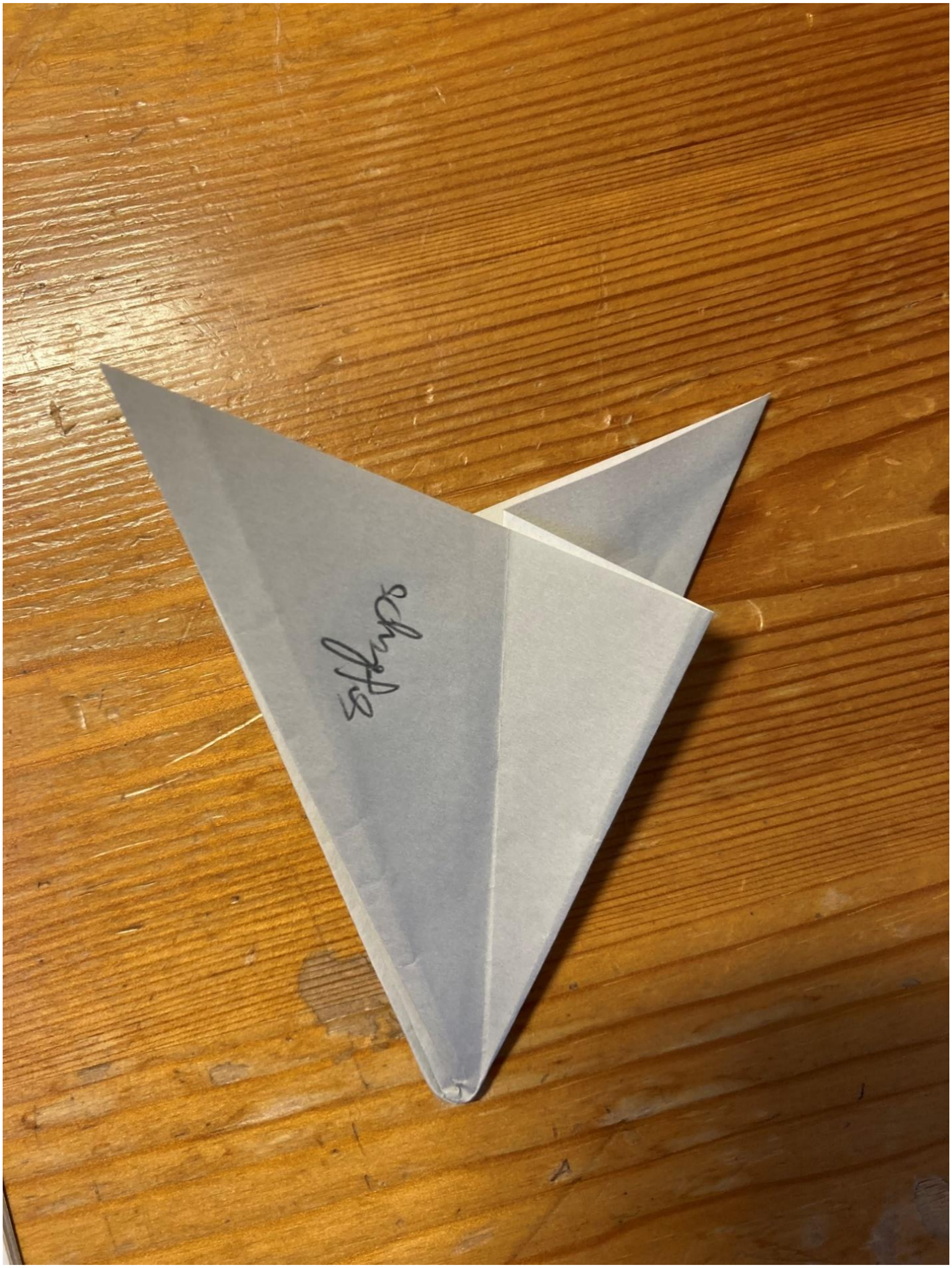


Figure 22. The folded structure of the crease pattern for 8 flaps above

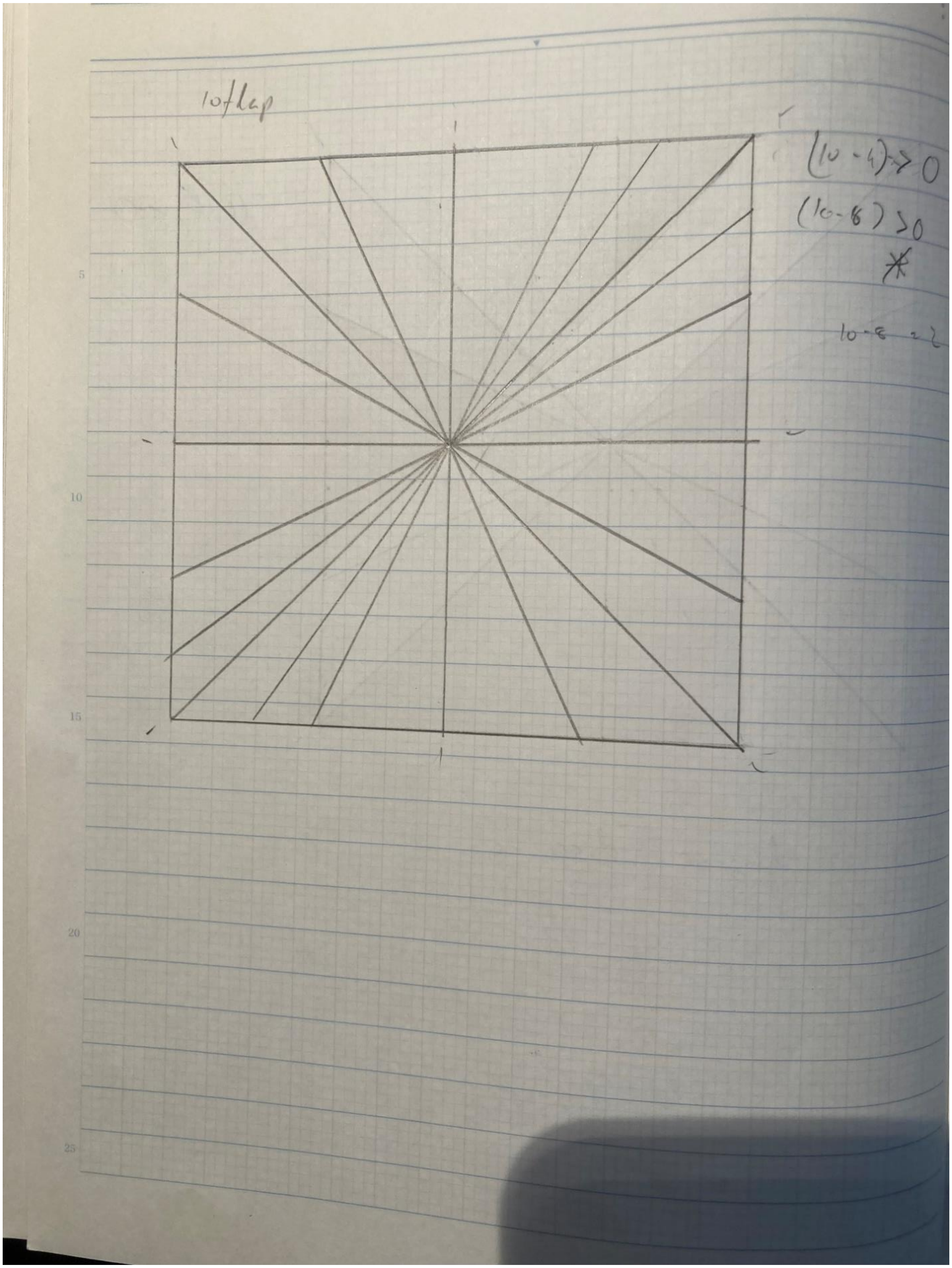


Figure 23. Designed crease pattern based on the method of stage 1 experiment 2 for 10 flaps

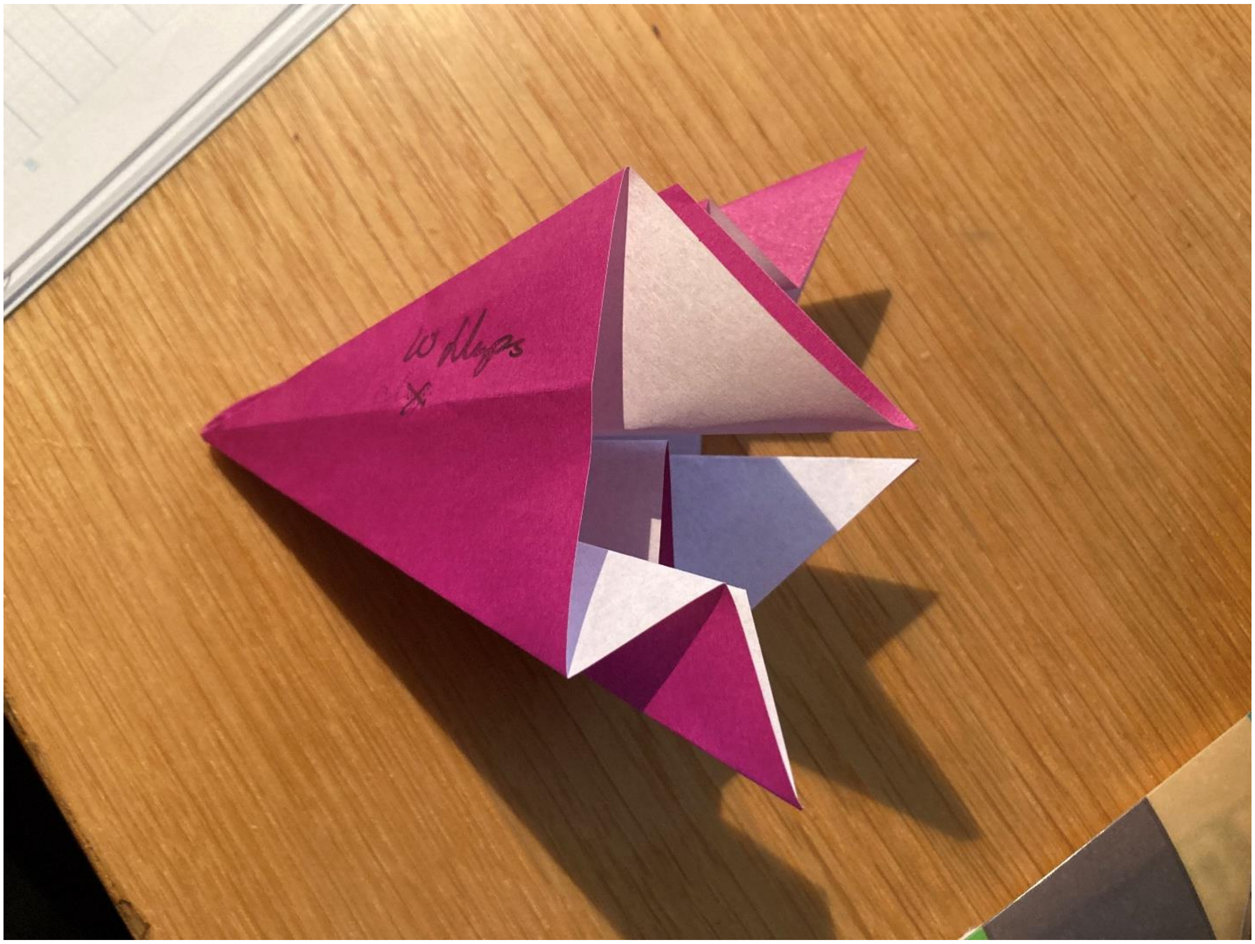


Figure 24. The folded structure of the crease pattern for 10 flaps above

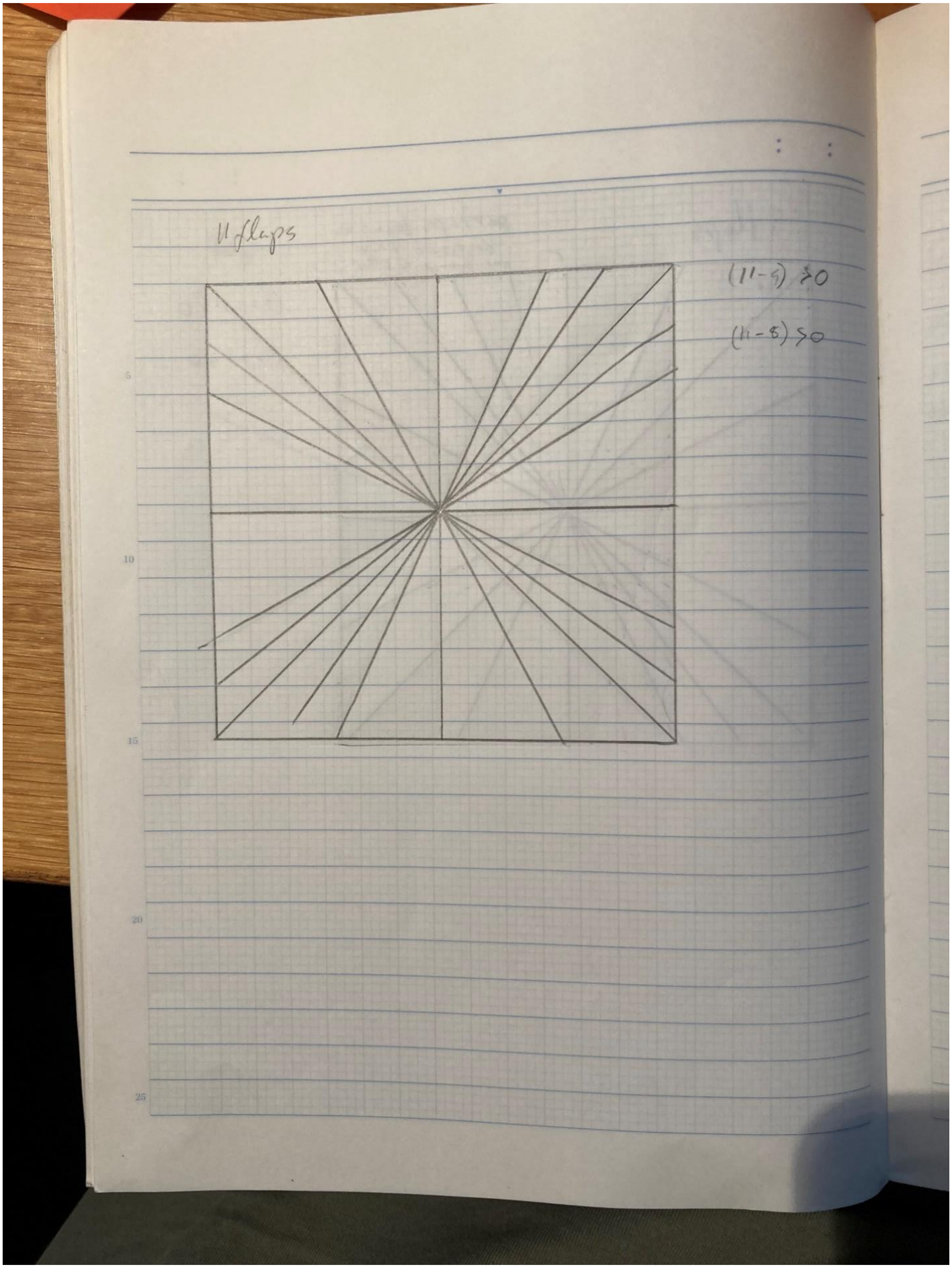


Figure 25. Designed crease pattern based on the method of stage 1 experiment 2 for 11 flaps

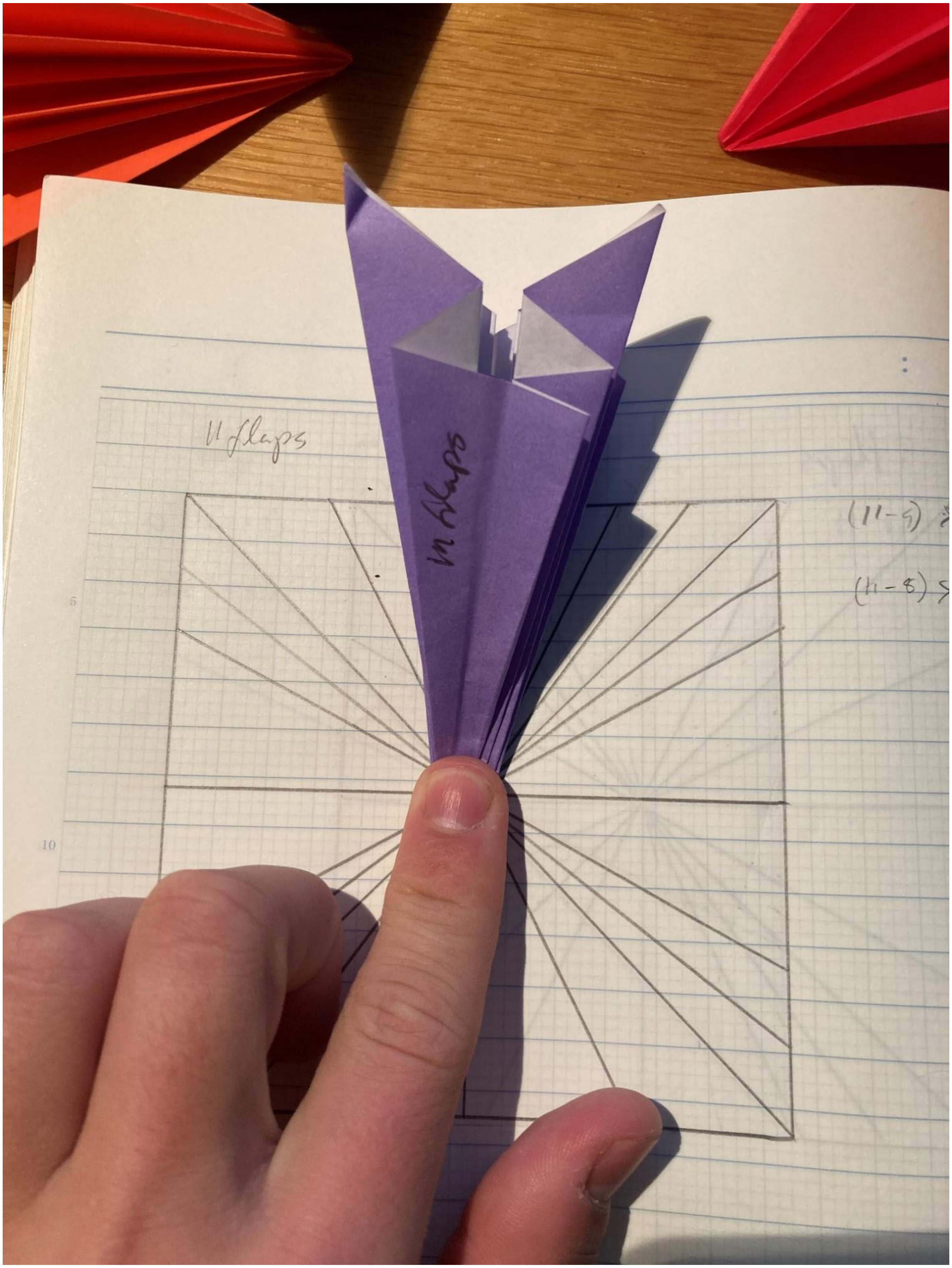


Figure 26. The folded structure of the crease pattern for 11 flaps above

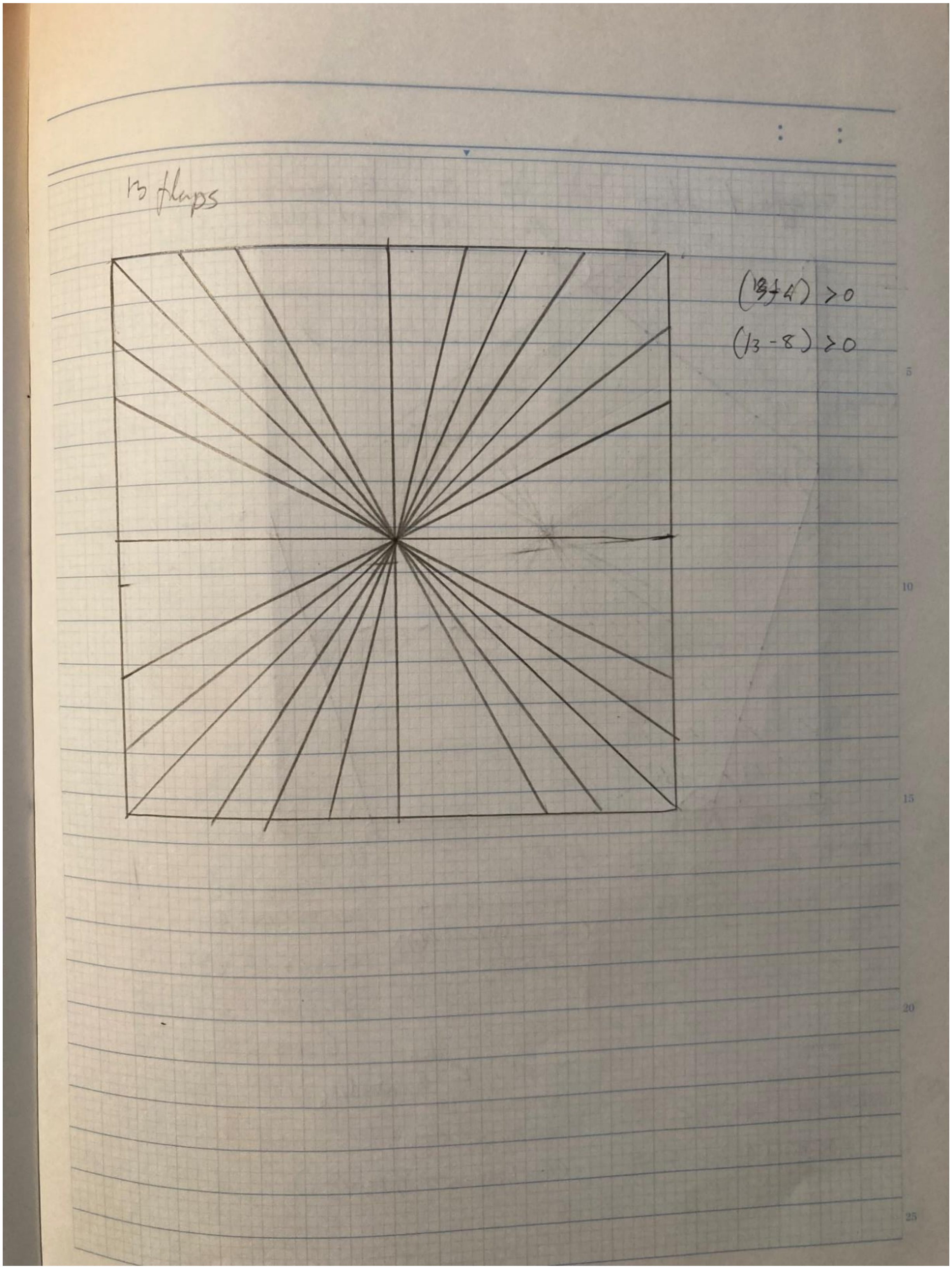


Figure 27. Designed crease pattern based on the method of stage 1 experiment 2 for 13 flaps

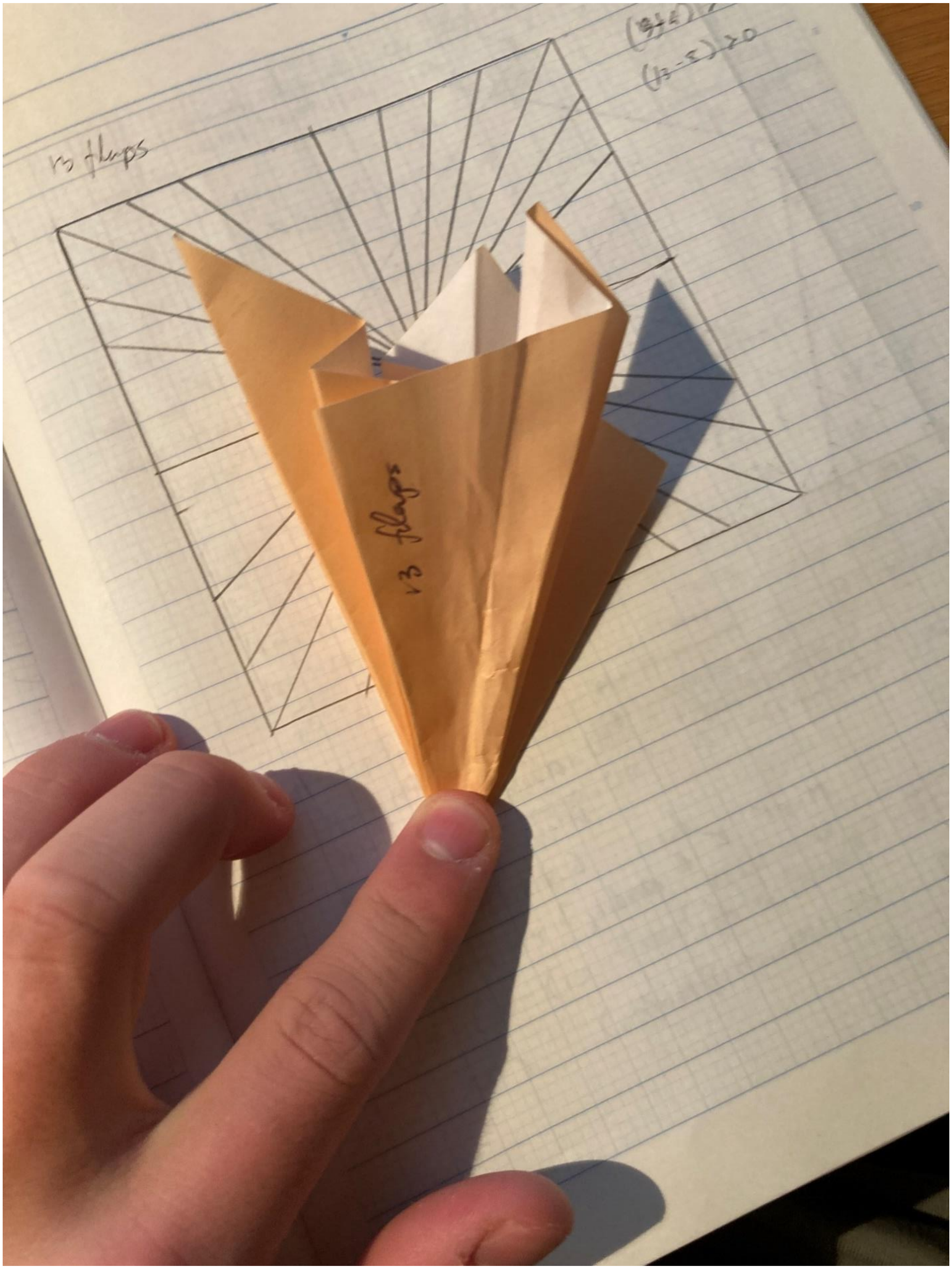


Figure 28. The folded structure of the crease pattern for 13 flaps above

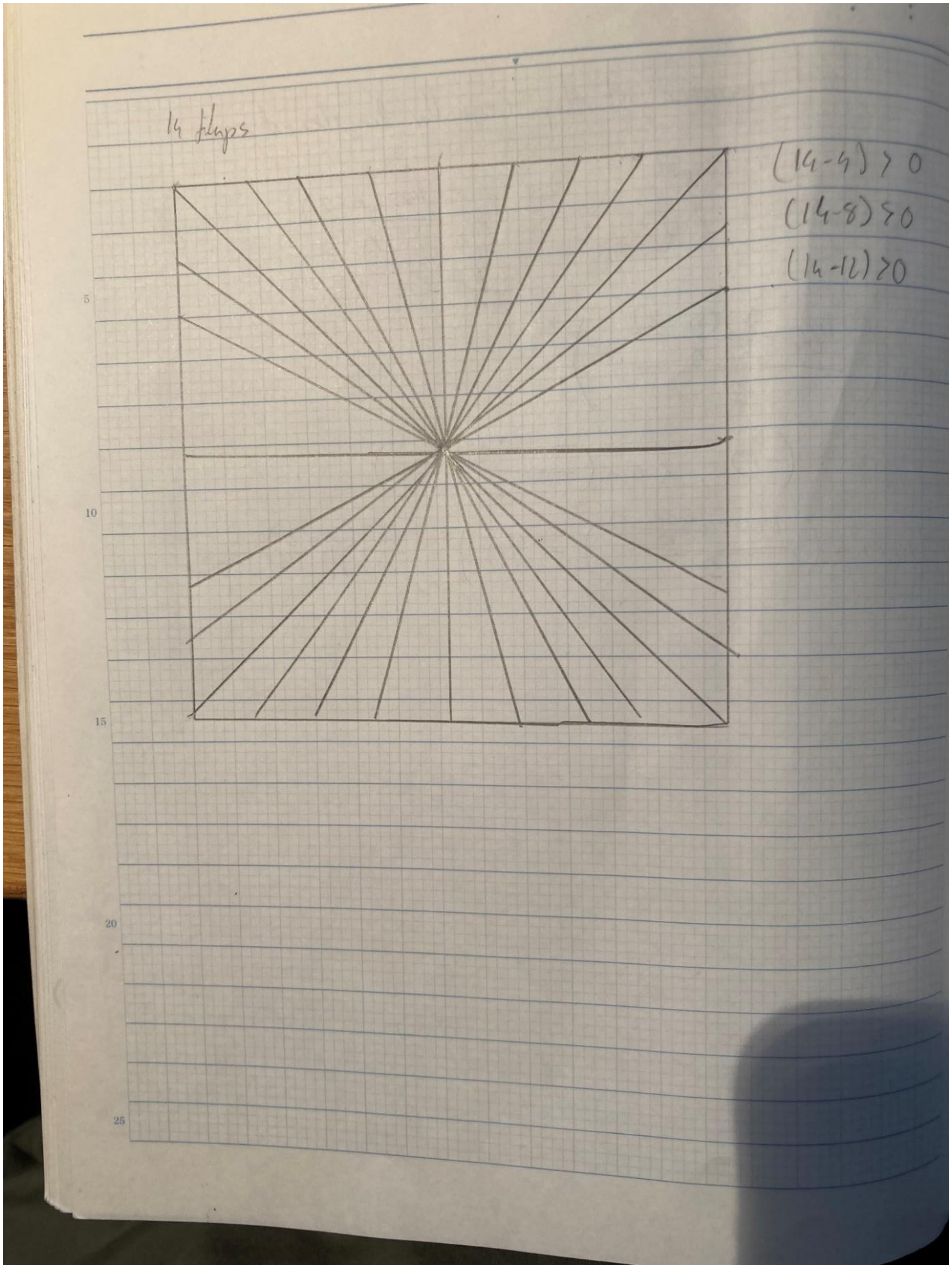


Figure 29. Designed crease pattern based on the method of stage 1 experiment 2 for 14 flaps

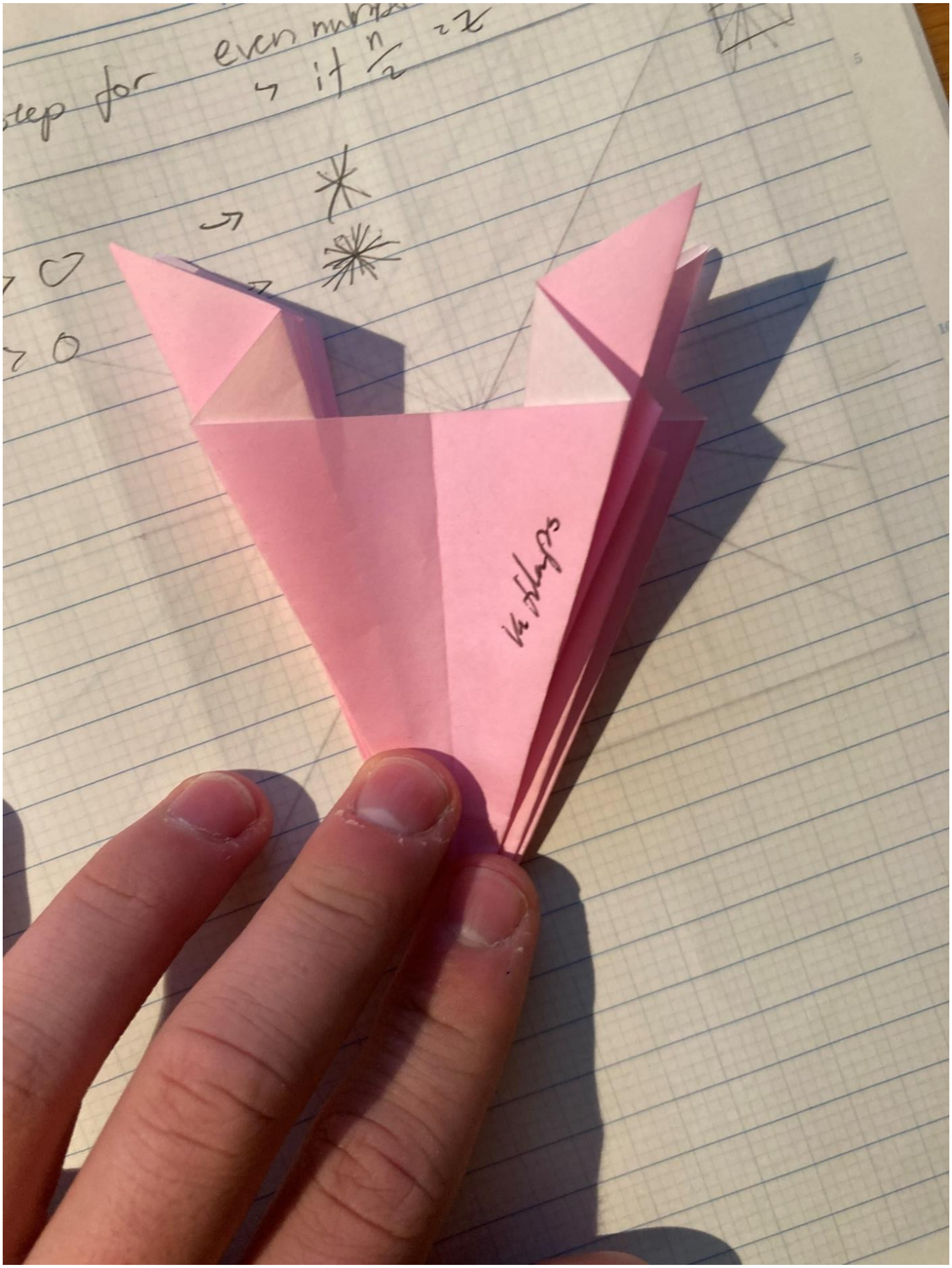


Figure 30. The folded structure of the crease pattern for 14 flaps above

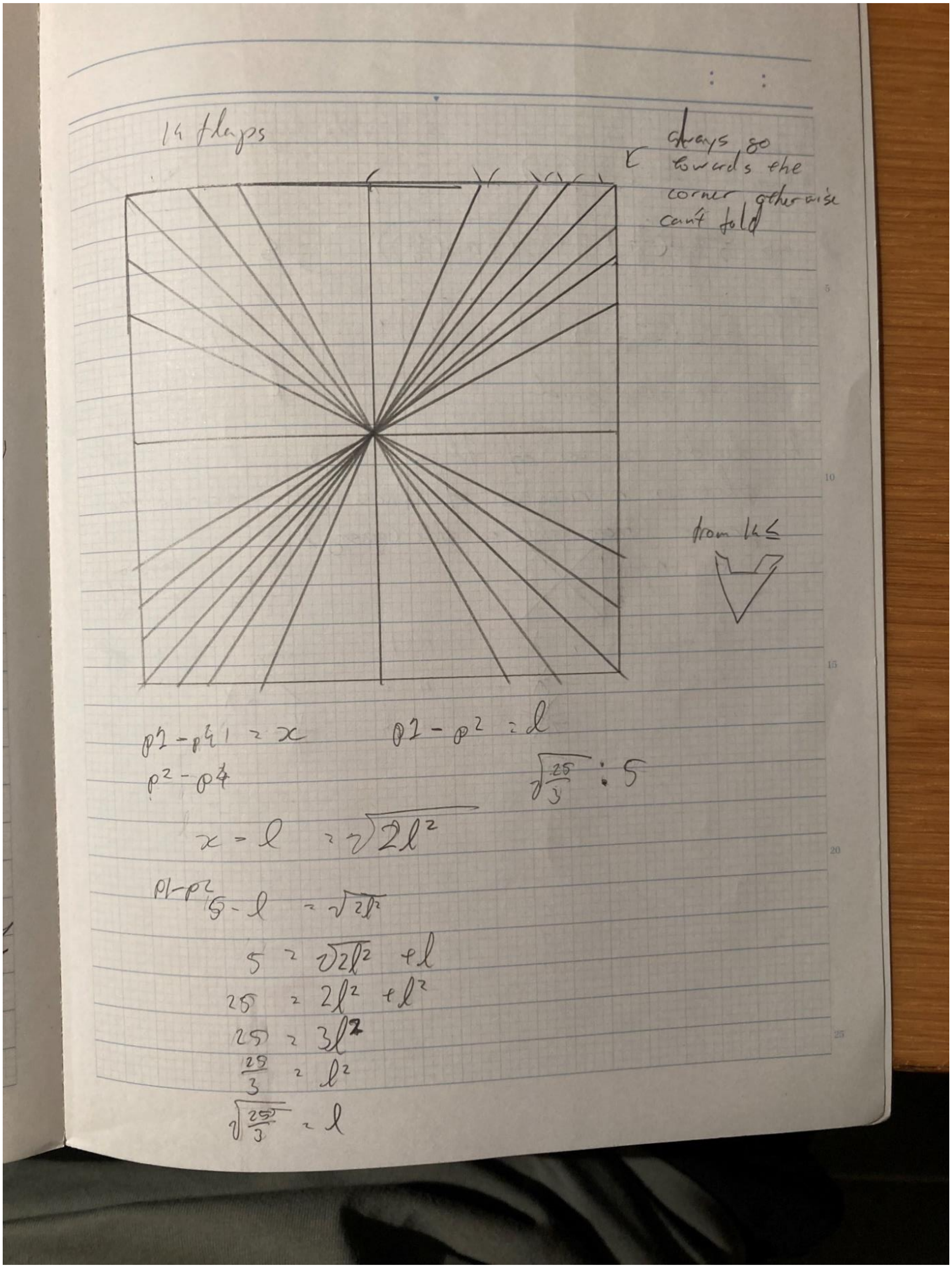


Figure 31. Designed crease pattern based on the method of stage 1 experiment 2 for 14 flaps different configuration

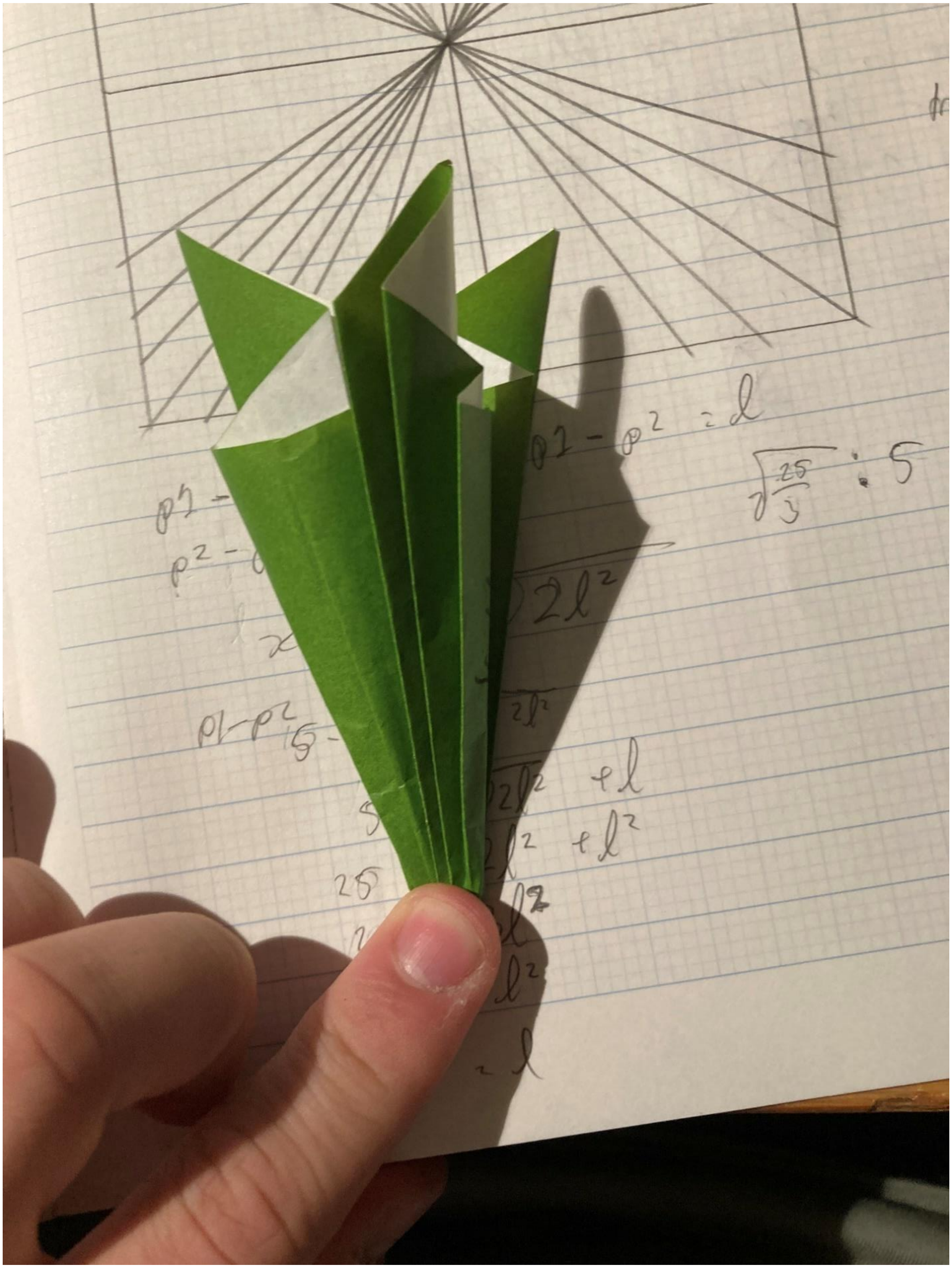


Figure 32. Folded structure of the crease patter for 14 flaps with a different configuration above

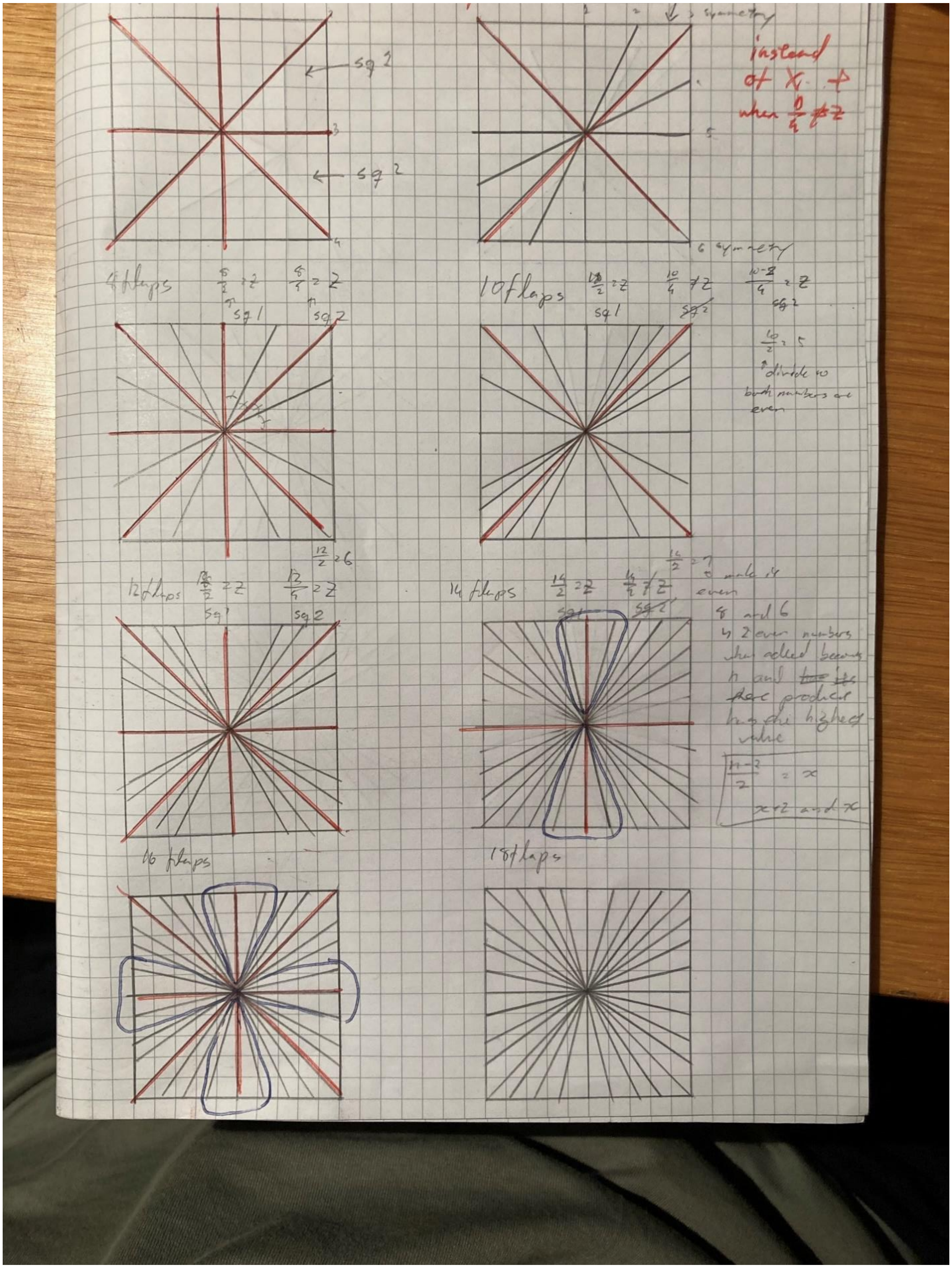


Figure 33. Designed crease pattern based on the method of stage 1 experiment 2 for 4, 8, 10, 12, 14, 16, 18 flaps

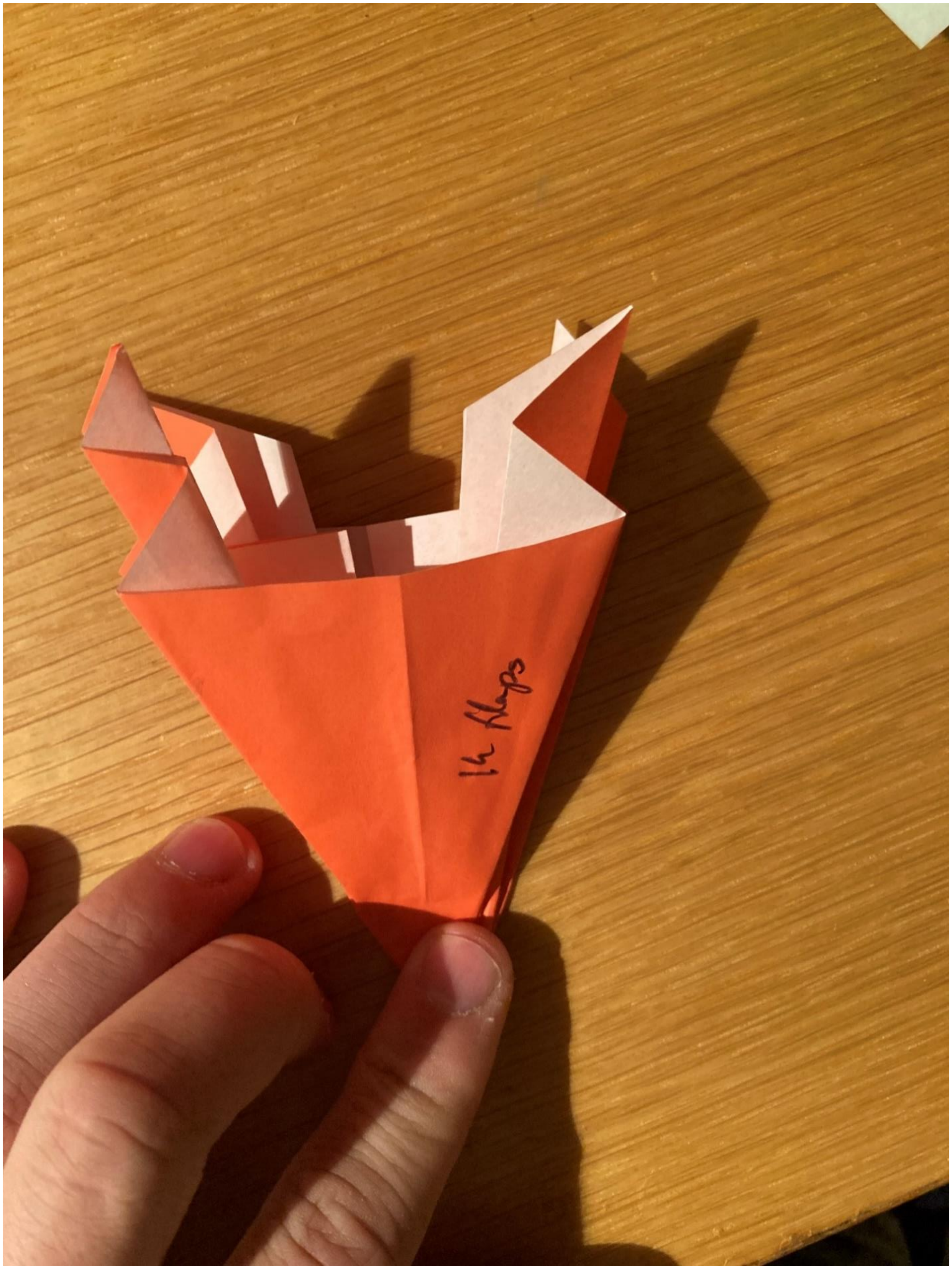


Figure 34. Folded structure using the method of stage 1 experiment 2 for 14 flaps

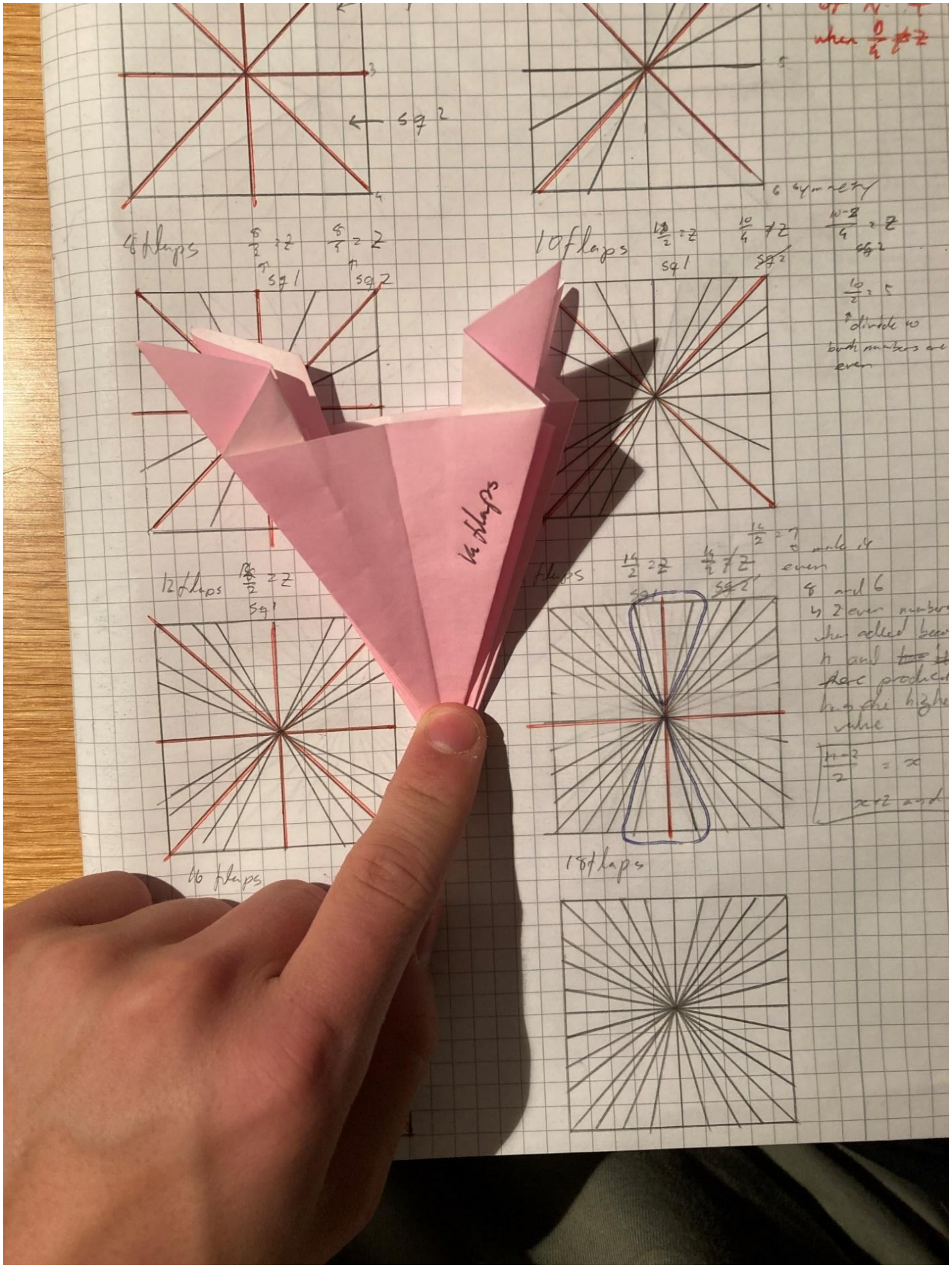


Figure 35. Folded structure using the method of stage 1 experiment 2 for 14 flaps different configuration

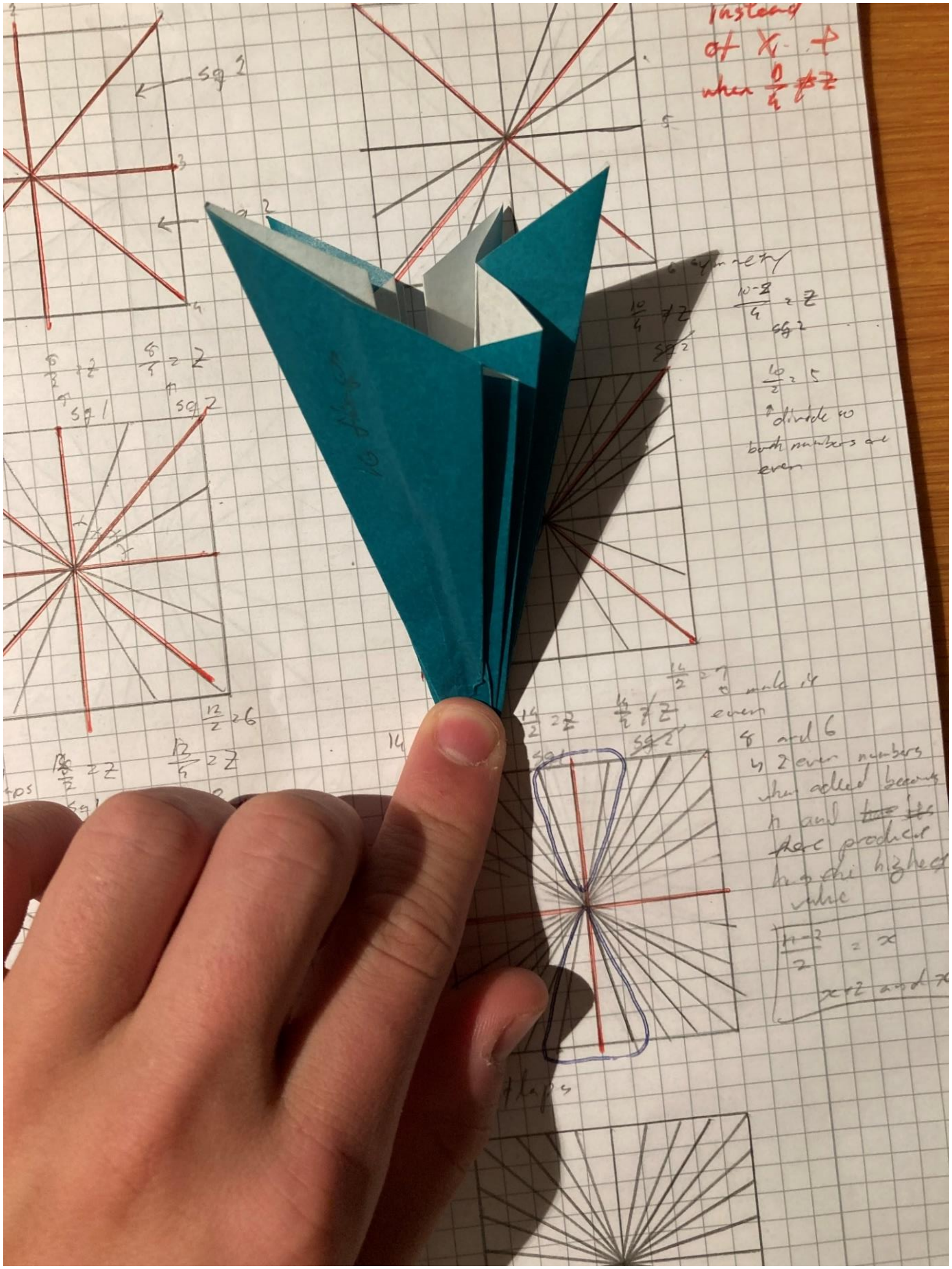


Figure 36. Folded structure using the method of stage 1 experiment 2 for 10 flaps

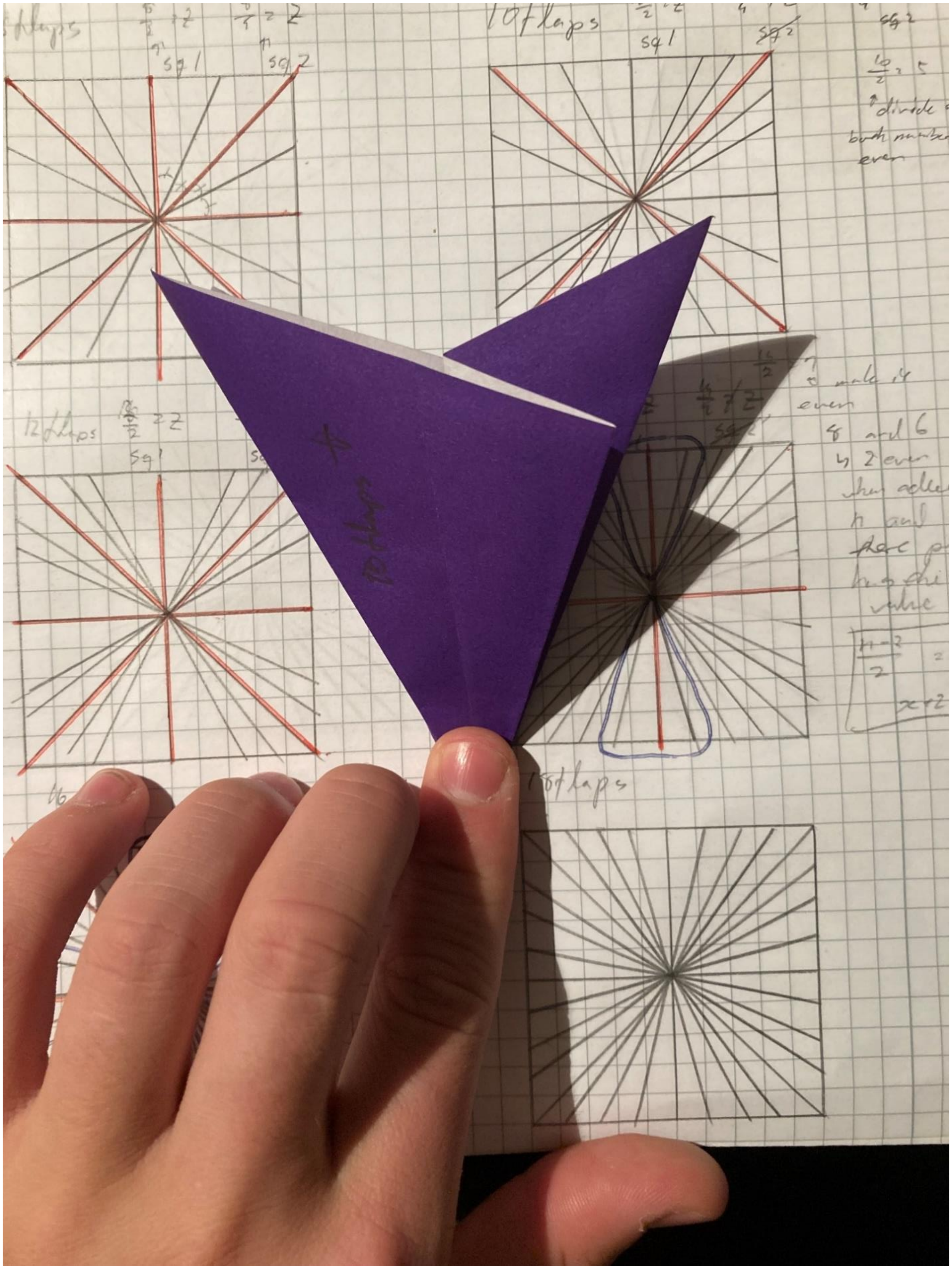


Figure 37. Folded structure using the method of stage 1 experiment 2 for 10 flaps differnet configuration

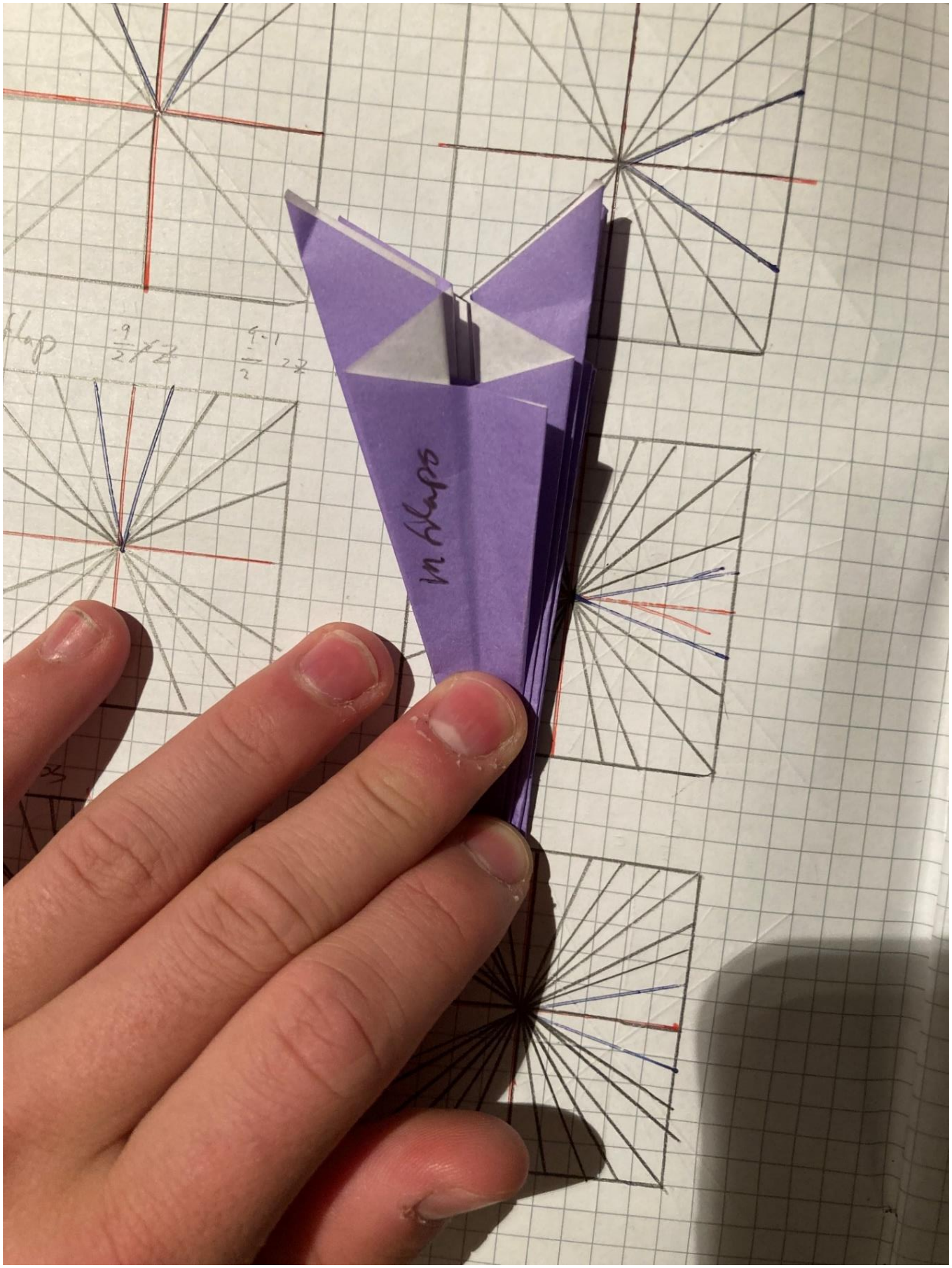
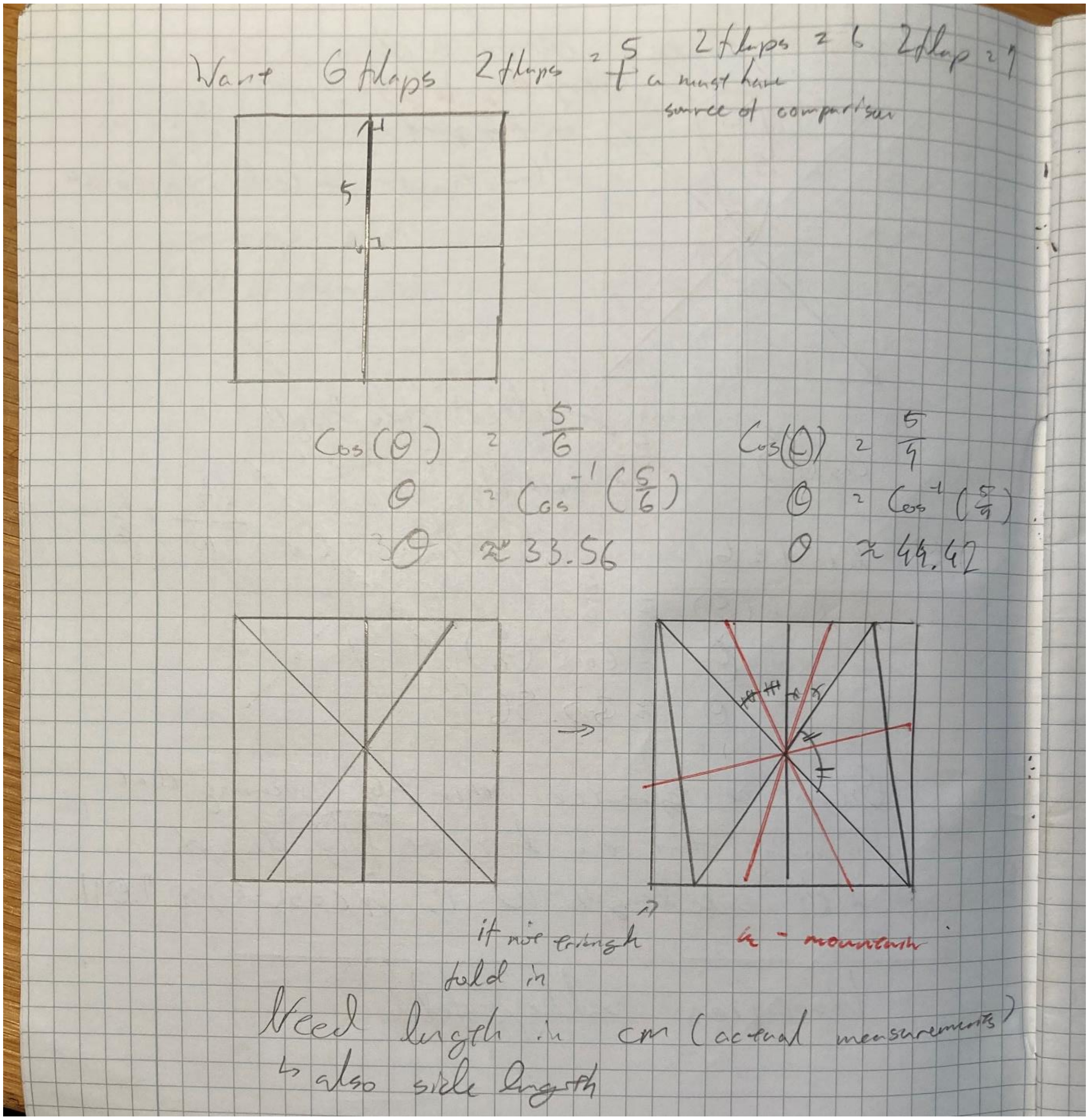


Figure 38. Folded structure using the method of stage 1 experiment 2 for 17 flaps

Appendix 4. Experiment 3 data, trigonometry-based algorithm



Want 6 flaps 2 flaps = 5 + a must have source of comparison 2 flaps = 6 2 flap = 7

$\cos(\theta) = \frac{5}{6}$

$\theta = \cos^{-1}\left(\frac{5}{6}\right)$

$\theta \approx 33.56$

$\cos(\theta) = \frac{5}{7}$

$\theta = \cos^{-1}\left(\frac{5}{7}\right)$

$\theta \approx 44.42$

if not enough fold in

R - mountain

Need length in cm (actual measurements)

↳ also side length

Figure 39. Six flap crease pattern

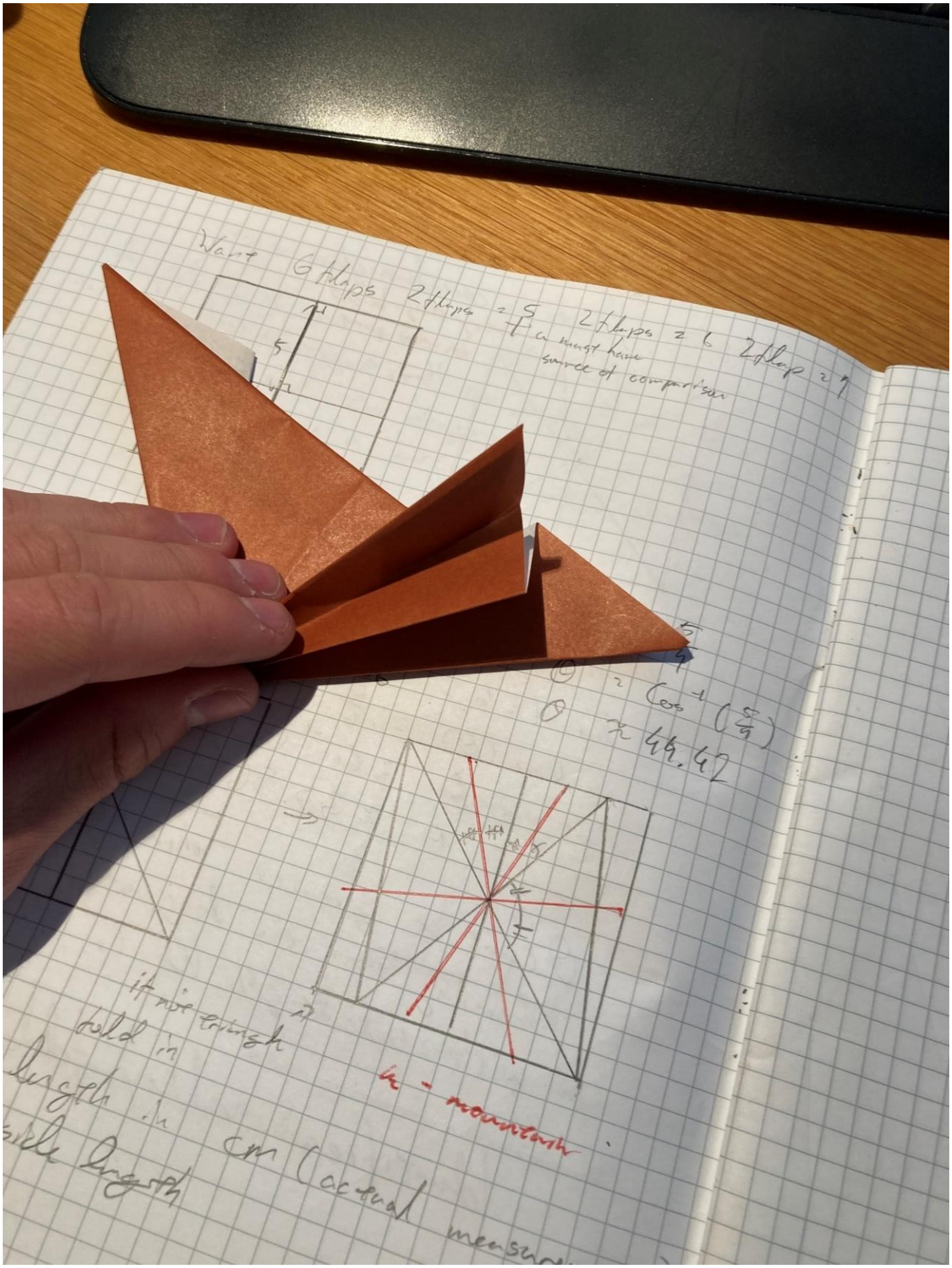
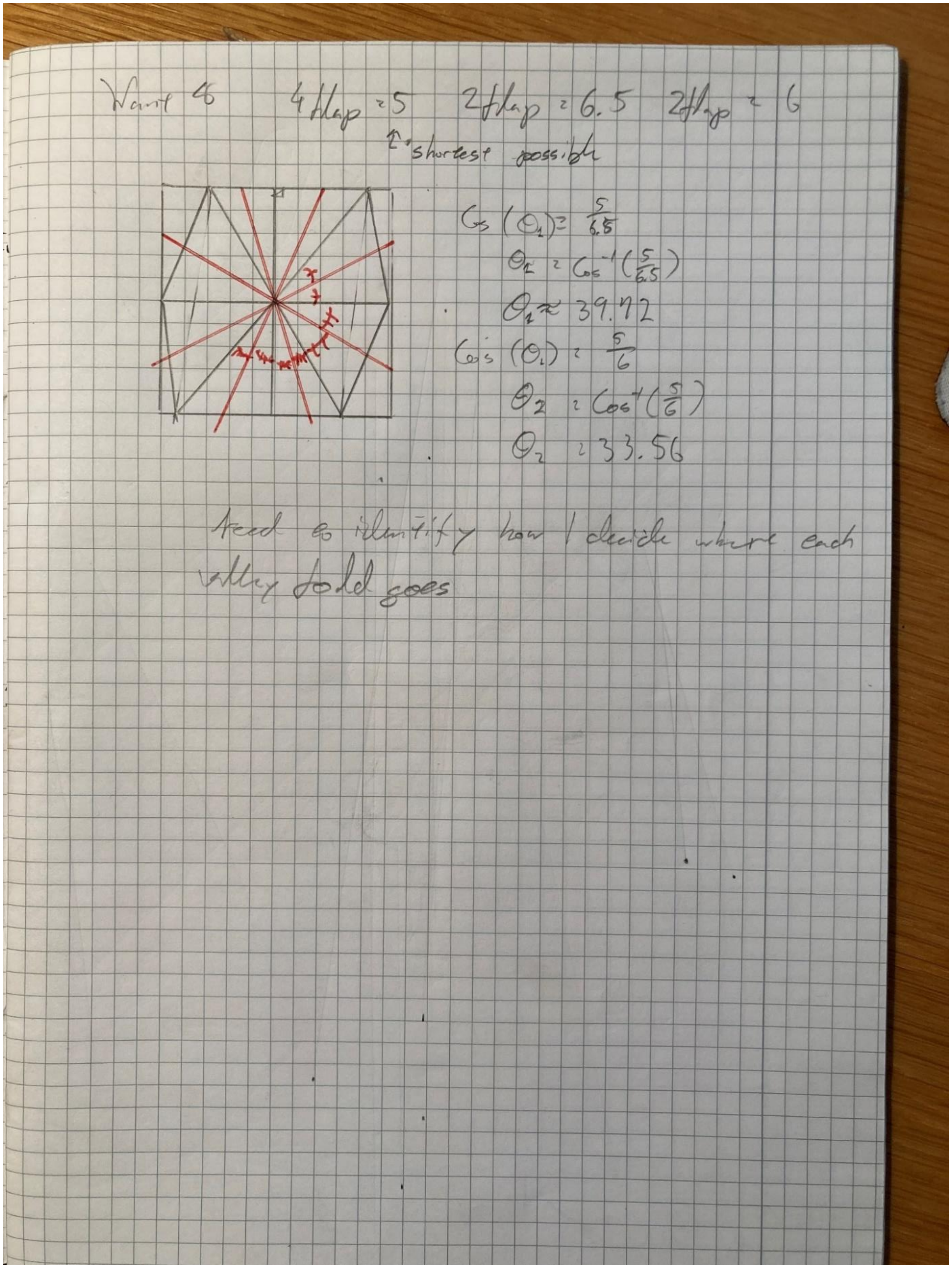


Figure 40. Six flap crease pattern folded



Want 8 4 flap = 5 2 flap = 6.5 2 flap = 6

↑ shortest possible

$\cos(\theta_1) = \frac{5}{6.5}$

$\theta_1 = \cos^{-1}\left(\frac{5}{6.5}\right)$

$\theta_1 \approx 39.72$

$\cos(\theta_1) = \frac{5}{6}$

$\theta_2 = \cos^{-1}\left(\frac{5}{6}\right)$

$\theta_2 = 33.56$

Need to identify how I decide where each valley fold goes

Figure 41. 8 flap crease pattern

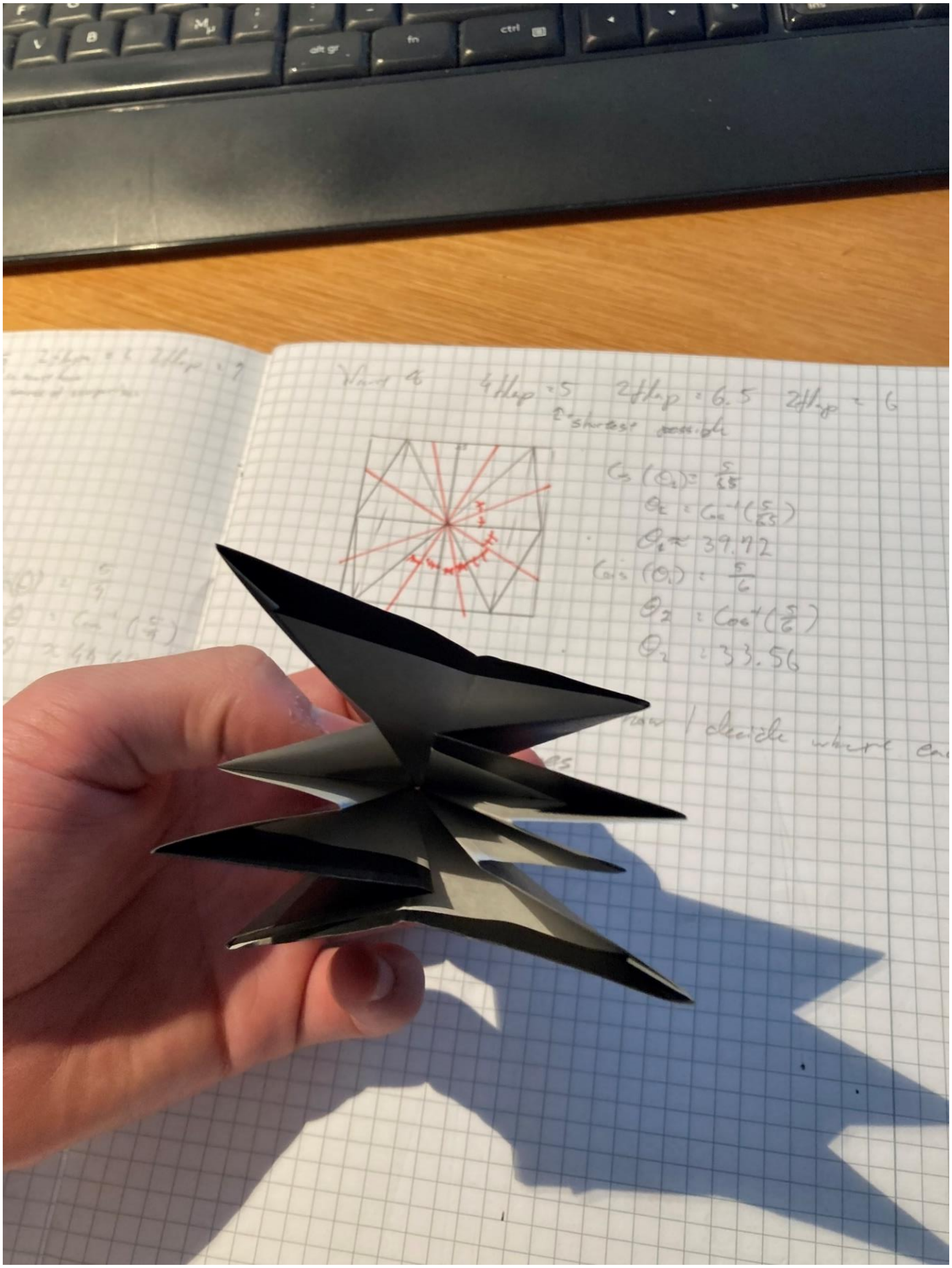
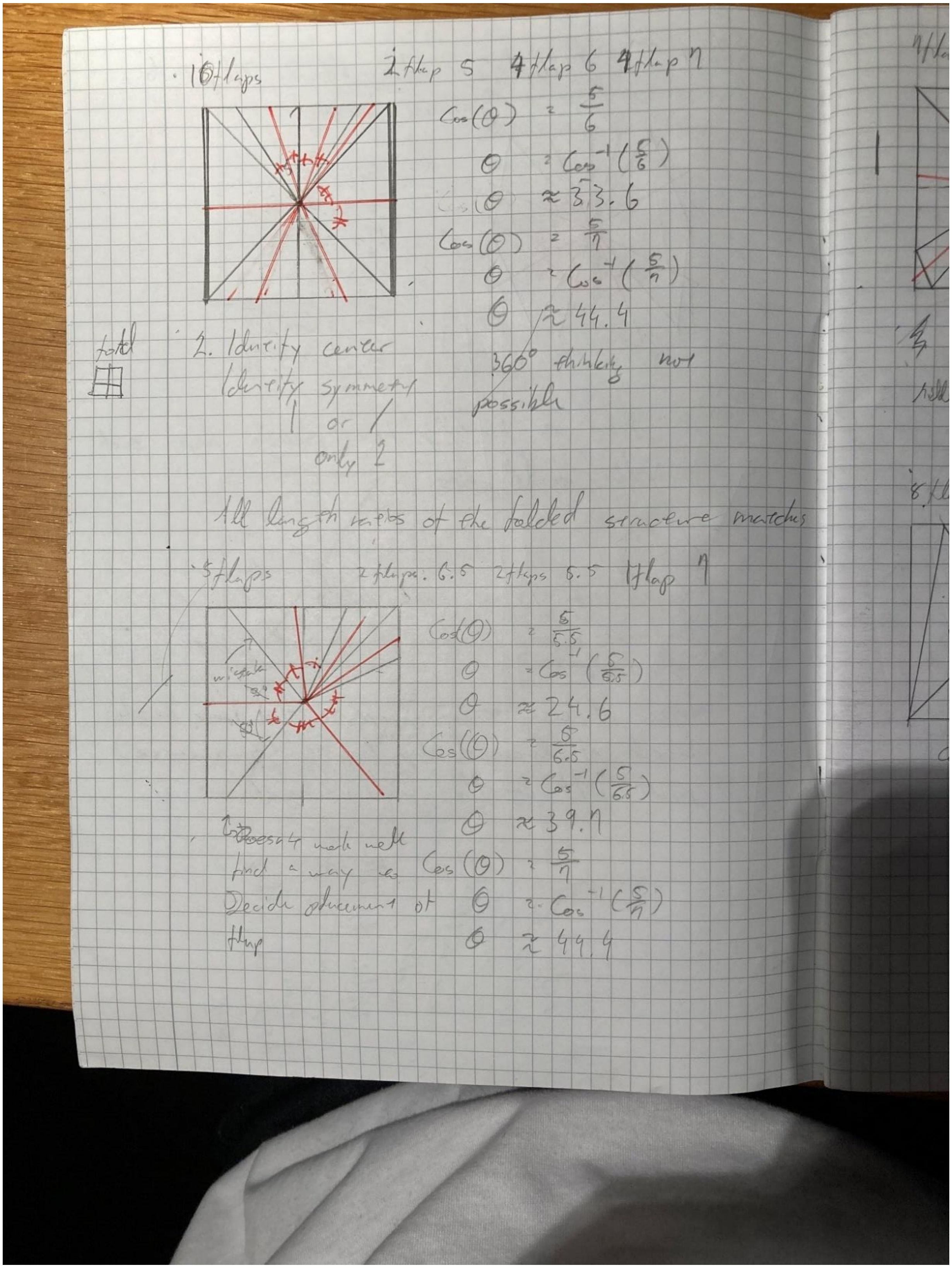


Figure 42. 8 flap crease pattern folded



· 10 flaps 2 flap 5 4 flap 6 4 flap 7

$\cos(\theta) = \frac{5}{6}$

$\theta = \cos^{-1}\left(\frac{5}{6}\right)$

$\theta \approx 33.6$

$\cos(\theta) = \frac{5}{7}$

$\theta = \cos^{-1}\left(\frac{5}{7}\right)$

$\theta \approx 44.4$

360° thinking not possible

fold

2. Identify center
Identify symmetry
| or /
only 2

All length ratios of the folded structure matches

5 flaps 2 flaps: 6.5 2 flaps 6.5 1 flap 7

$\cos(\theta) = \frac{5}{5.5}$

$\theta = \cos^{-1}\left(\frac{5}{5.5}\right)$

$\theta \approx 24.6$

$\cos(\theta) = \frac{5}{6.5}$

$\theta = \cos^{-1}\left(\frac{5}{6.5}\right)$

$\theta \approx 39.7$

$\cos(\theta) = \frac{5}{7}$

$\theta = \cos^{-1}\left(\frac{5}{7}\right)$

$\theta \approx 44.4$

Doesn't work well
find a way to
Decide placement of
flap

Figure 43. 10 flap crease pattern and 5 flap crease pattern

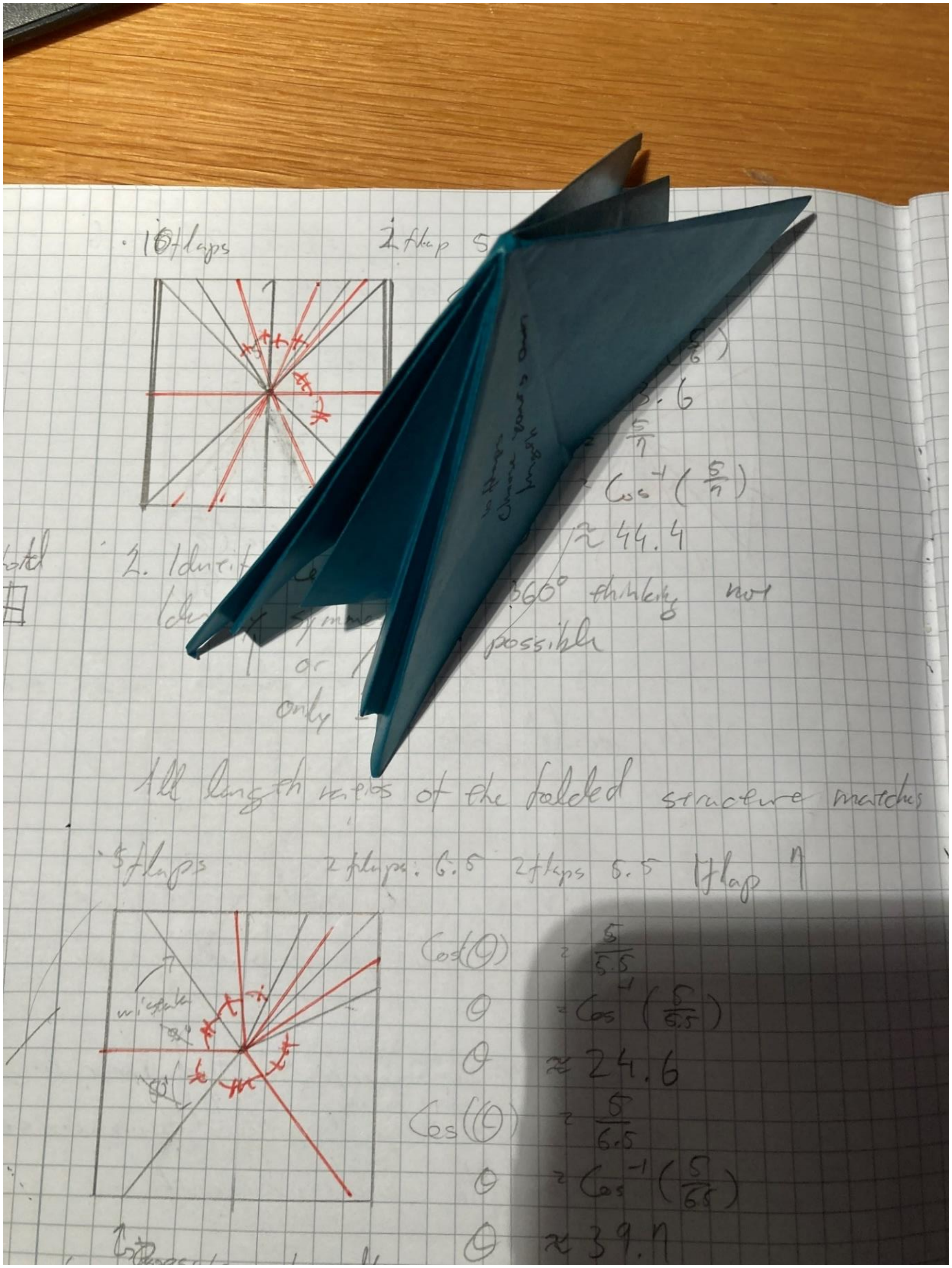


Figure 44. 10 flap crease pattern folded

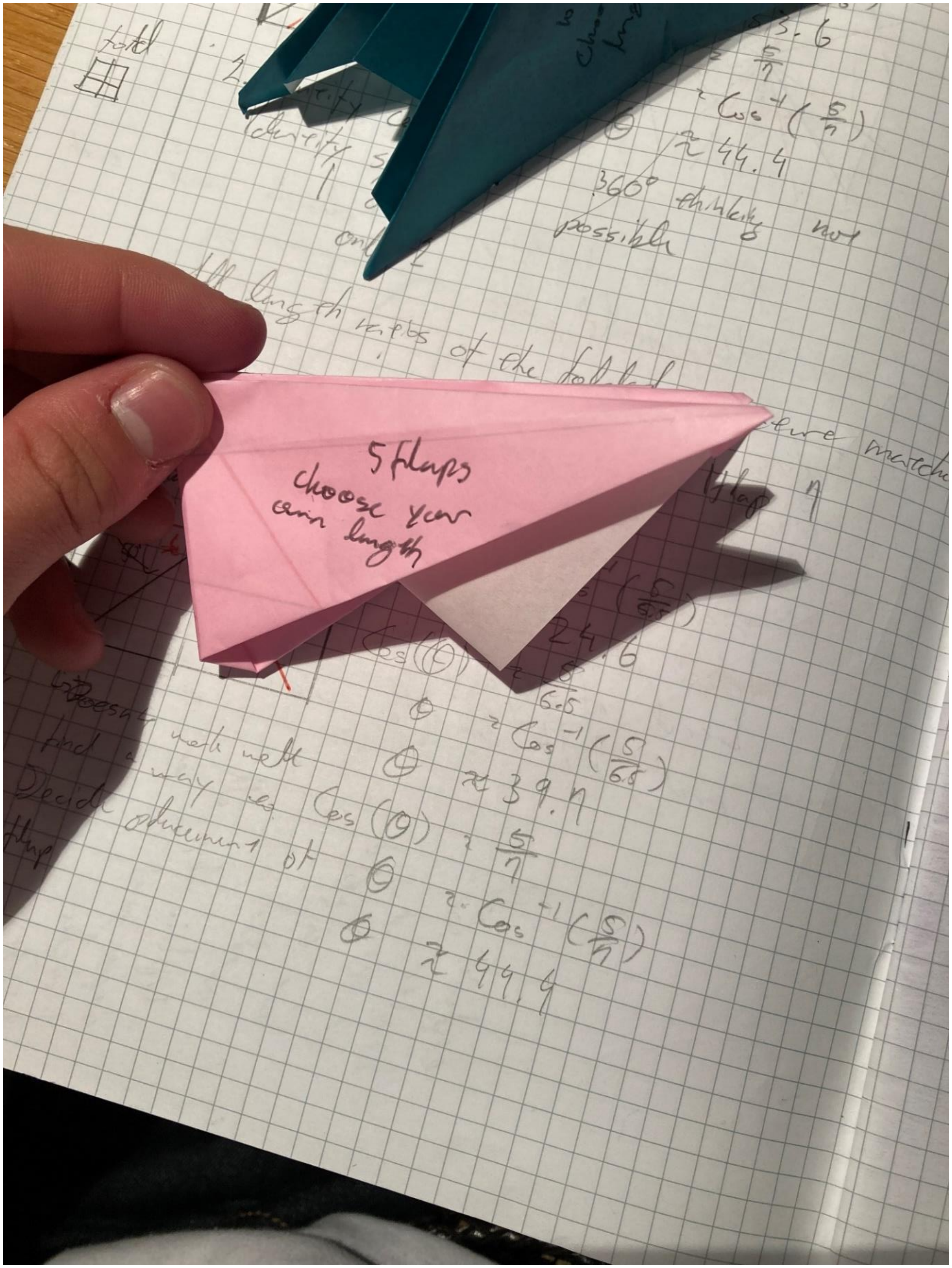
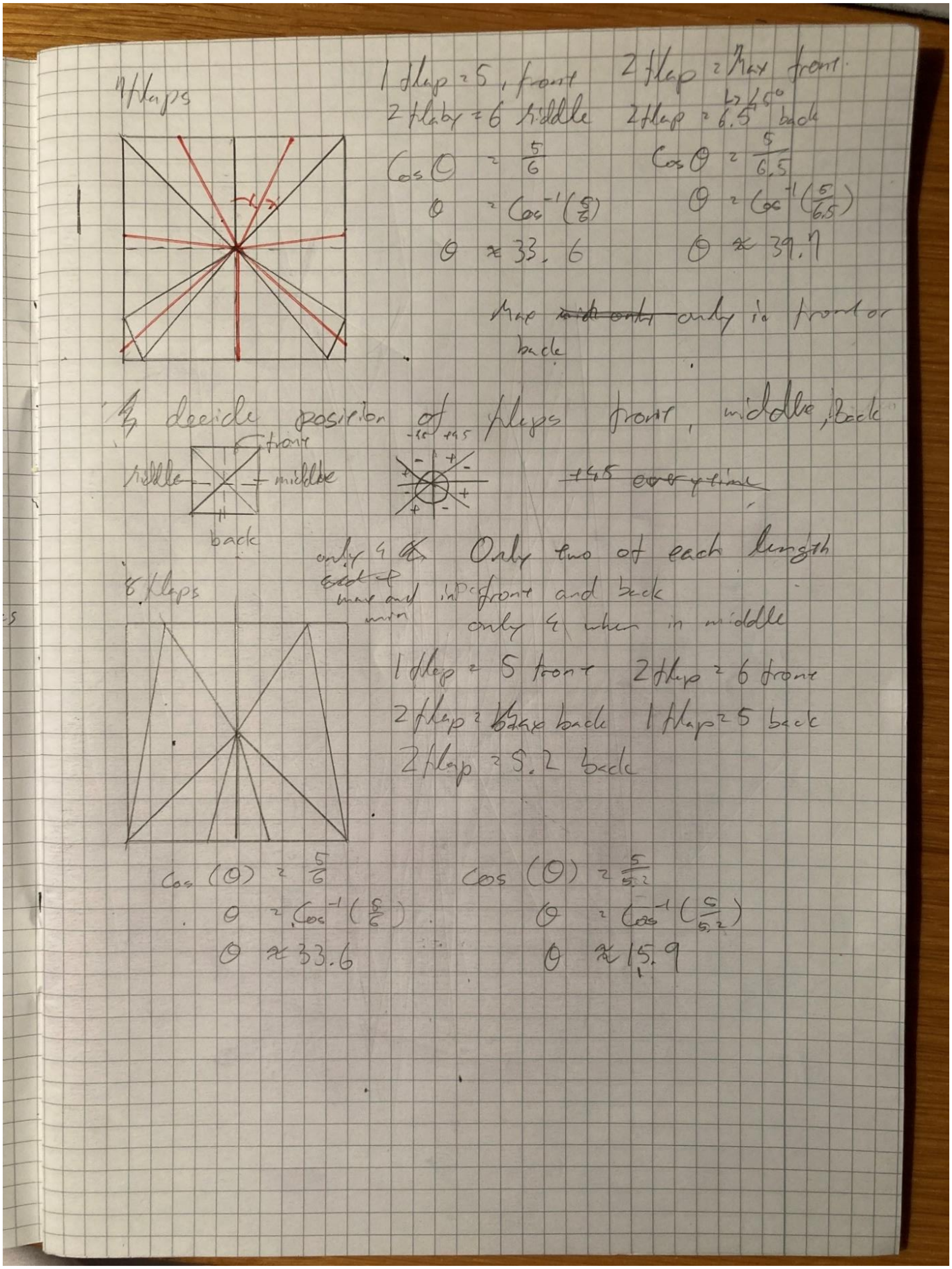

5 flaps
choose your
arm length
360° thinking not
possible
≈ 44.4
$\cos^{-1}(\frac{5}{7})$
$\cos(\theta) \approx \frac{5}{6.5}$
$\theta \approx \cos^{-1}(\frac{5}{6.5})$
$\theta \approx 39.7$
$\cos(\theta) \approx \frac{5}{7}$
$\theta \approx \cos^{-1}(\frac{5}{7})$
$\theta \approx 44.4$

Figure 45. 5 flap crease pattern folded



7 flaps

1 flap = 5, front 2 flap = Max front

2 flaby = 6 middle 2 flap = 6.5 back ↳ 45°

$\cos\theta = \frac{5}{6}$ $\cos\theta = \frac{5}{6.5}$

$\theta = \cos^{-1}(\frac{5}{6})$ $\theta = \cos^{-1}(\frac{5}{6.5})$

$\theta \approx 33.6$ $\theta \approx 39.7$

Max ~~with only~~ only in front or back

↳ decide position of flaps front, middle, Back

~~+45 every time~~

only 4 ~~&~~ ~~each of~~ max and min

↳ Only two of each length in front and back

only 4 when in middle

8 flaps

1 flap = 5 front 2 flaps = 6 front

2 flap = ~~6~~ Max back 1 flap = 5 back

2 flap = 5.2 back

$\cos(\theta) = \frac{5}{6}$ $\cos(\theta) = \frac{5}{5.2}$

$\theta = \cos^{-1}(\frac{5}{6})$ $\theta = \cos^{-1}(\frac{5}{5.2})$

$\theta \approx 33.6$ $\theta \approx 15.9$

Figure 46. 7 flaps and 8 flaps crease pattern

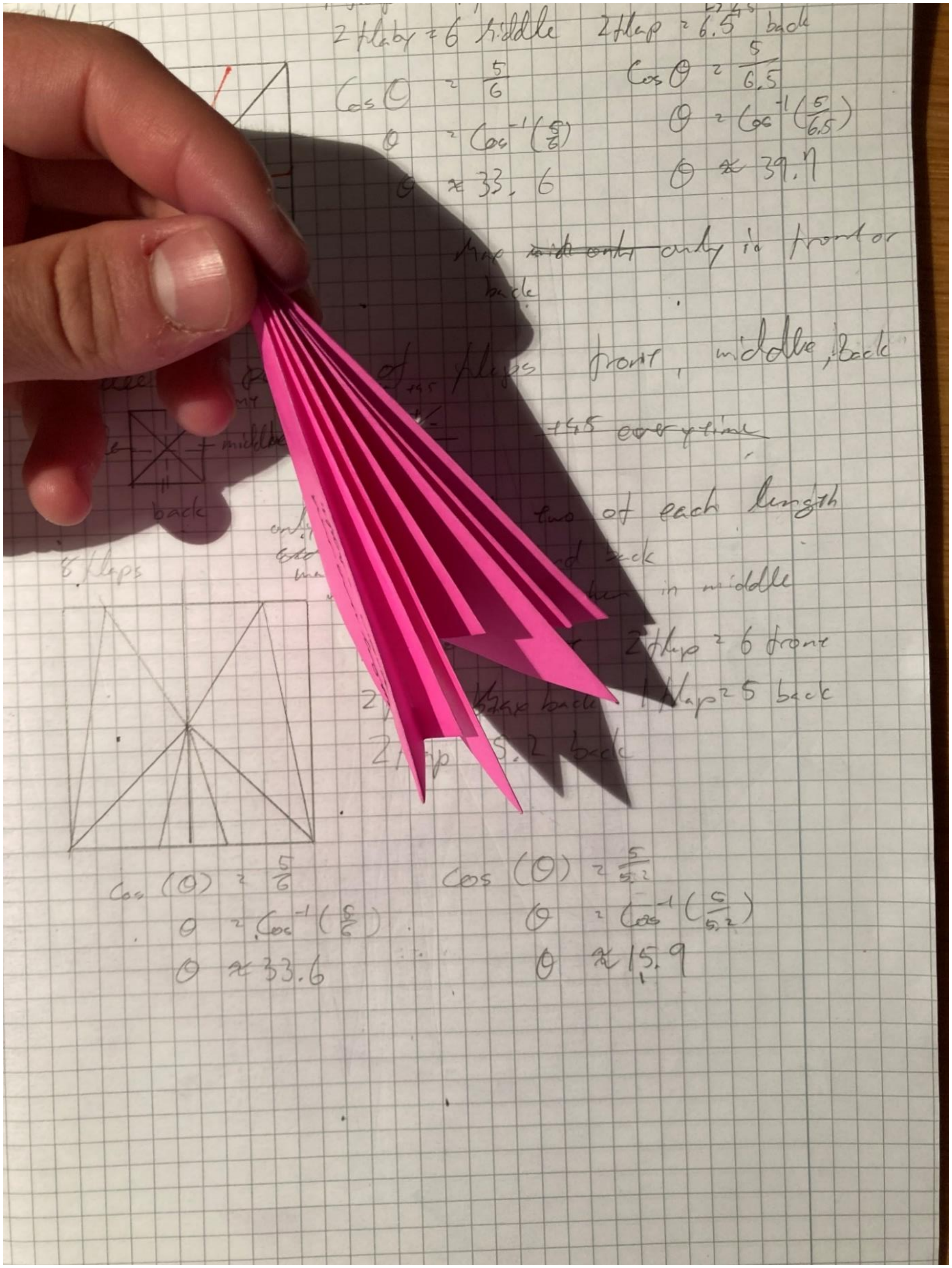


Figure 47. 7 flaps crease pattern folded

Appendix 5. The final product of the first stage, booklet (Sydnes, 2022)

How to Create New Origami Designs

Toma Sydnes

Introduction

Through this step-by-step guide you will be able to learn how to create new **origami bases** which will help you create your own origami design. What is special about my method is that it can all be performed by anyone who knows basic trigonometry, and is able to measure angles and lengths. Be aware over the cases where the method does not work (all of them have been listed in the section "limitations"). This step-by-step guide do not inform how to **shape** the **origami base**, which is the artistic and most time-consuming part of the creation of new origami designs.

The greatest difference between the method I have developed and the **circle-river packing** which is the most common and most successful method is that mine can be performed by solving equations, which **circle-river packing** cannot do. **Circle-river packing** specializes in optimizing the space which has made the process become an NP-hard, meaning that it can not be solved spontaneously with today's knowledge, and only can be solved through trial and error. On the other hand, the **circle-river packing** does not have any limitations on number of **flaps**, length ratios, and can also create **closed-ended flaps.** The **circle-river method** can be performed by defining the length of the **open-ended flaps** as the radius of the **circles** and the length of the **closed-ended flaps** as the width of the **river**. Continuing you will expand all of them at the same rate as much as possible so that the position compared to each other do not change and so that the centre of the circle is consistently on the **paper**.

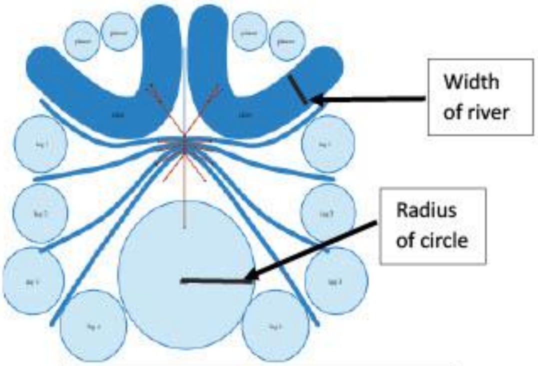


Figure 1., example of radius of circle and width of river (Robert J. Lang, 2022)

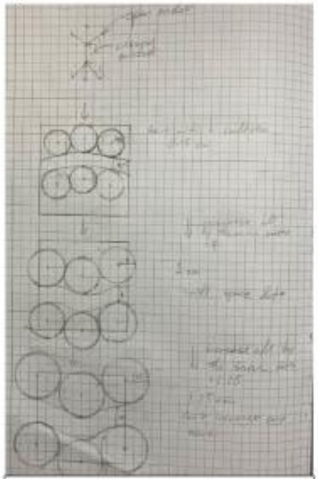


Figure 2., example of **circle-river** method in act

Vocabulary explanation

- *Origami base* – The folded paper with the minimum needed elements to create the final product
- Shaping – The process of adding in details to an **origami base**, for my method this includes changing **open-ended flaps** into **closed-ended flaps**
- Flap – a part of the paper which is partially disconnected from the rest of the paper

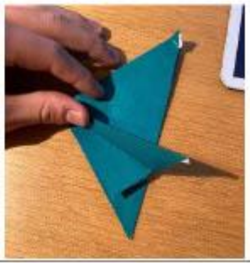


Figure 3., meaning of flaps

- Closed-ended flaps – flaps which are connected to other flaps on both sides

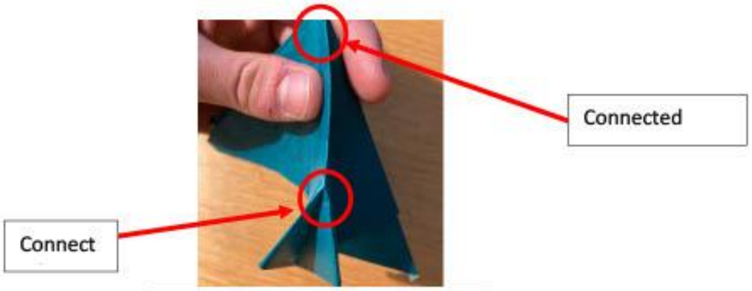


Figure 4., meaning of closed-ended flaps

- Open-ended flaps – flaps which are connected to other flaps only on one side

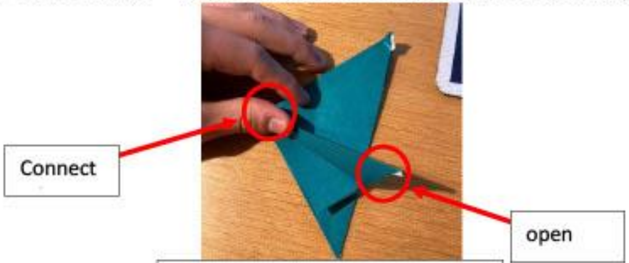


Figure 5., meaning of open-ended flaps

- Circle in circle-river packing – Representational figure of **open-ended flaps** in circle-river packing
- River in circle-river packing – Representational figure of **closed-ended flaps** in circle-river packing optimizing

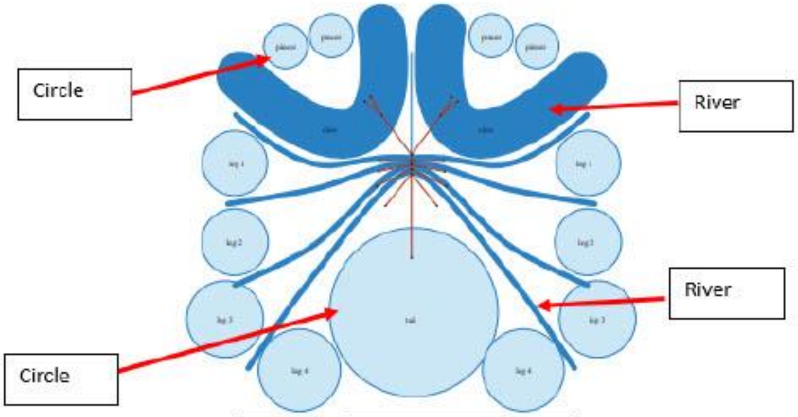


Figure 6 meaning of circle and river in circle-river packing (Robert J. Lang, 2009)

- Paper – the square paper which will be folded

- Perpendicular line – a line which is perpendicular to the centre of the side (red line on Figure 7)

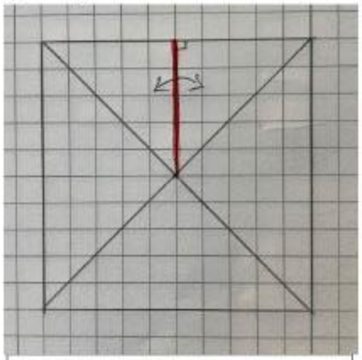


Figure 7., meaning of perpendicular line

- Position – where the flaps' positions are defined by front, back and middle which are all parts of the square when it is divided into 4 parts by two diagonal lines and each of them has their own **perpendicular line**

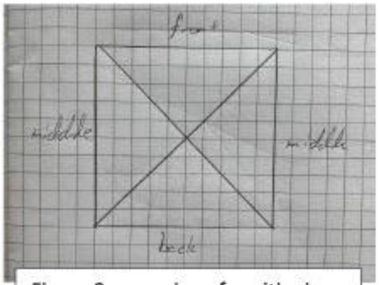


Figure 8., meaning of positioning terms

- Connectivity – how well-connected flaps are. The higher the connectivity the less you can do when shaping the origami base. It is determined by the distance between the folds and the ratio to the size of the flap
- Crease lines – a line which shows where the fold is supposed to be

- Valley fold – a fold which is folded inwards

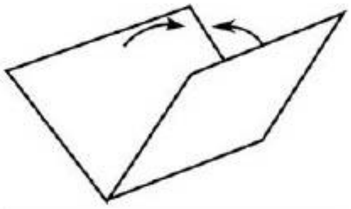


Figure 9., meaning of valley fold (Origami-instructions.com, 2014)

- Mountain fold – a fold which is folded outwards

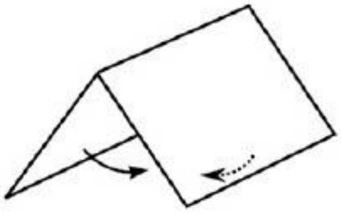


Figure 10., meaning of mountain fold (Origami-nstructions.com,2014)

Limitations

- The method does not work if the **paper** is not a square
- The method does not work when the length of a **flap** is less than half of the side length of the **paper**
- The method does not work when the length of a **flap** is greater than $\sqrt{half\ of\ side\ length^2 + half\ of\ side\ length^2}$
- The method does not work when the number of **flaps** with equal lengths exceeds 8
- The method does not work when the number of **flaps** with the maximum length exceeds 4
- The method does not work when the number of **flaps** with the minimum length exceeds 4
- The method can not create **closed-ended flaps**
- All crease lines must merge at the centre of the square paper

5

Method to create the crease pattern for the origami base

1. Choose the number of **flaps**, its length and **positioning (front, middle or back)** (see section for limitations for guidance)
    a. When deciding number of **flaps** you should draw a stick figure of the structure you would like to create, also known as a tree
        i. The number of edges in the tree is the number of flaps needed
    b. When deciding the lengths of the **flaps** you should draw a stick figure of the structure and decide on the length ratios between the flaps
    c. When deciding the **positioning** of the **flaps** keep in mind the **flap's** length and its **connectivity**
2. Plot the wanted length (w) and side length of the **paper** (l) into the equation below to get the angle between the **perpendicular line** and the **crease line** of wanted **flap**. It can be on any side of the **perpendicular line**

$$cos^{-1}\left(\frac{l}{2w}\right) = \theta$$

    a. The angle of maximum length is always 45° and minimum length is always 0°
    b. All these crease lines are **valley folds**
    c. The smaller the angle between the **crease lines** of the flaps the higher the **connectivity**
    d. All crease lines run from the centre to the edge of the paper
    e. The angles will be measured from the **perpendicular line** and can be on either side of the line, where you can choose where to place it
    f. It is recommended to draw these lines on the **paper** or on a different piece of paper before folding

3. If the smallest unit of polygon created by the **crease lines** and the edge of the paper is a non-triangular polygon (see figure 11), then draw a new **crease line** between the points where the **crease lines** forming the sides of the polygon meet the edge of the **paper** to form a triangle as (see figure 12). This process is repeated till there are no non-triangular polygon's left as the smallest unit of polygon

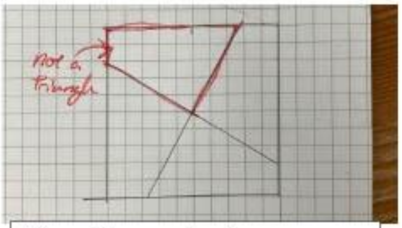


Figure 11., example of non-triangular polygon in step 3

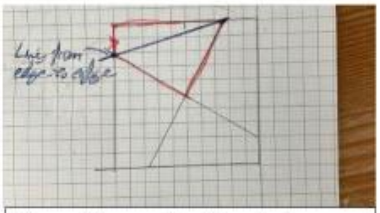


Figure 12., creating triangles from non-triangular polygon

4. Draw **crease lines** in between the **crease lines** which form the **flaps**. This will divide the angles in between two equal parts. The newly created **crease lines** are all **mountain folds**

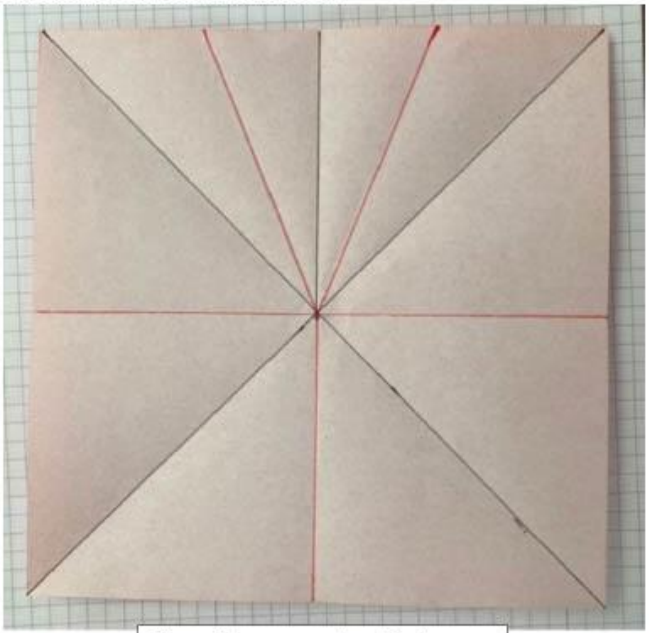


Figure 13.,an example of final crease pattern (black = valley fold, red = mountain fold)

7

Method to fold the crease pattern of the origami base

1. Fold all the **crease lines** that are **valley folds** (folds which will become the **flaps**)

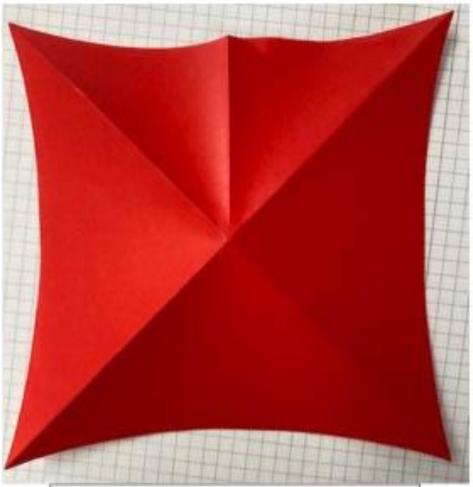
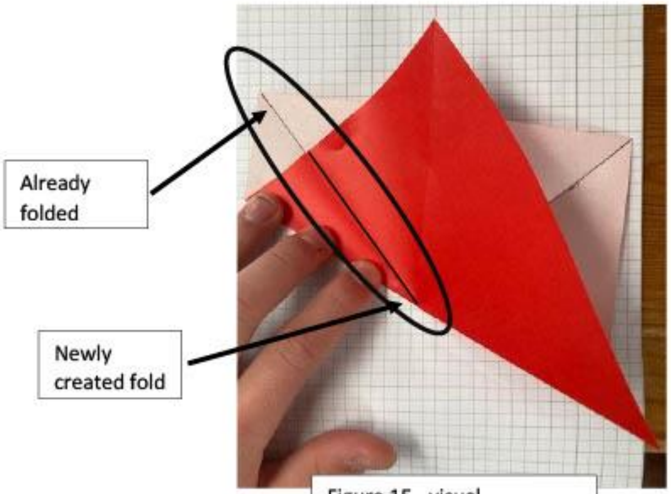


Figure 14, visual example of step 1 completed

2. Fold the already folded fold, on top of the neighbouring fold to create a **valley fold** between the two, dividing the angle in two



Already folded

Newly created fold

Figure 15., visual explanation of step 2

a. After completing step 1 and step 2 the paper will be folded in a similar way to bellow. Only having valley folds

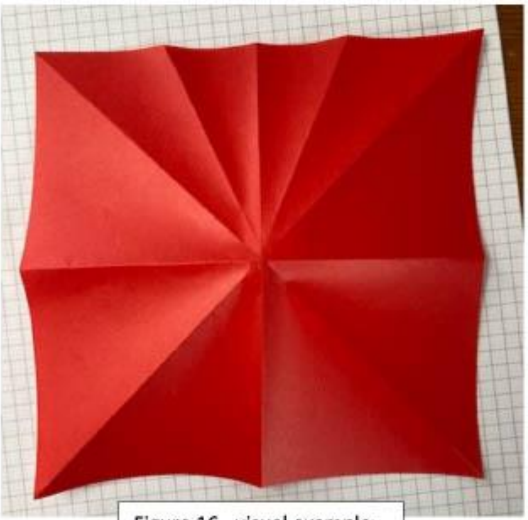


Figure 16., visual example of step 1 and 2 completed

3. Construct the **origami base** by having the folds which creates the flaps be **valley folds** and the rest as **mountain folds**

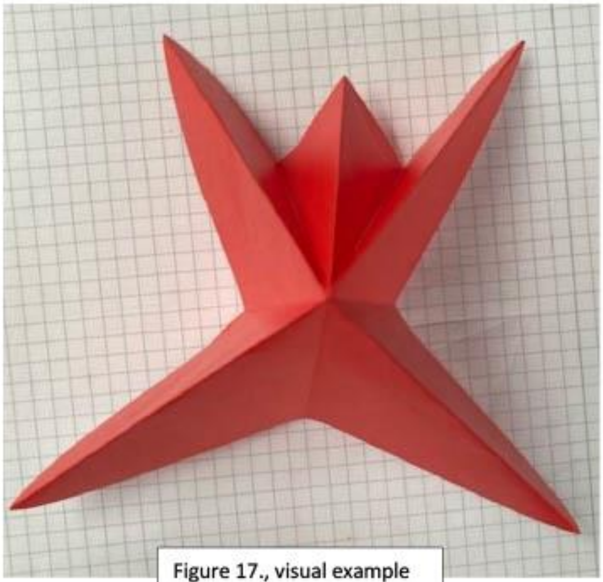
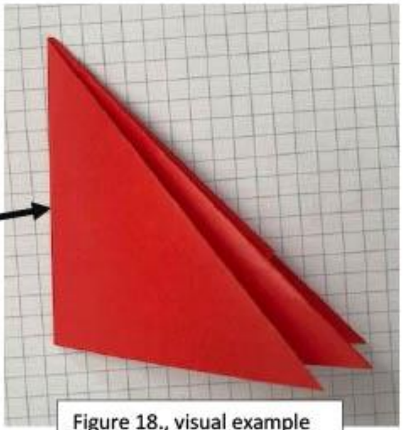


Figure 17., visual example of step 3

9

4. When folding the origami base flat, you will need to change one of the **mountain folds** into a **valley fold** or get rid of two **mountain folds**
    a. This must be done according to Maekawa's Theorem, which states that "for an origami to be able to fold flat the number of **mountain folds** and **valley folds** must differ by two" (Wikipedia, 2022) known as flat foldability



Original **valley fold** is changed to a **mountain fold**

Figure 18., visual example of step 4

5. If all the steps above are performed, you have constructed the wanted **origami base** and it is ready for **shaping**

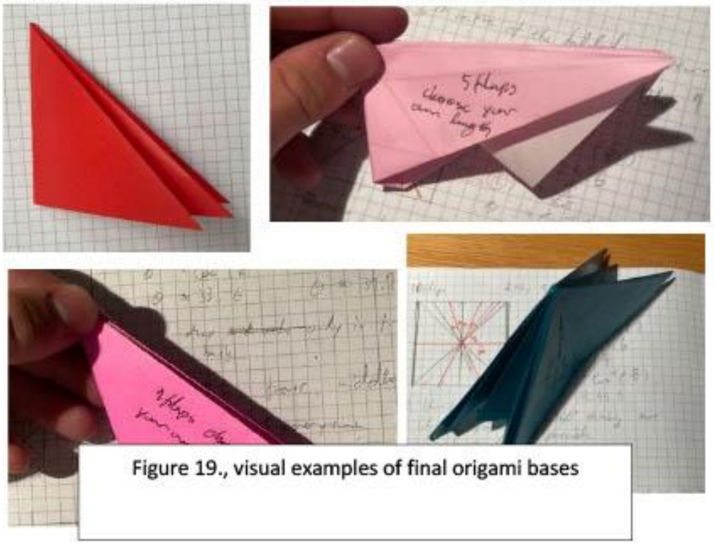


Figure 19., visual examples of final origami bases

Techniques of shaping

When **shaping** there are many different techniques you can use. Here you will find a few examples of the simplest **shaping** folds. Remember that **shaping** is the most time-consuming and difficult part of creating a new origami design, but also the part where you will be able to use your creativity and artistic mind the most.

1. **Valley fold**
2. **Mountain fold**
3. Squash fold
    a. A technique where you unfold a **flap** and refold it by squashing it. During the process, you will recreate the flap by widening and moving the position of the **mountain fold** creating the flaps thus decreasing its connectivity. In addition, you will still be able to keep the **flap** length you originally had

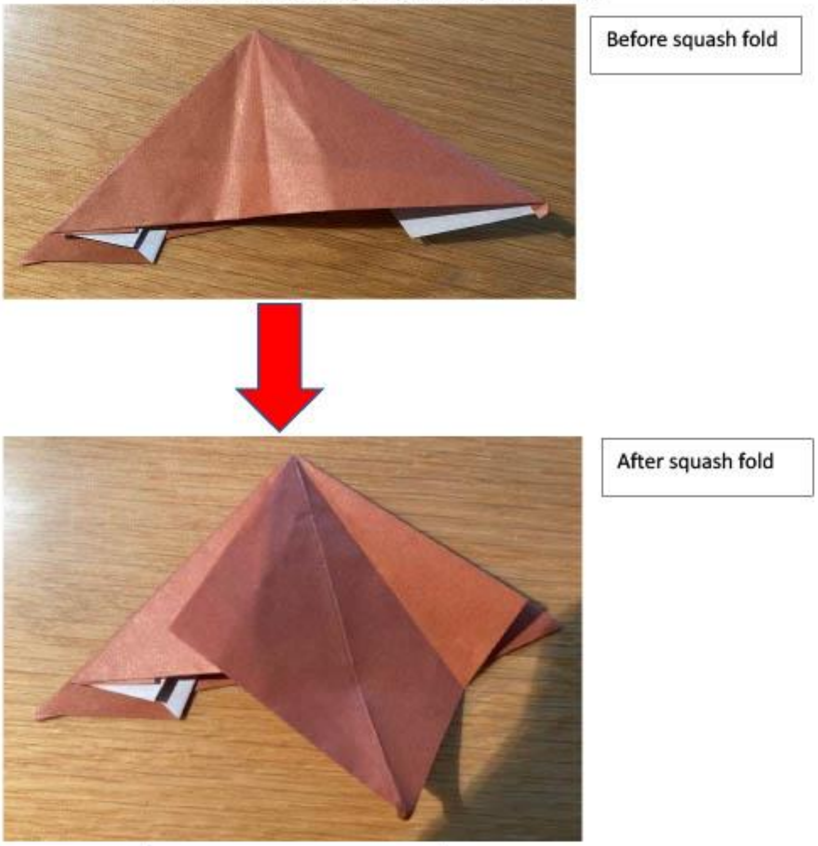
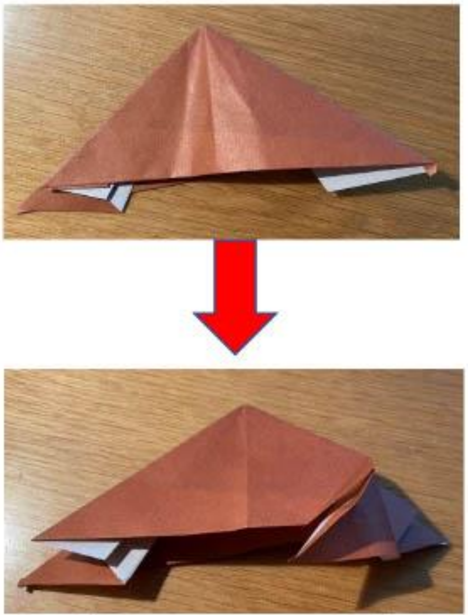


Figure 20., visual example of squash fold

11

4. Reverse fold
    a. Inside reverse fold
        i. Takes the outside edge of the **open-ended flap** and fold it inward. During the process, it will change the **open-ended flap** into a **closed-ended flap** with 2 **open-ended flaps** on the edge missing the pointy edge

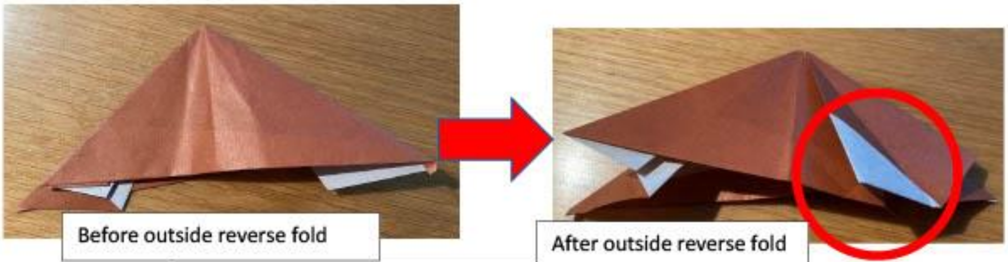


Before inside reverse fold

After inside reverse fold

Figure 21., visual example of inside reverse fold

    b. Outside reverse fold
        i. Takes the edge of the **open-ended flap** and removes the pointy edge from the **open-ended flap**

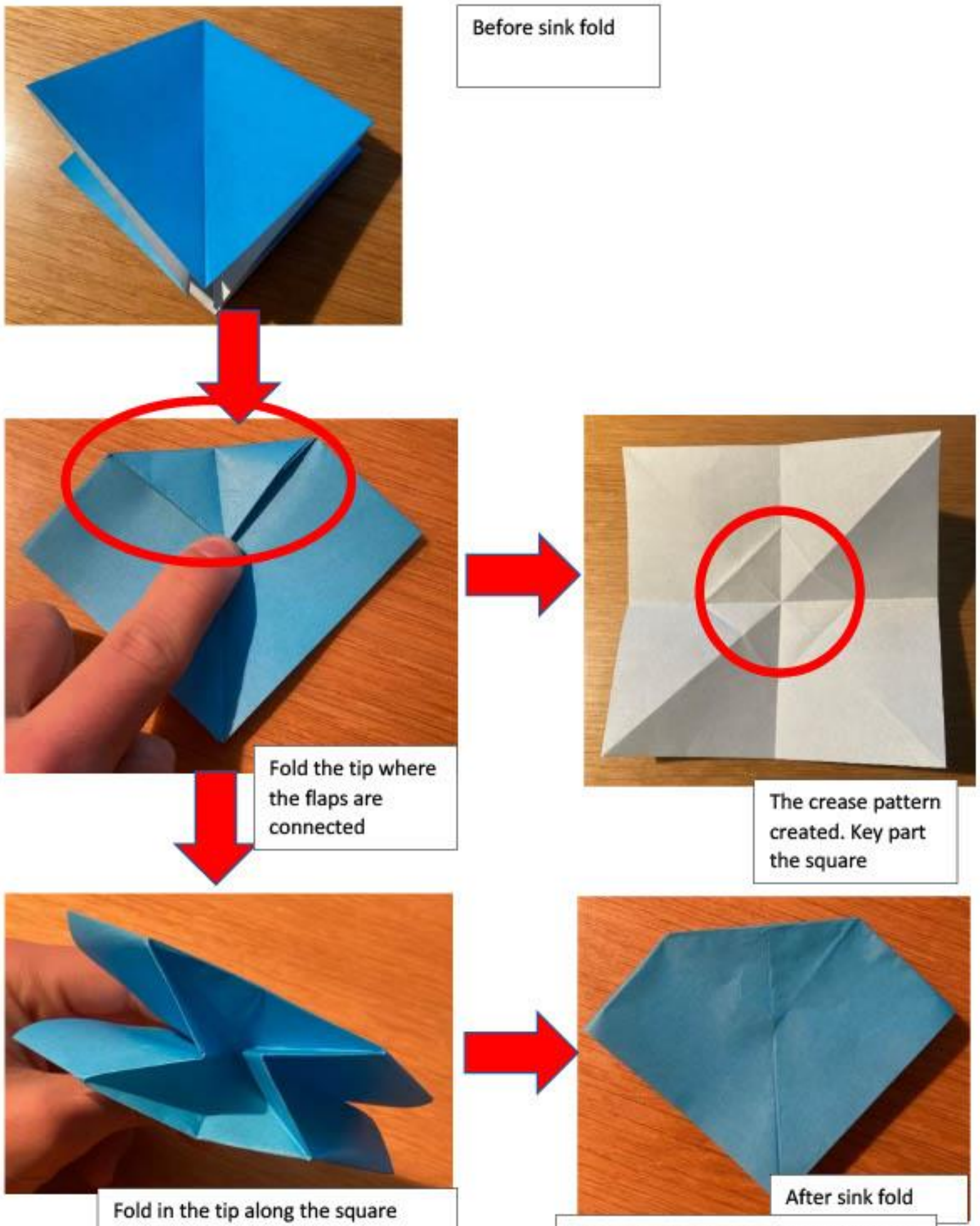


Before outside reverse fold

After outside reverse fold

Figure 22., visual example of outside reverse fold

5. Sink fold
    a. A fold which is the most difficult of the ones presented, but is useful to hideaway big parts of the paper which is connected to several flaps, while maintaining flaps connectivity



Before sink fold

Fold the tip where the flaps are connected

The crease pattern created. Key part the square

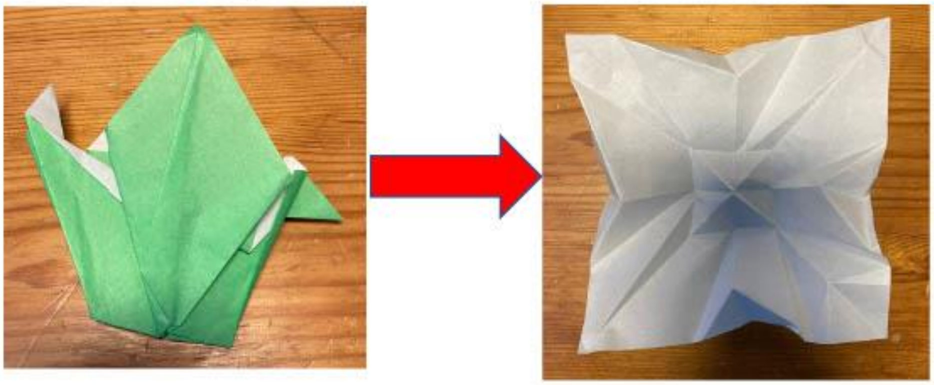
Fold in the tip along the square shaped crease pattern

After sink fold

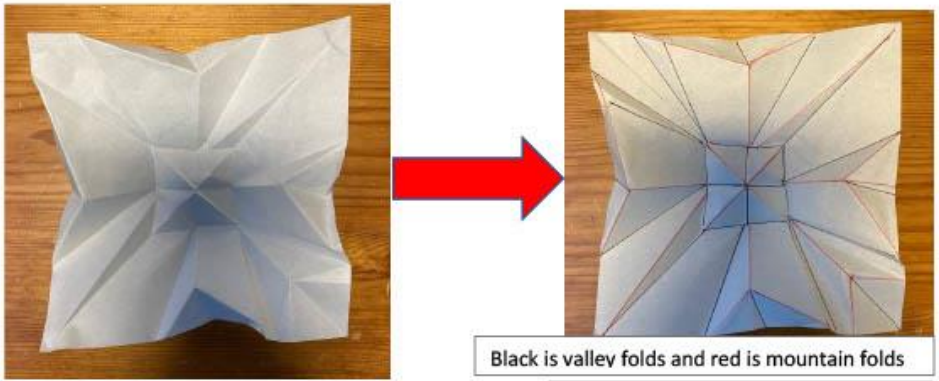
Figure 23., visual example of sink fold

How to get final crease pattern of the New Origami design

1. Un fold the origami structure



2. Draw lines through crease lines



Black is valley folds and red is mountain folds

Figure 24., visual example of how to get the final crease pattern

3. When all the steps are completed you get the final crease pattern

Last note

Whenever creating new origami designs you must be aware that it is a long process, taking many folds and unfolds (trials and errors) and it is crucial to be creative to come around the limitations of origami. I will recommend to start with simple designs with a small number of flaps with easily foldable angles.

References

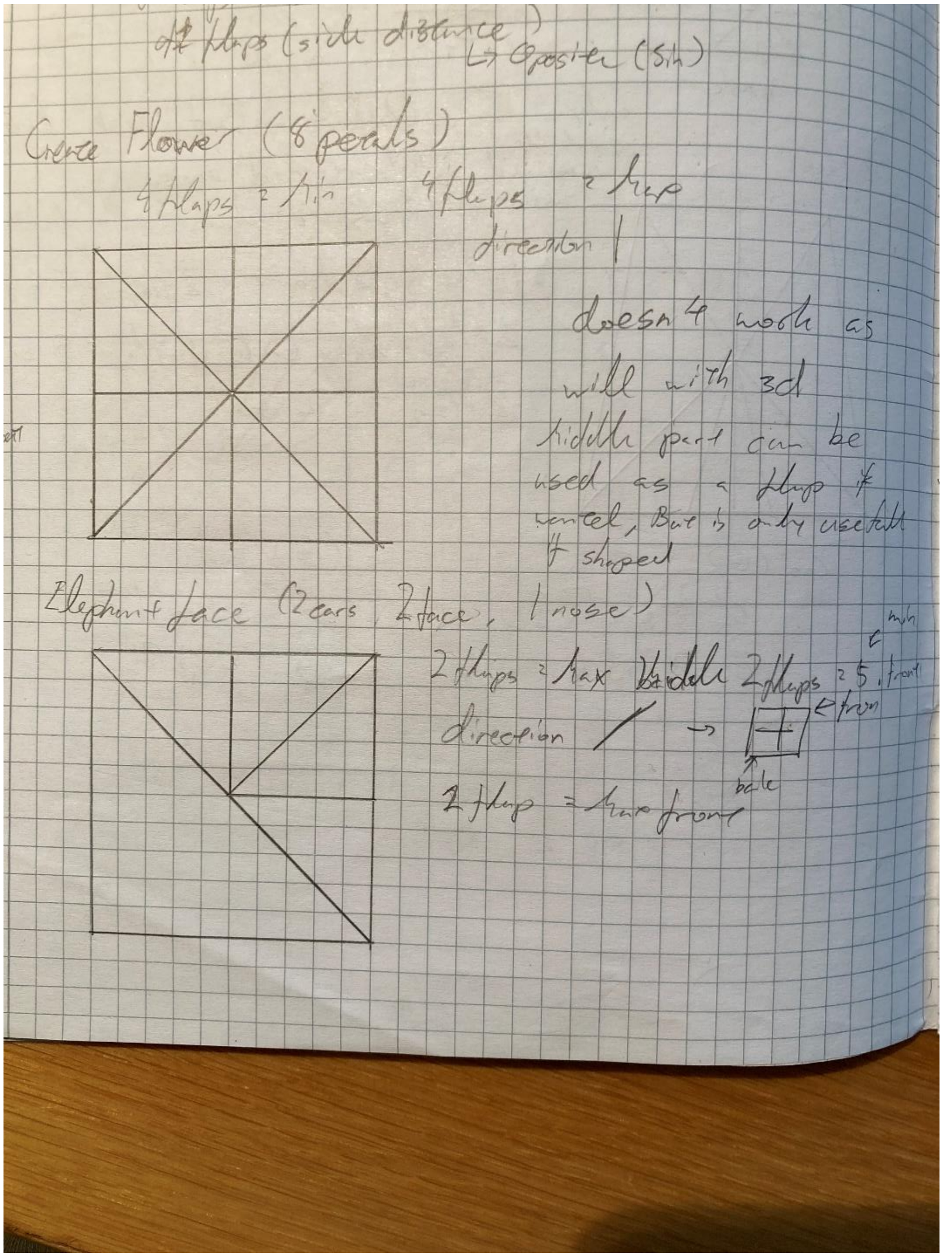
Origami-instructions.com, (2014). *Origami Valley and Mountain Folds - How to make Origami Valley and Mountain Folds*. [online] Available at: <http://www.origami-instructions.com/origami-valley-and-mountain-folds.html?no_redirect=true> [Accessed 4 September 2022].

Wikipedia (2022). *Maekawa's theorem*. [online] Available at: <https://en.wikipedia.org/wiki/Maekawa%27s_theorem> [Accessed 4 September 2022].

Robert J. Lang (2009). *Mathematical Methods in Origami Design*. [pdf] Available at: <http://m.archive.bridgesmathart.org/2009/bridges2009-11.pdf.> [Accessed 18. June 2022]

Scu.edu. (2022). *Types of Folds and Bases*. [online] Available at: <http://ecs.engr.scu.edu/studentpages/Wilcox/KMT2015/folds.html> [Accessed 25 Sep. 2022].

Appendix 6. Testing of the final product of Stage 1 with intended structure



4t flaps (side distance)
↳ Opositer (Sin)

Crease Flower (8 peals)

4 flaps = Min 4 flaps = Max

direction |

doesn't work as will with 3d
Middle part can be used as a flaps if wanted, But is only usefull if shaped

Elephant face (2 ears, 2 face, 1 nose)

2 flaps = Max Middle 2 flaps = 5, front
min

direction / →
← fron
back

2 flap = Max front

Figure 48. Crease pattern 8 petal flower and elephant-face

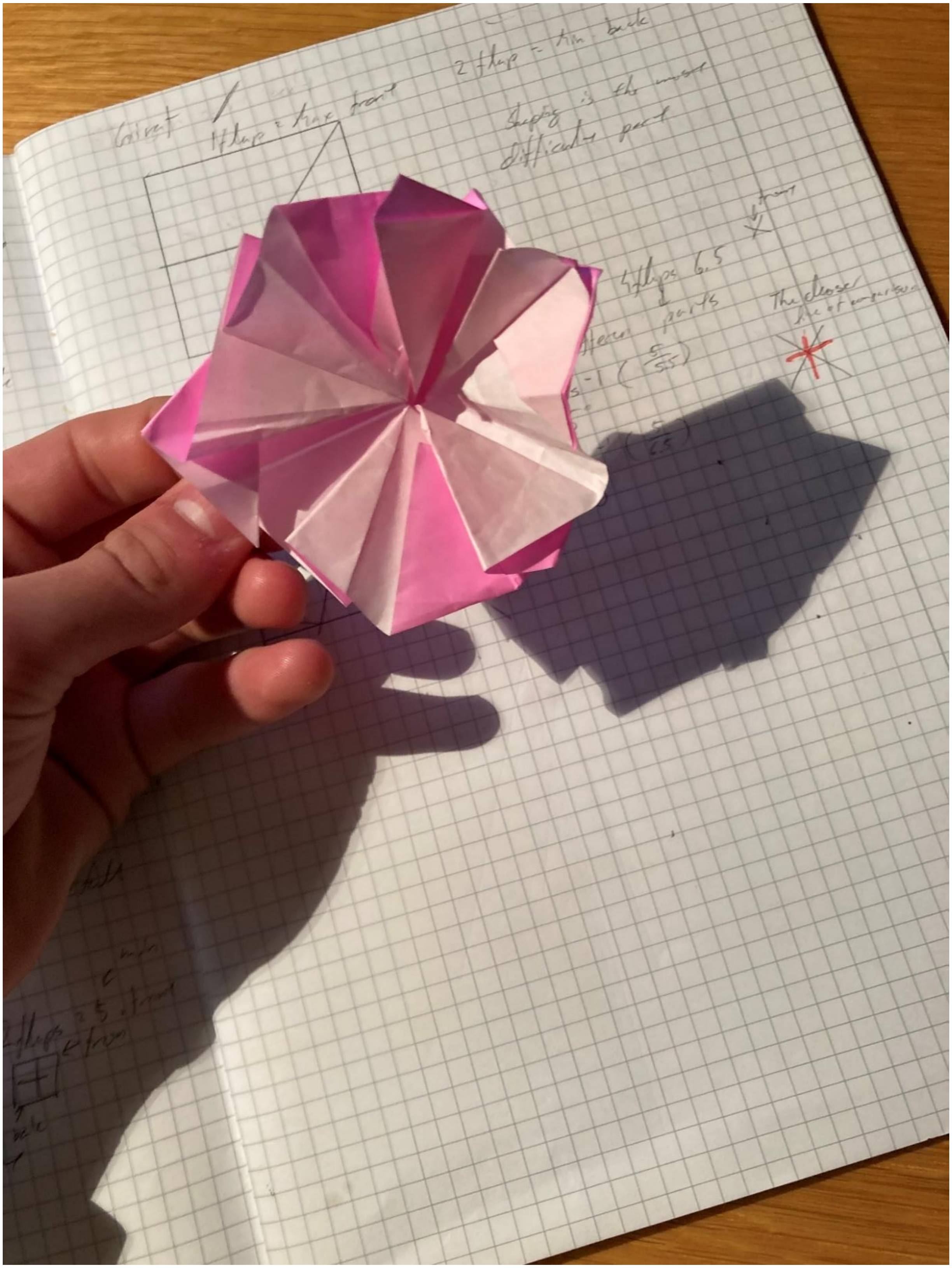


Figure 49. 8 petal flower crease pattern folded and shaped, success

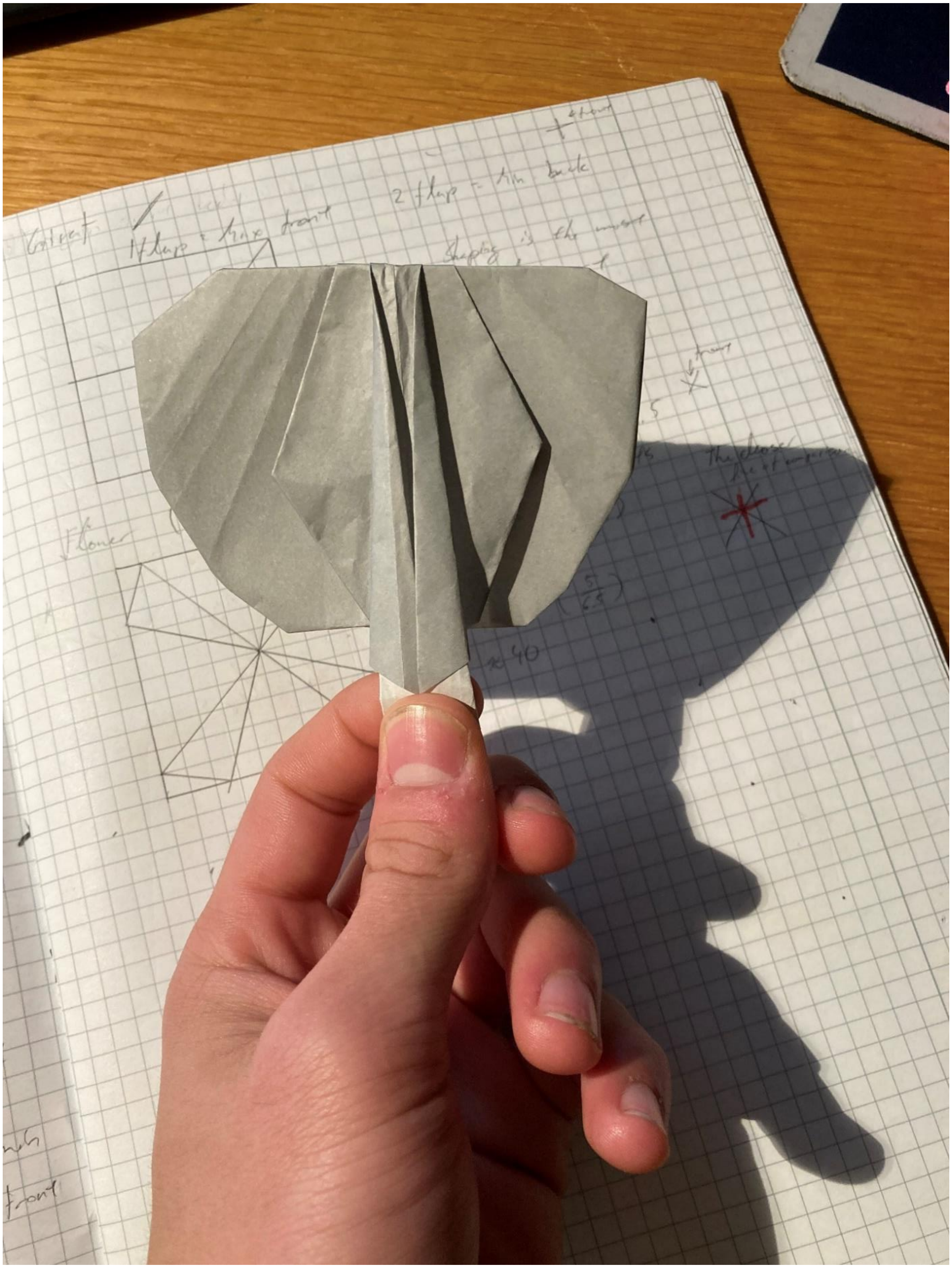
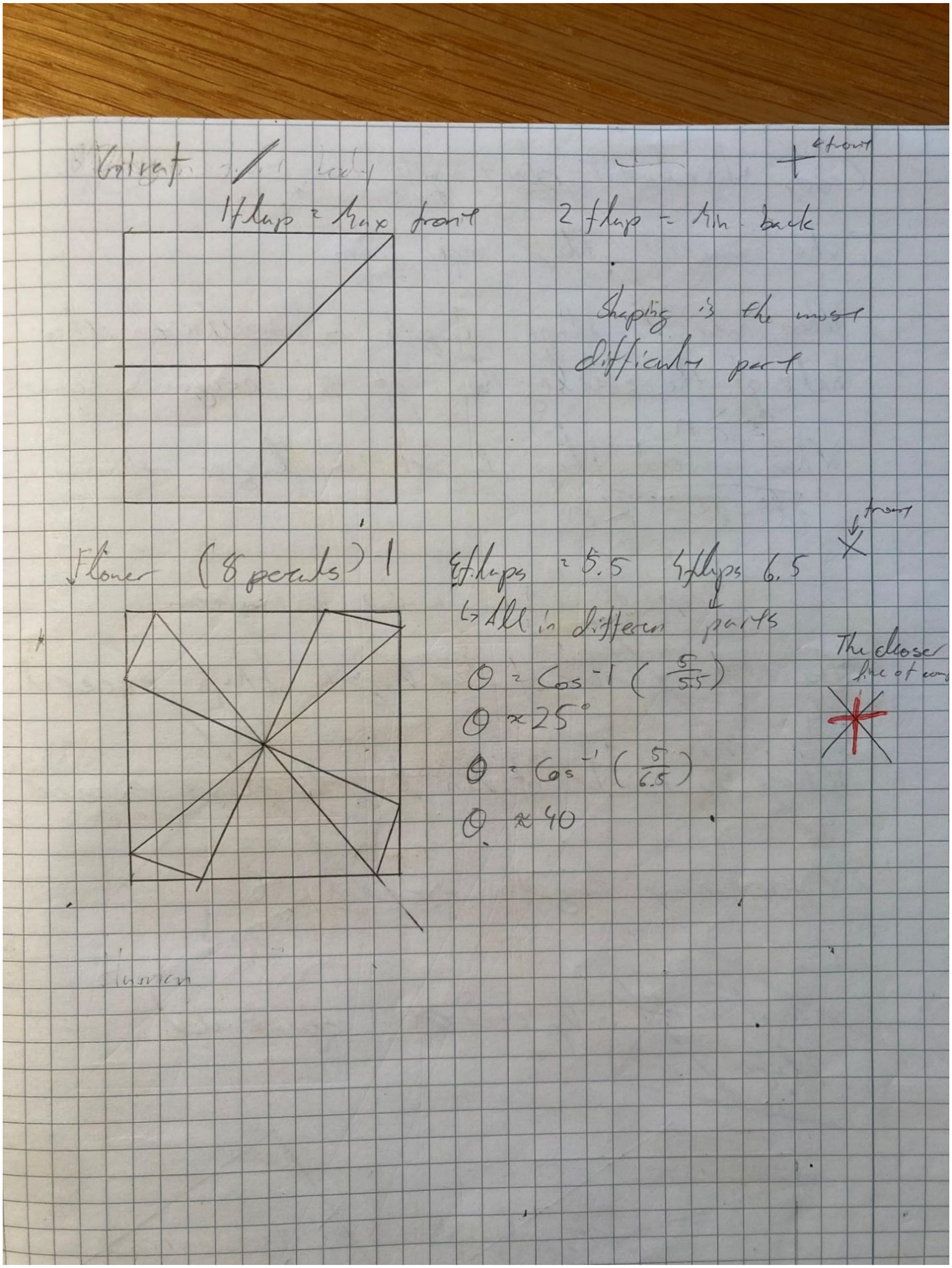


Figure 50. Elephant-face crease pattern folded and shaped, success



Shaping is the most difficult part

4 flaps = 5.5 4 flaps 6.5

↳ All in differen parts

$\theta = \cos^{-1}\left(\frac{5}{5.5}\right)$

$\theta \approx 25°$

$\theta = \cos^{-1}\left(\frac{5}{6.5}\right)$

$\theta \approx 40$

The closer line of com

Figure 51. Crease pattern for giraffe and a different 8 petal flower design

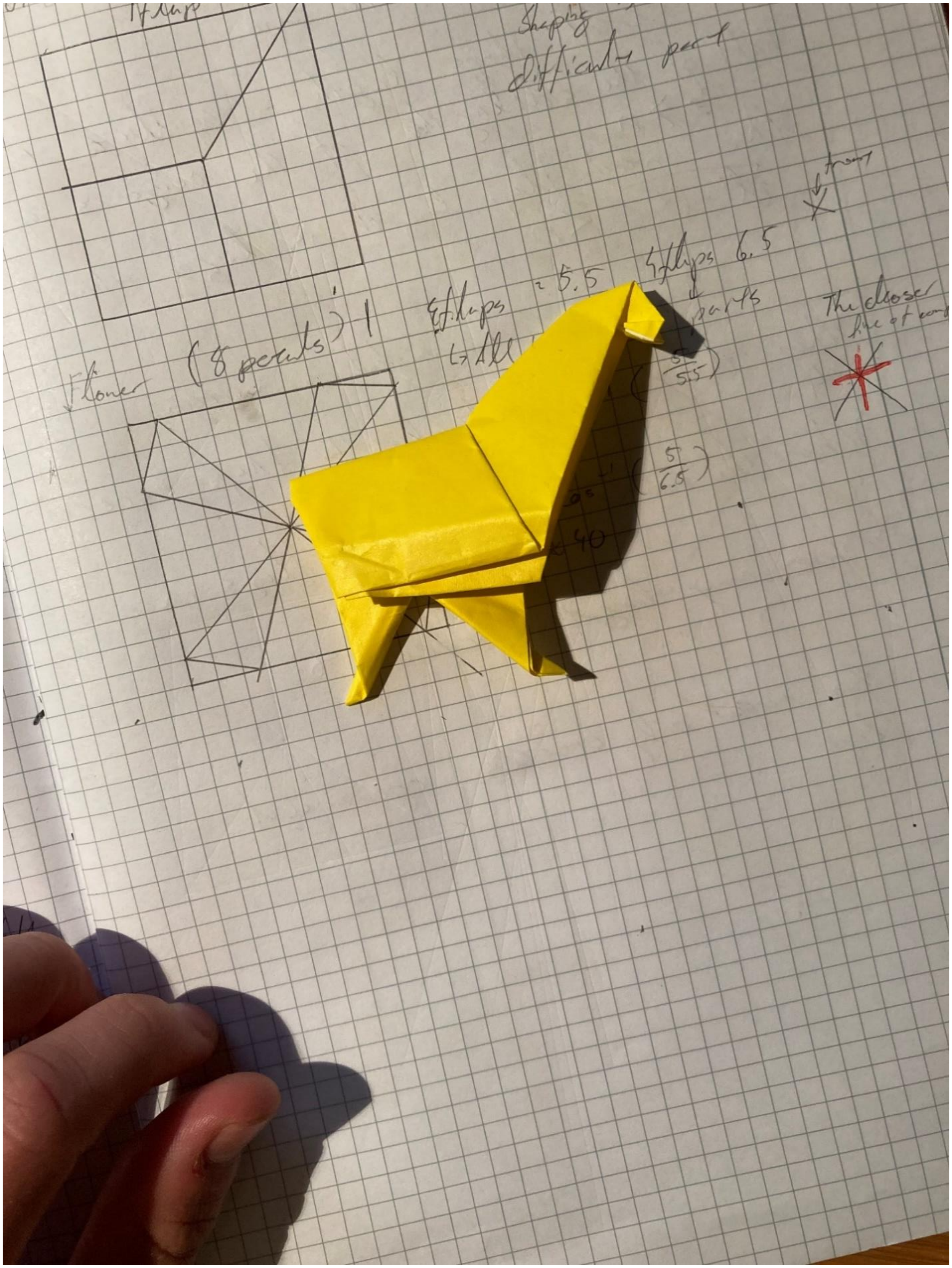


Figure 52. Giraffe crease pattern folded and shaped, fail

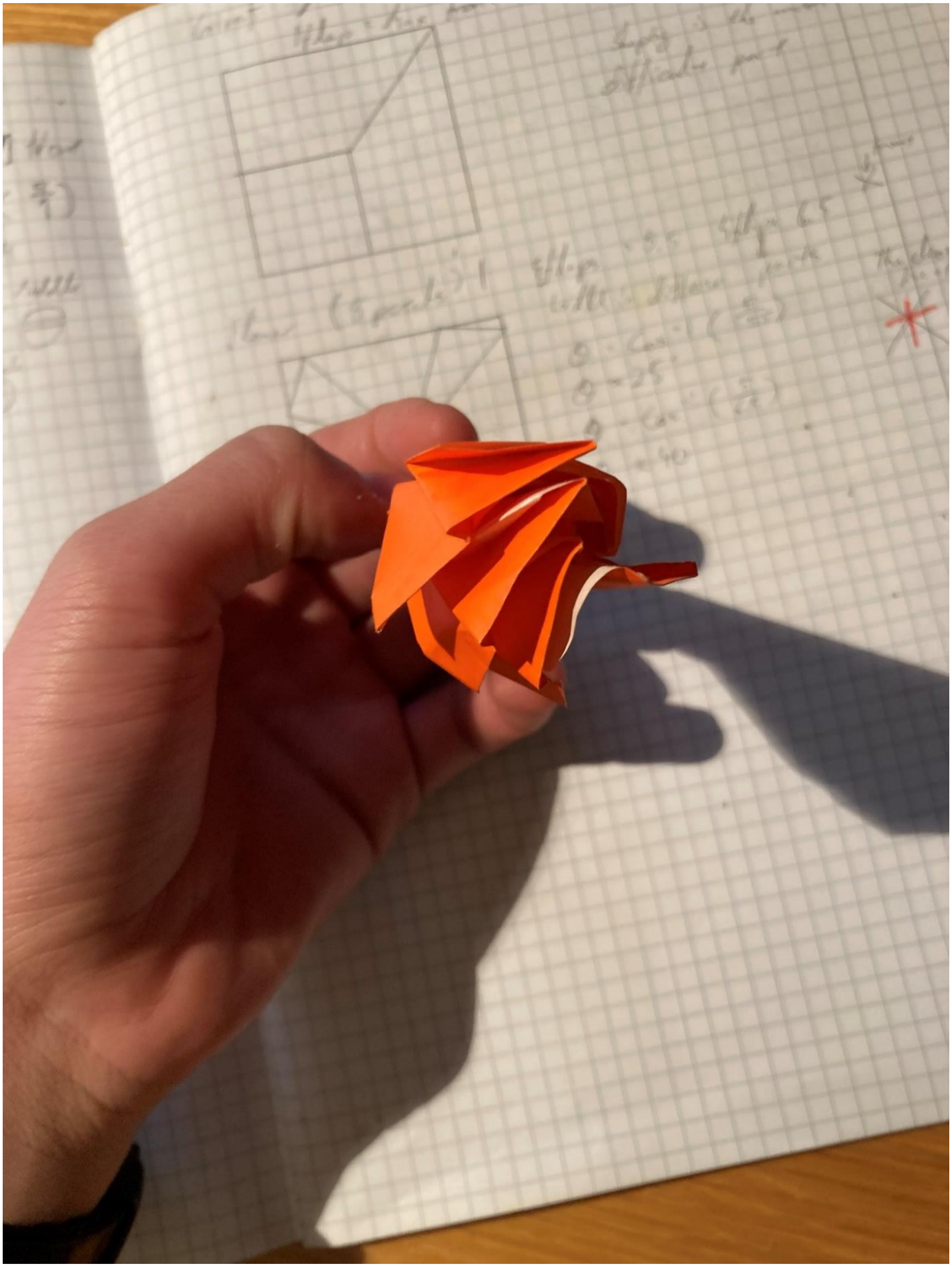


Figure 53. The different 8 petal flower crease pattern folded and shaped, fail

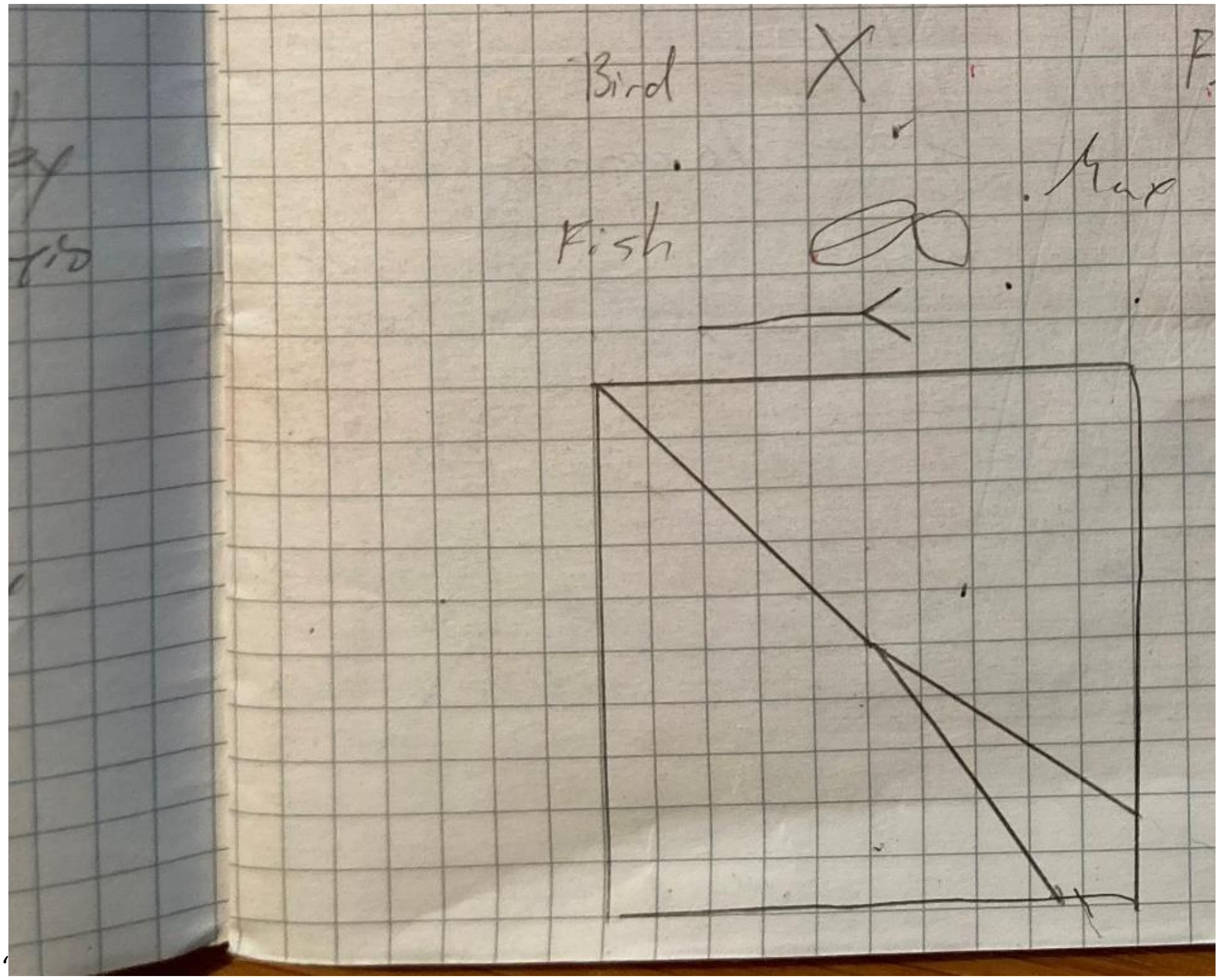


Figure 54. Crease pattern for fish

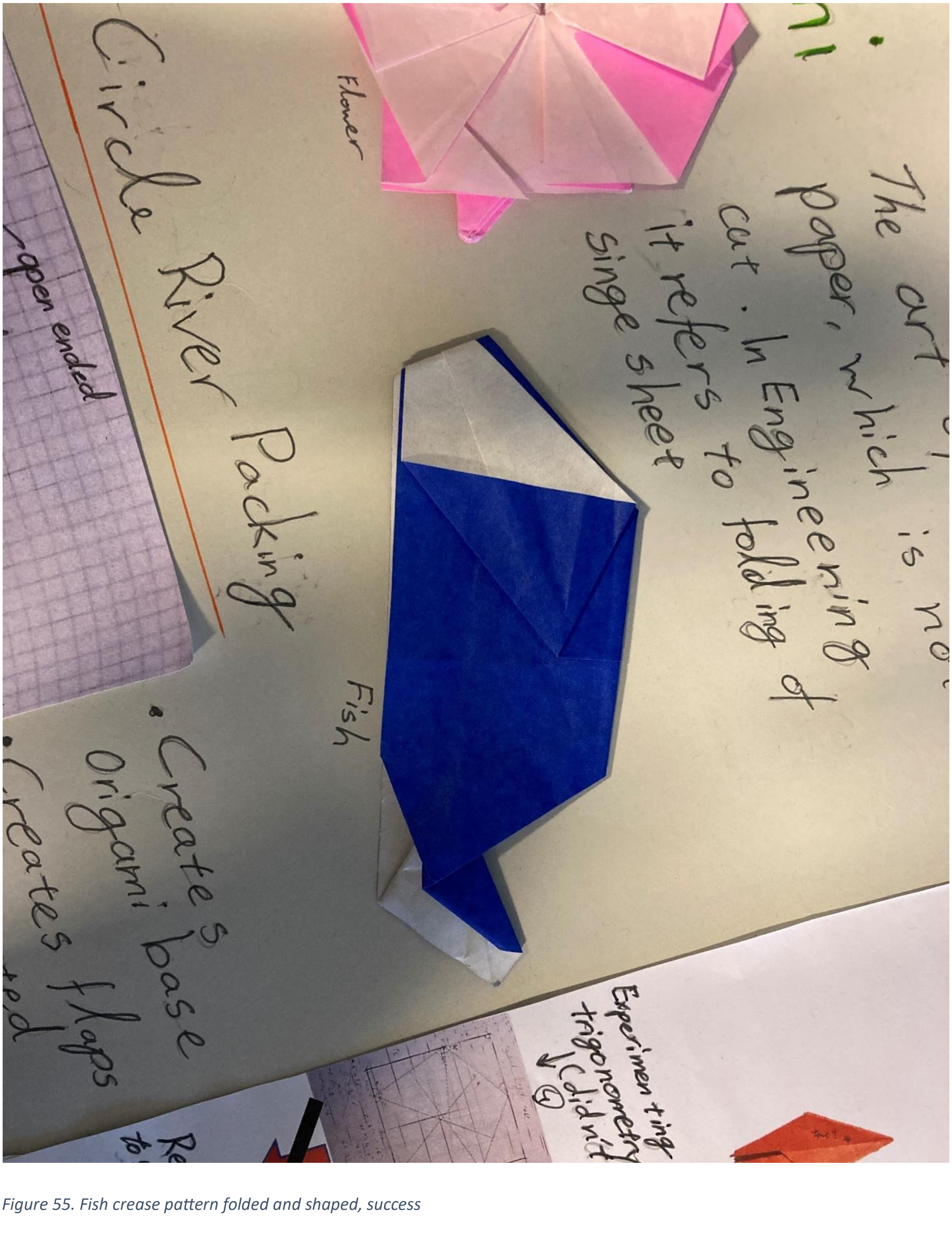


Figure 55. Fish crease pattern folded and shaped, success

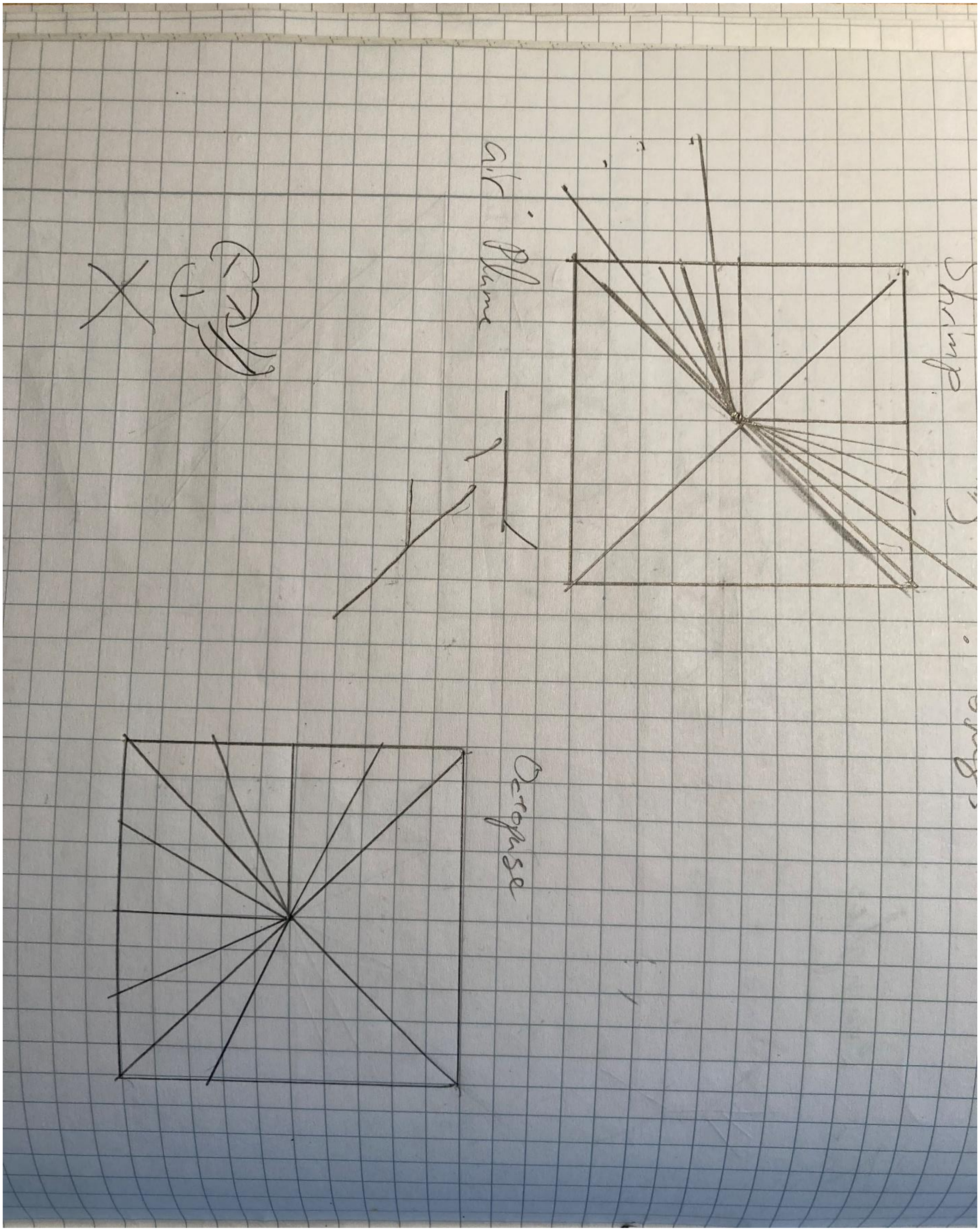


Figure 56. Crease pattern for shrimp and octopus

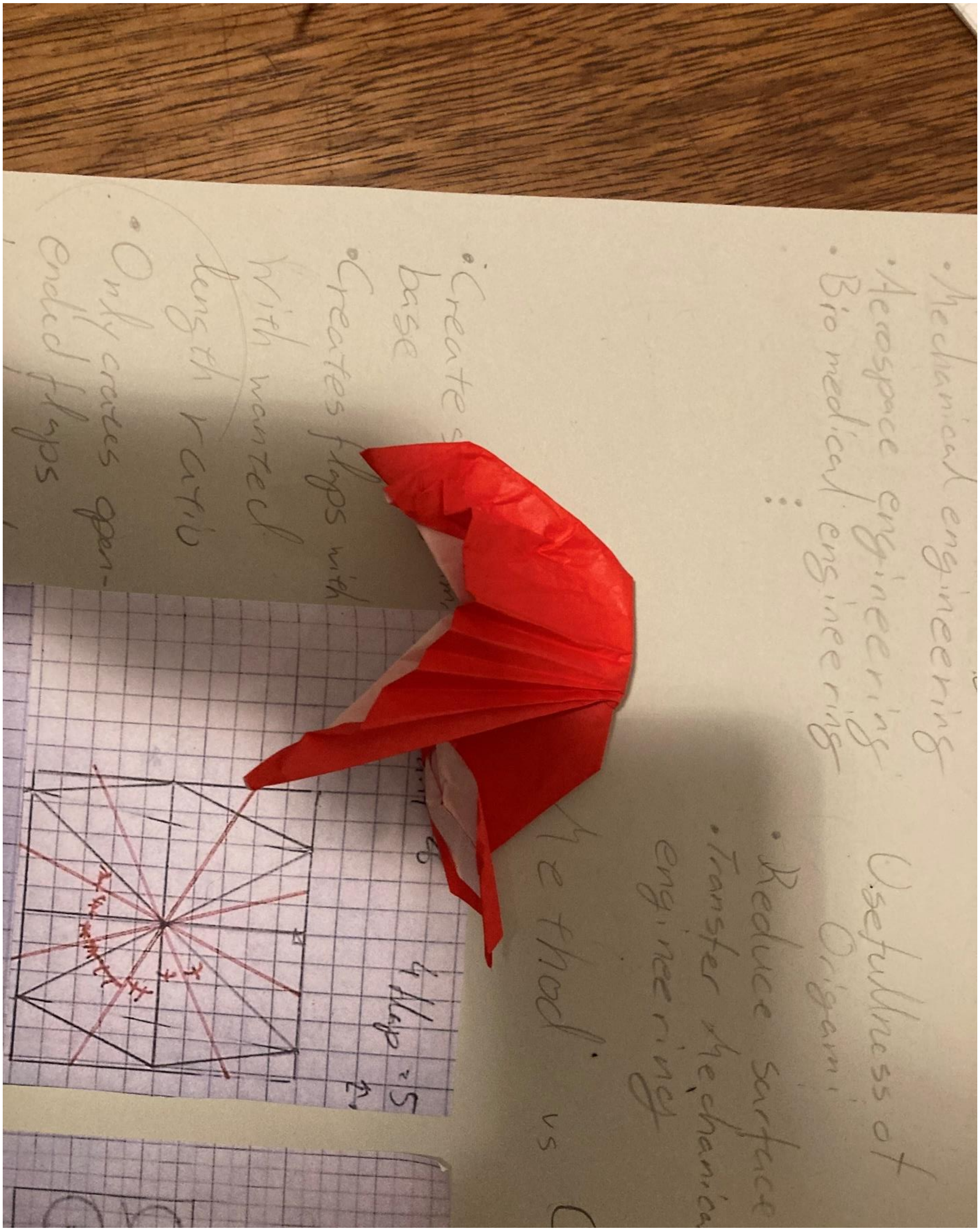


Figure 57. Shrimp crease pattern folded and shaped, fail

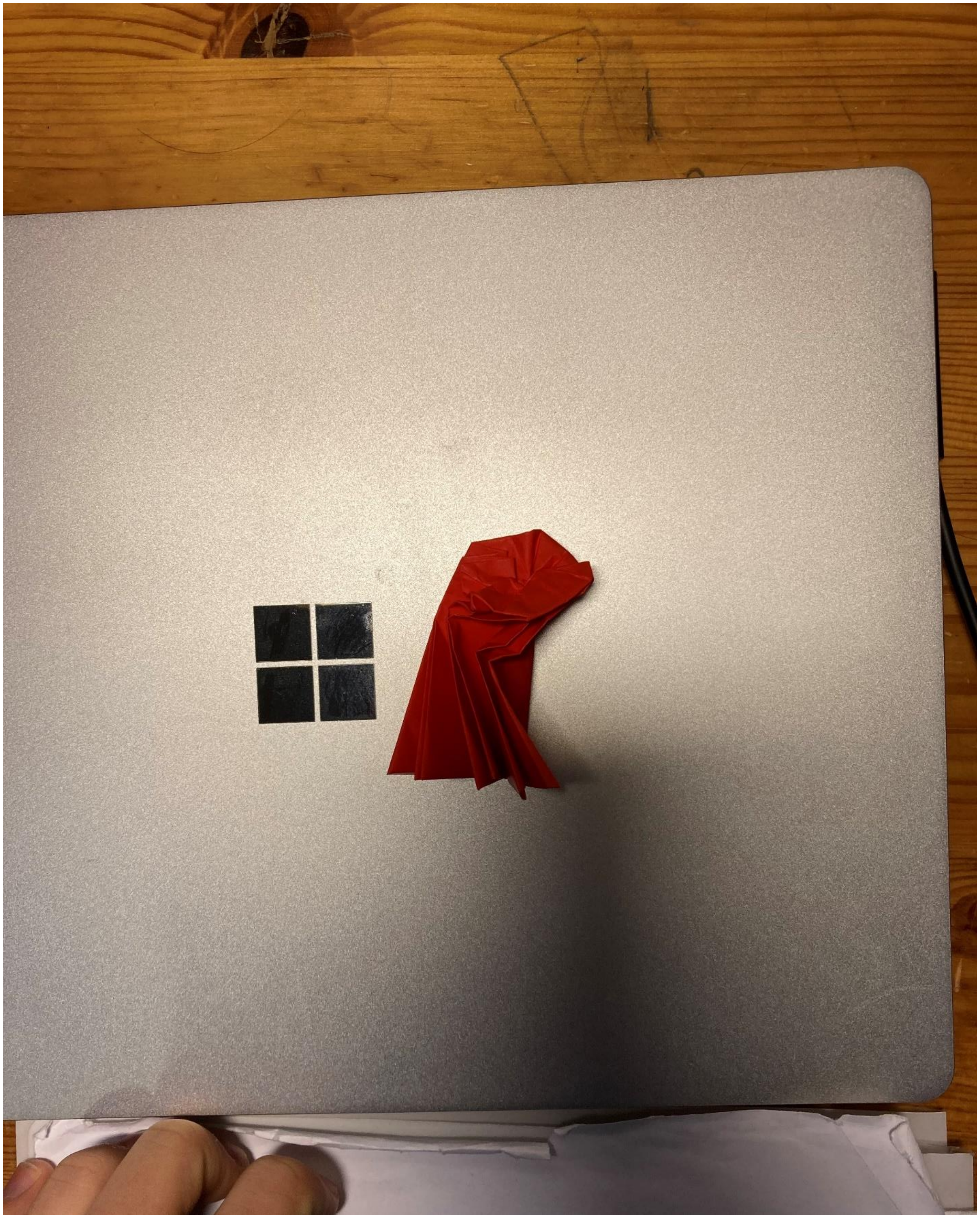


Figure 58. Octopus crease pattern folded and shaped, fail

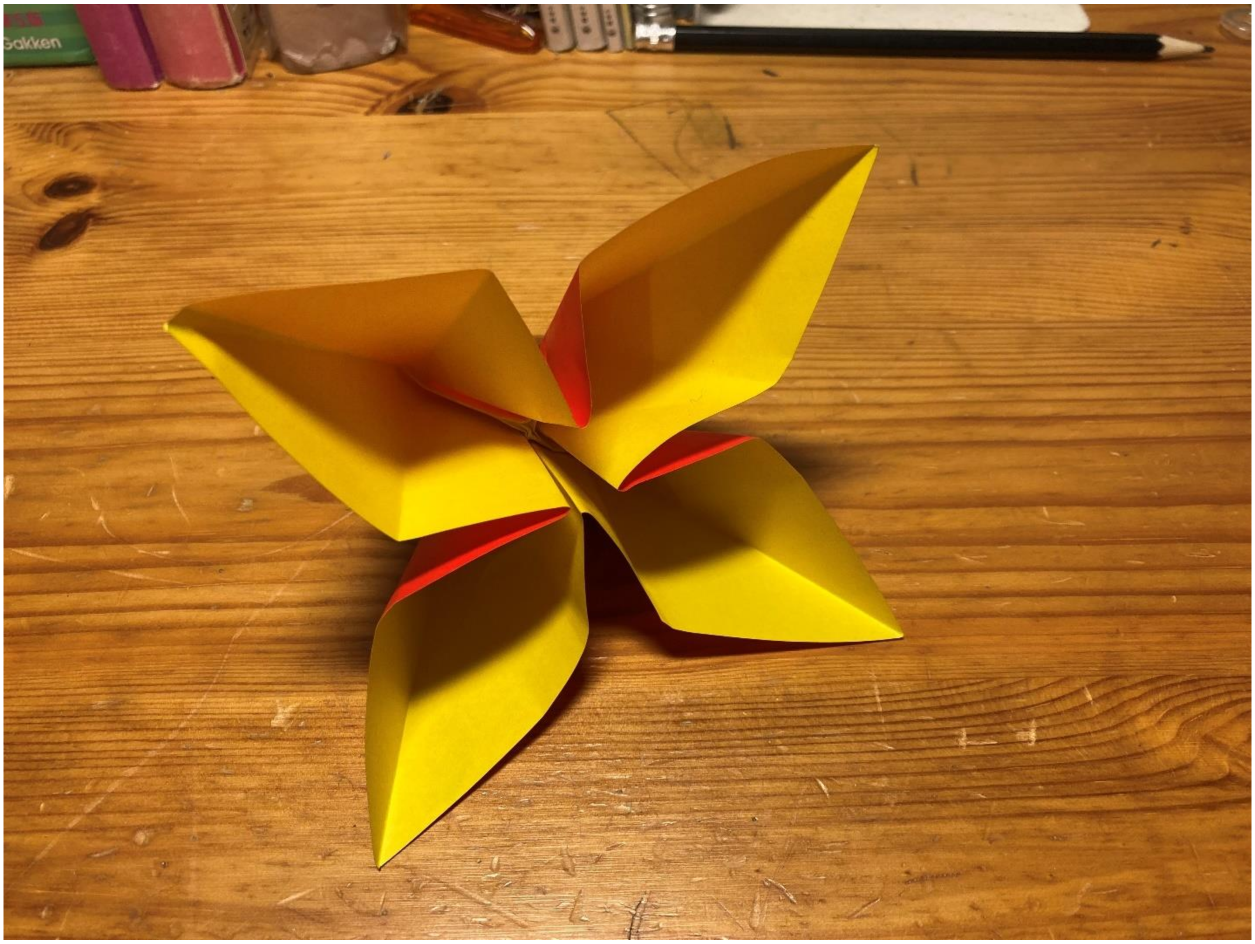


Figure 59. 3D 4 petal flower, success

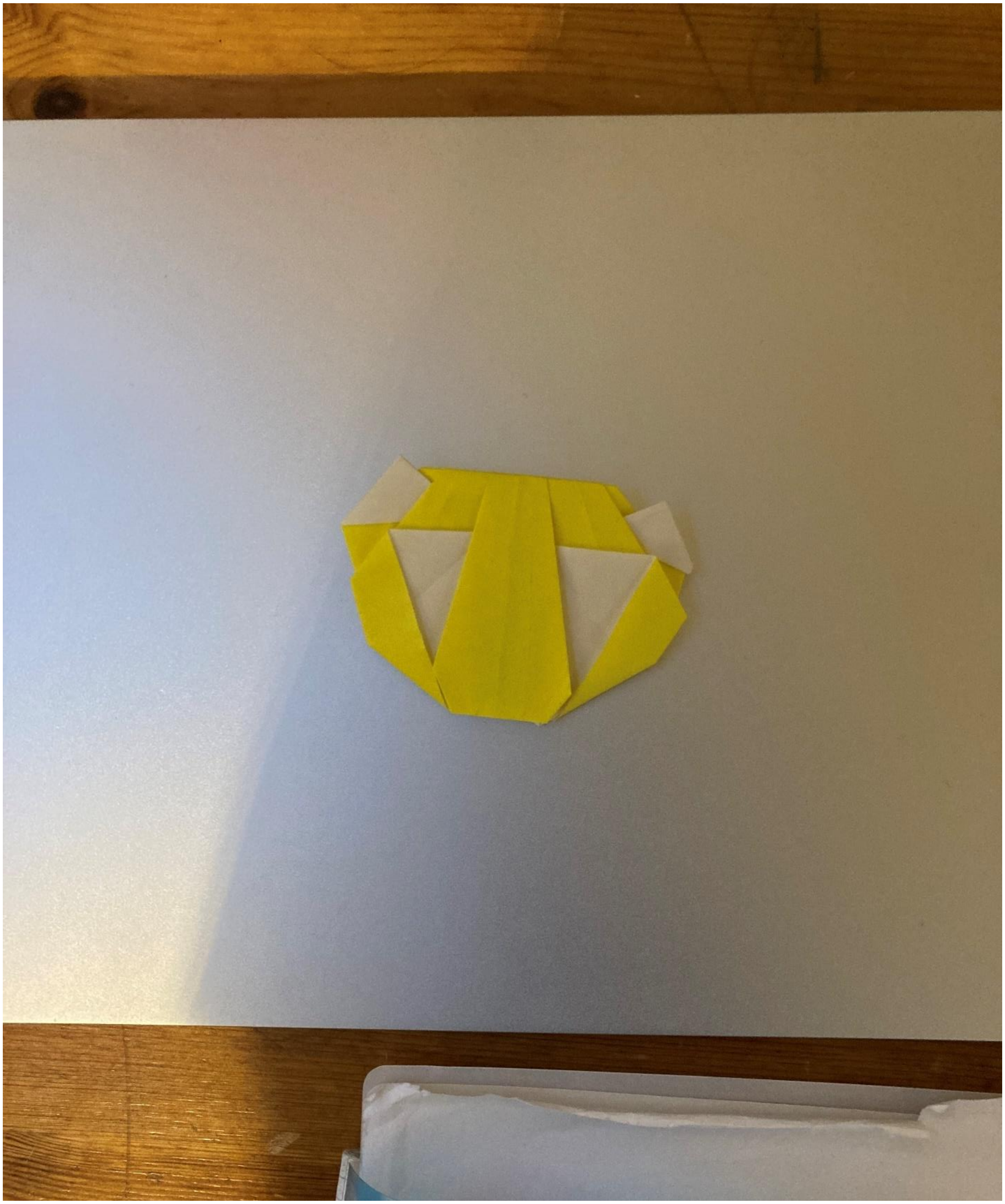


Figure 60. Lion-face, success

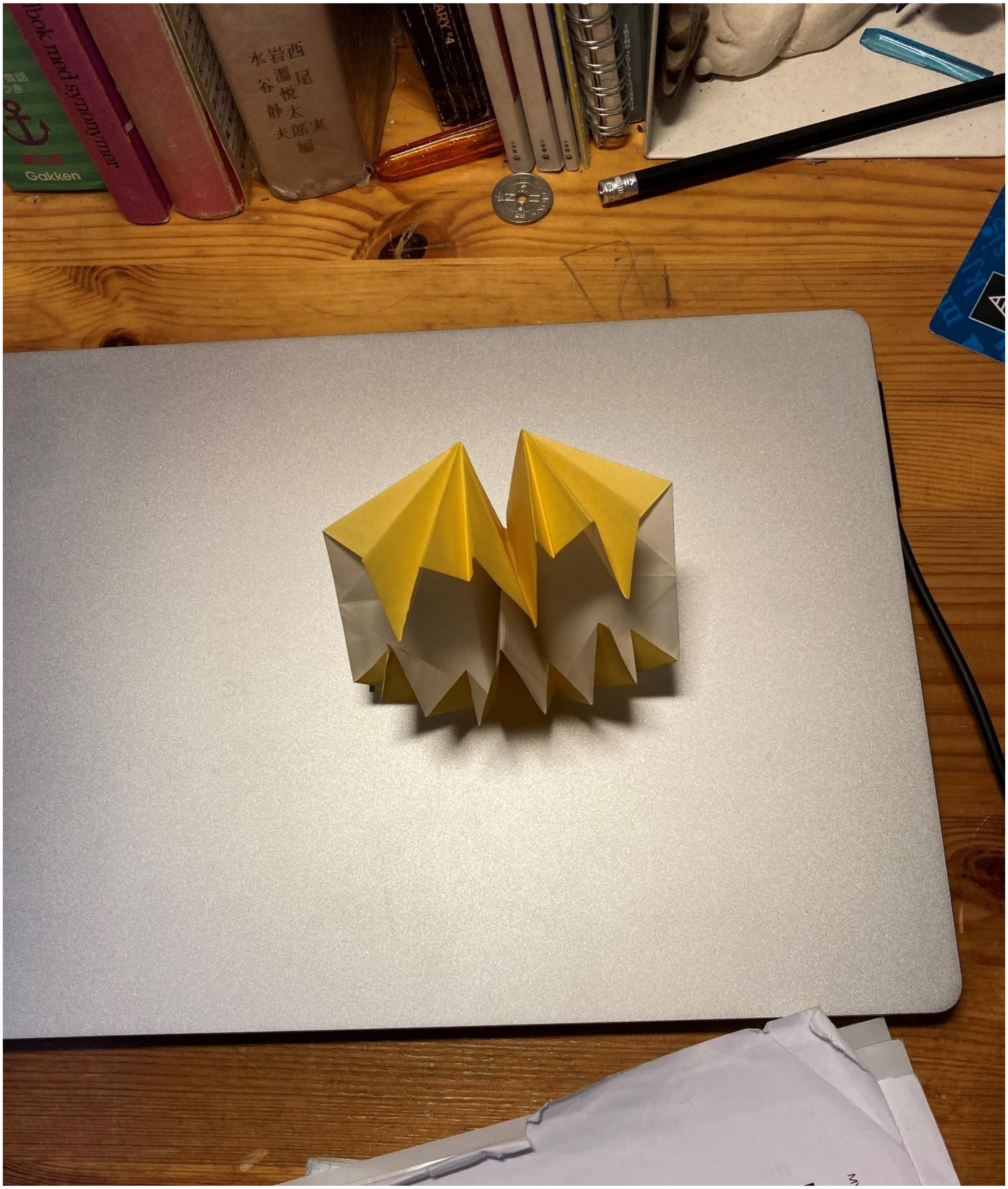


Figure 61. 3D Clam shell (Tridacna), success

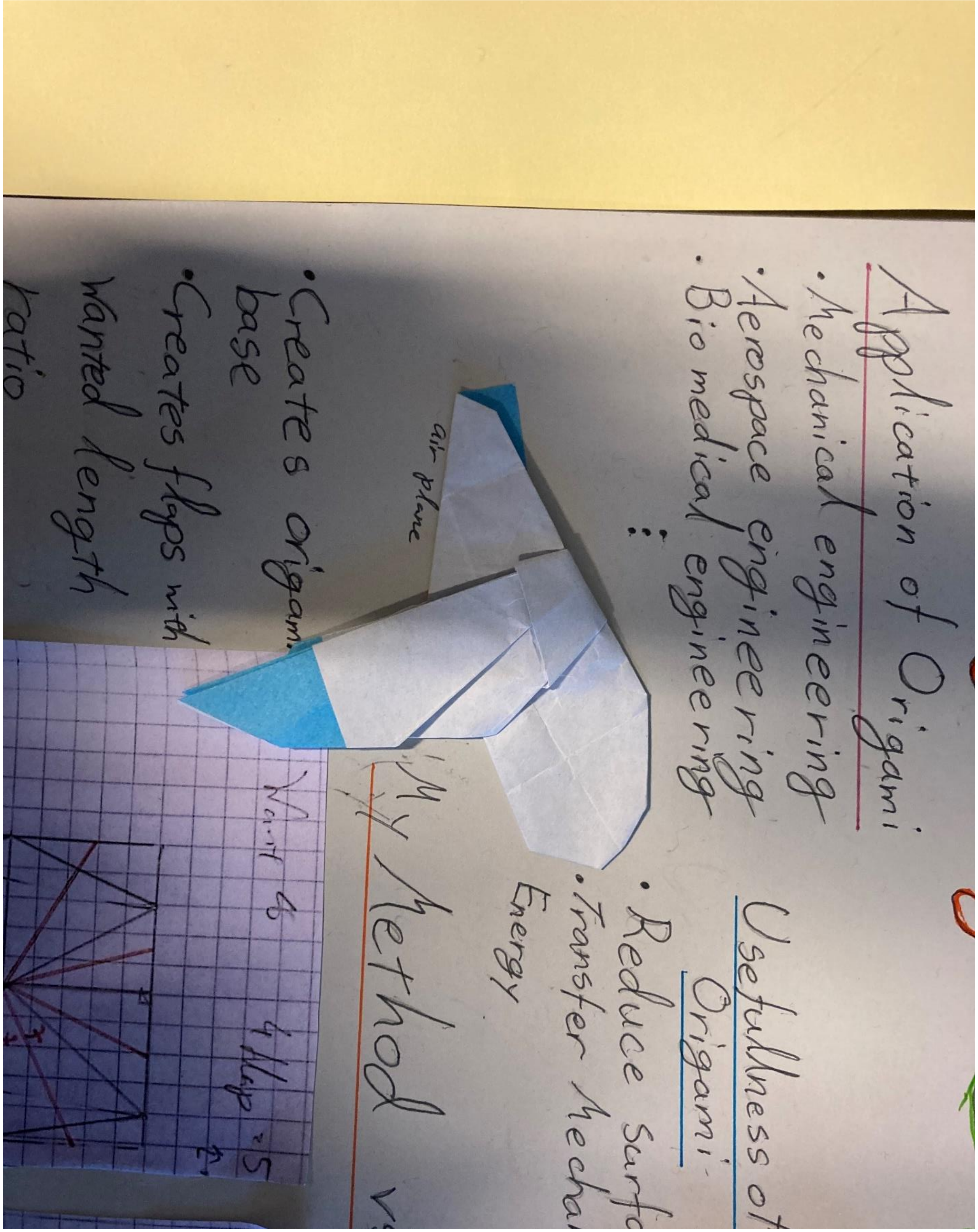


Figure 62. Airplane, success

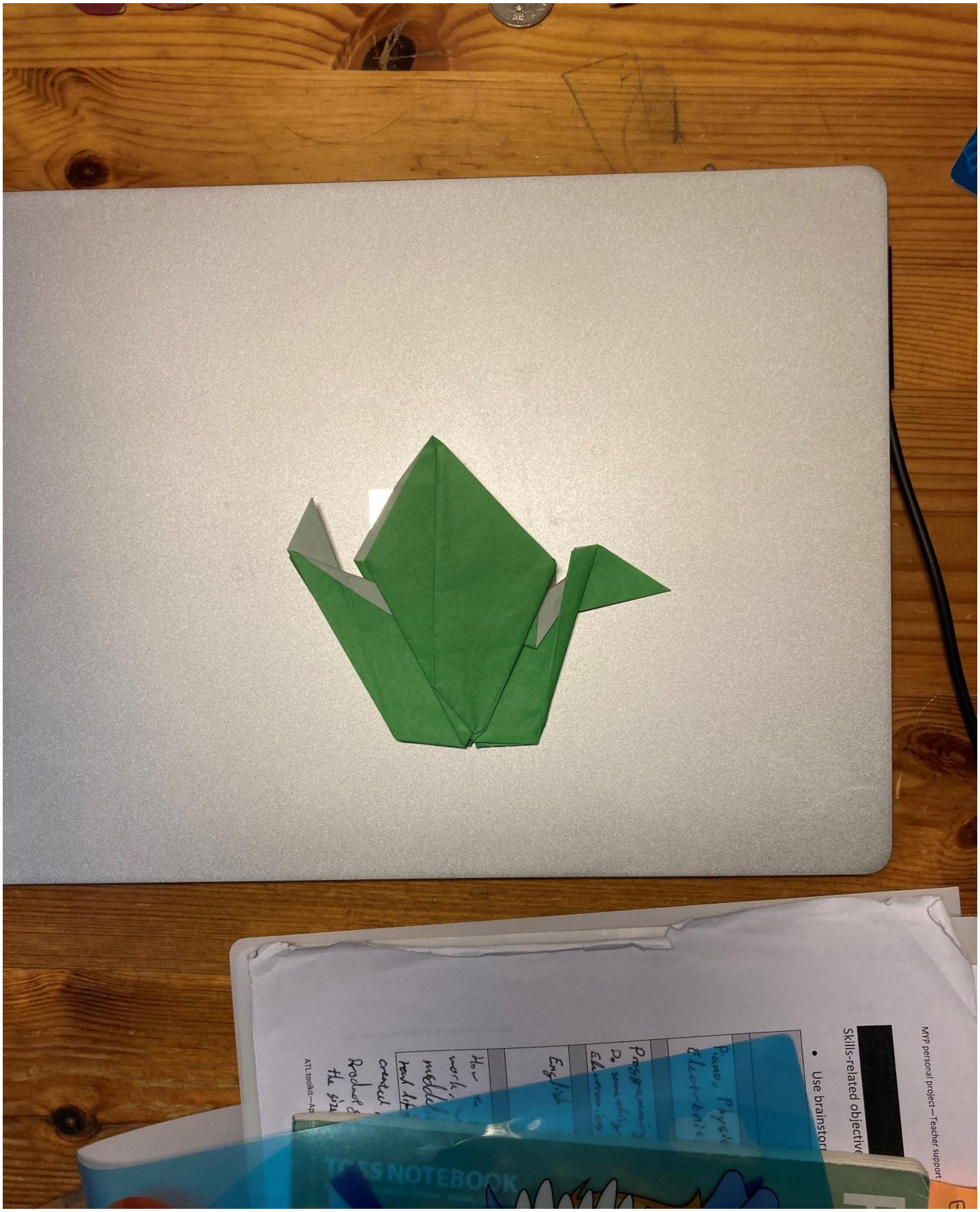


Figure 63. Bird, success

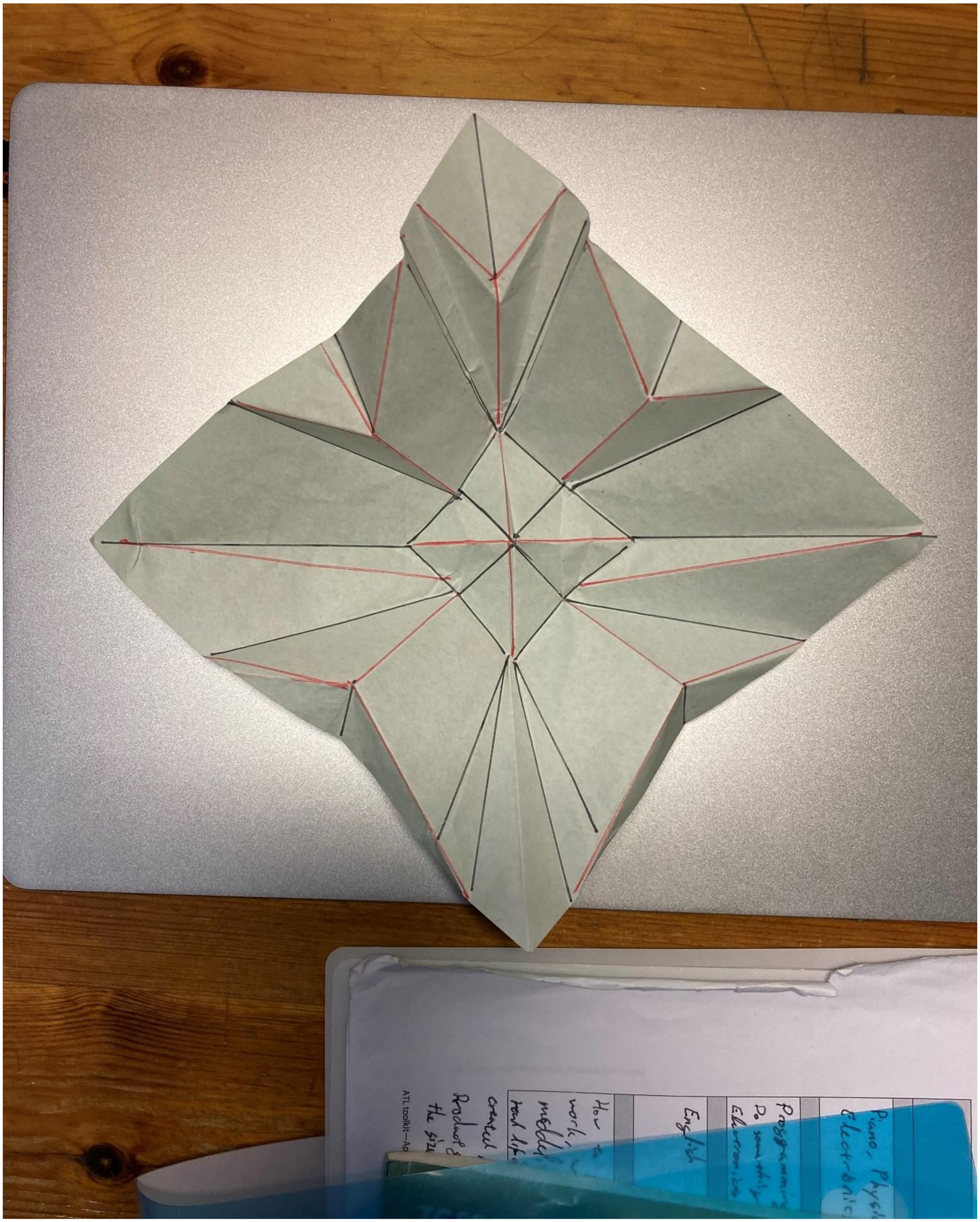
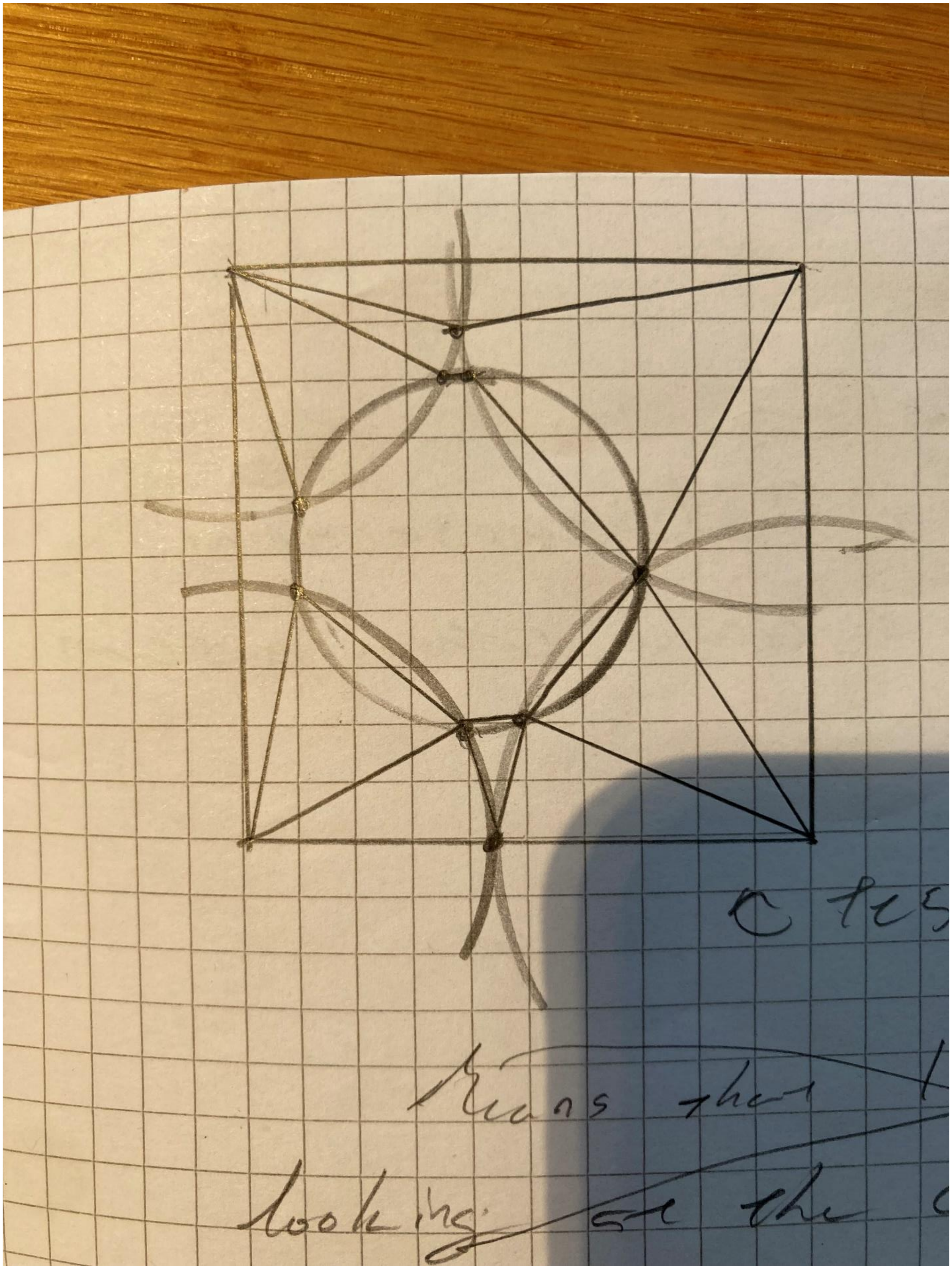
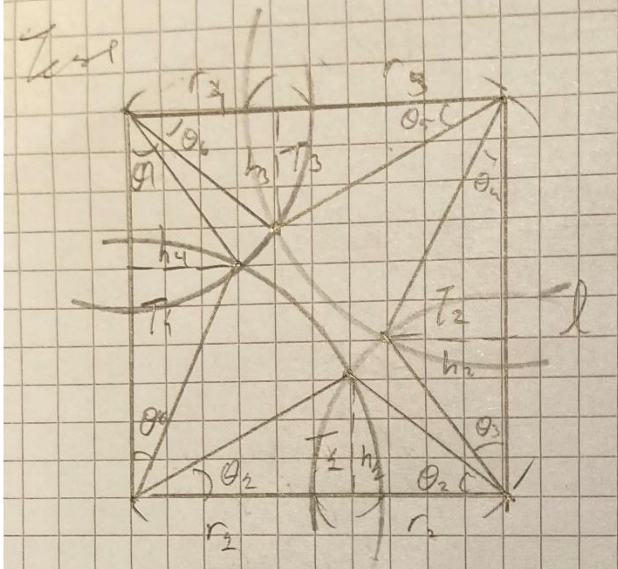


Figure 64. Bird design final crease pattern

Appendix 7. Second stage experimental sketches exploring a new approach





$l = 10$

Our focus [illegible]
is specific valu[illegible]
for industry

$$r_1 = \frac{5}{\cos(45)} \approx 7.07$$

$$r_2 = 5$$

$$r_4 = 5$$

$$r_3 = \frac{5}{\cos(45)} \approx 7.07$$

T_1

$$\theta_1 = \sin^{-1}\left(\frac{\sqrt{\frac{r_1^2 + r_2^2 - l^2}{2}}}{r_1}\right)$$

← didn't w[illegible]

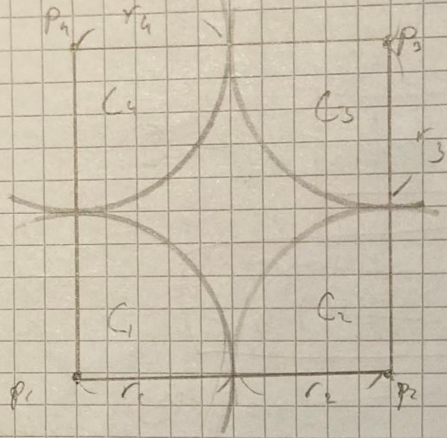
← firstly ~~th[illegible]~~ negative

$i^2 = -$

$$= \sin^{-1}\left(\frac{\sqrt{\frac{\left(\frac{5}{\cos(45)}\right)^2 + 25 - 100}{2}}}{\frac{5}{\cos(45)}}\right)$$

$=$

distance between centre of circle be decided by user

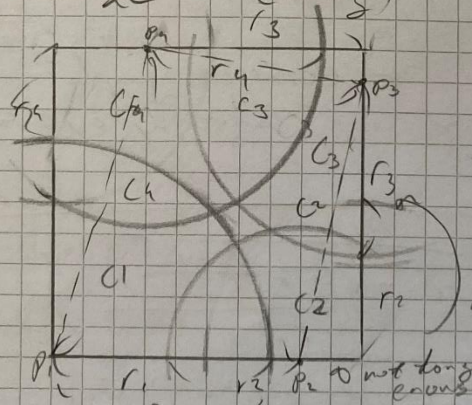


$C_1 \Rightarrow x^2+y^2=r_1^2$

$C_2 \Rightarrow (10-x)^2+y^2=r_2^2$

$C_3 \Rightarrow (10-x)^2+(10-y)^2=r_2^2$

$C_4 \Rightarrow x^2+(10-y)^2$



c = connectivity

C = circle

C change confusion

$0 \le c \le l$ $a^2+b^2=c^2$

more into ← pythagoras ← $c^2 = c_2$

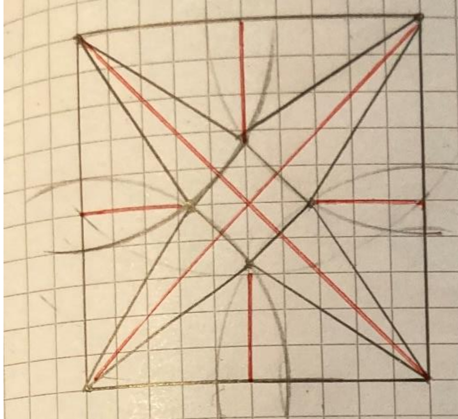
if $c > l - x$ ←
when c longer
than length lt

$C_1 \Rightarrow x^2+y^2=r_1^2$

$C_2 \Rightarrow (c_1-x)^2+y^2=r_2^2$

$C_3 \Rightarrow$

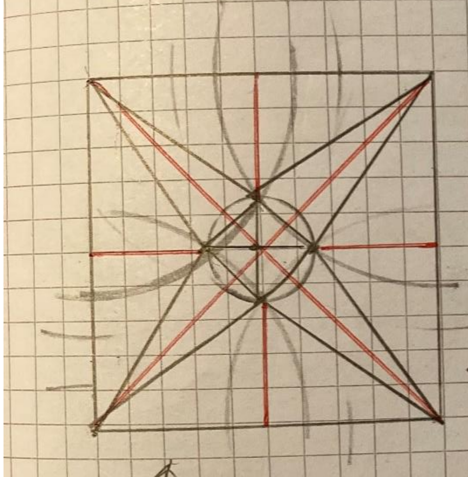
based on c2 need to find both side lengths using pythag



Circle intersection method test 3

all flaps equal

Make sure a circle dont the intersection behind another from centre of circle

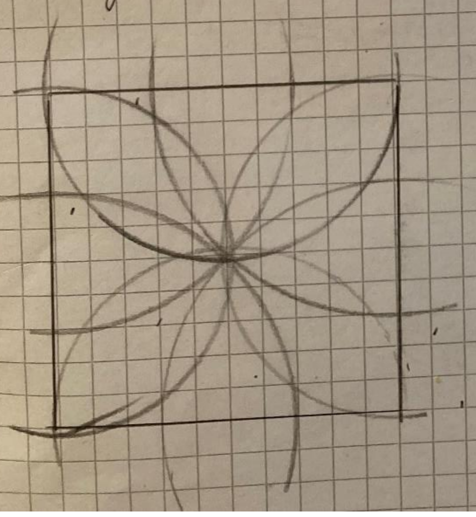


test 2

8 flaps

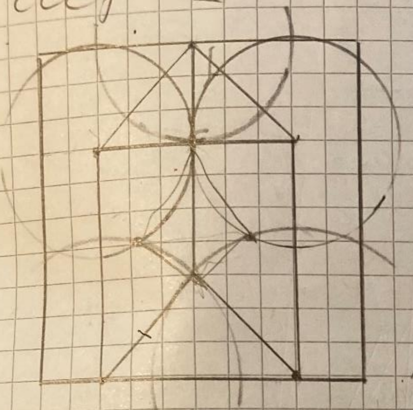
← need a way to do this using ratios

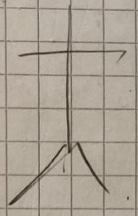
↑ This not really mathematics, but ~~the~~ mathematics will be needed to know where to place the flap

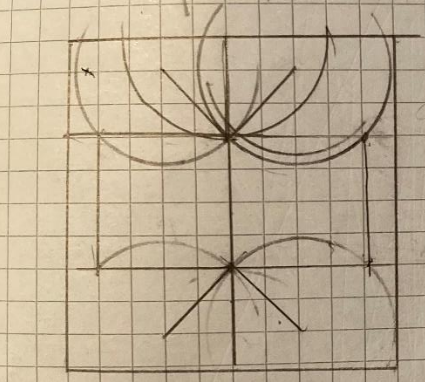


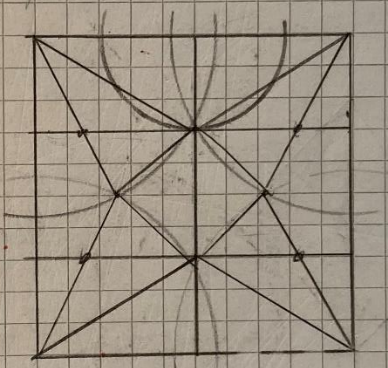
Rather than working on connectivity more important is getting a closed ended flap

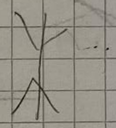
ecef 2



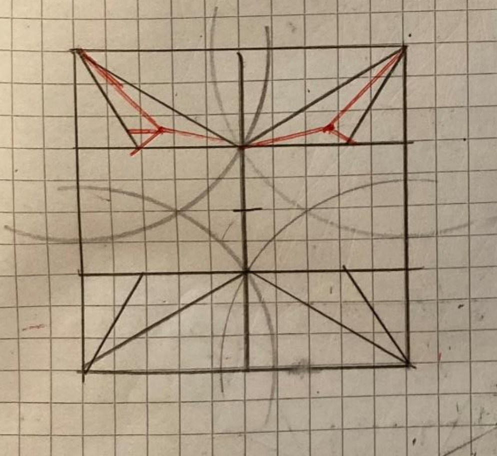


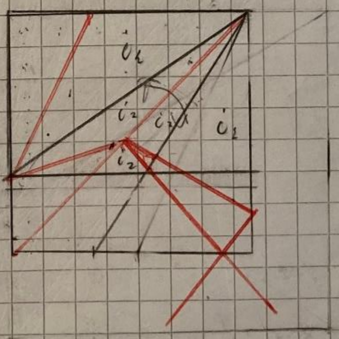




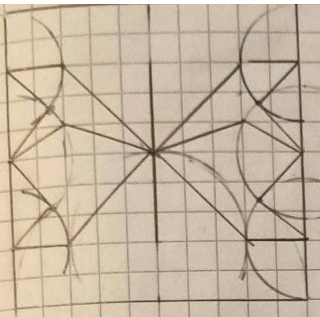


i - identical
to differ
scales

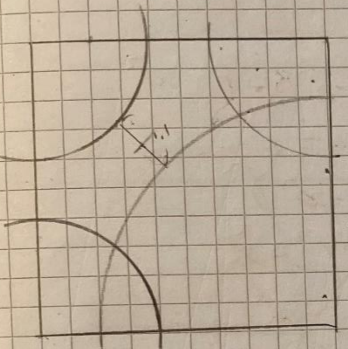




~~Circle rules only~~



Only open ended

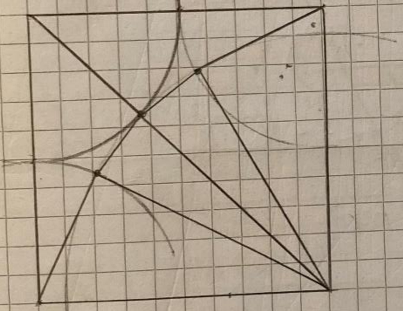


i: initiall length $\frac{l}{2}$

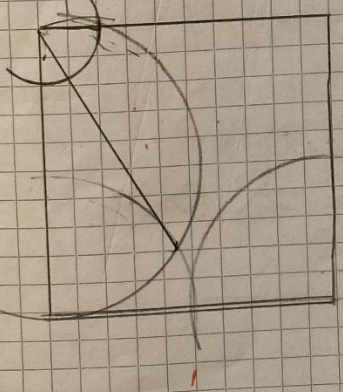
~~[illegible]~~ if not all touching something measure the length between (1.1 cm) thus actuall should be $\frac{n}{2}$.

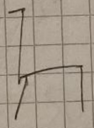
$$\times \frac{\sqrt{2(5^2)}+1.1}{\sqrt{2(5^2)}}$$

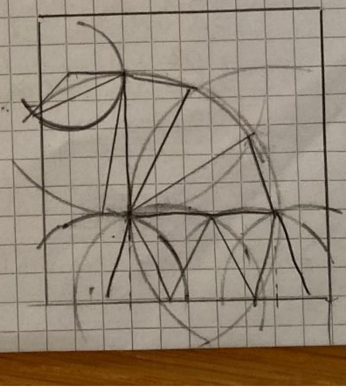
$$l \times \frac{\sqrt{2(5)^2}+0.55}{\sqrt{2(5^2)}}.$$



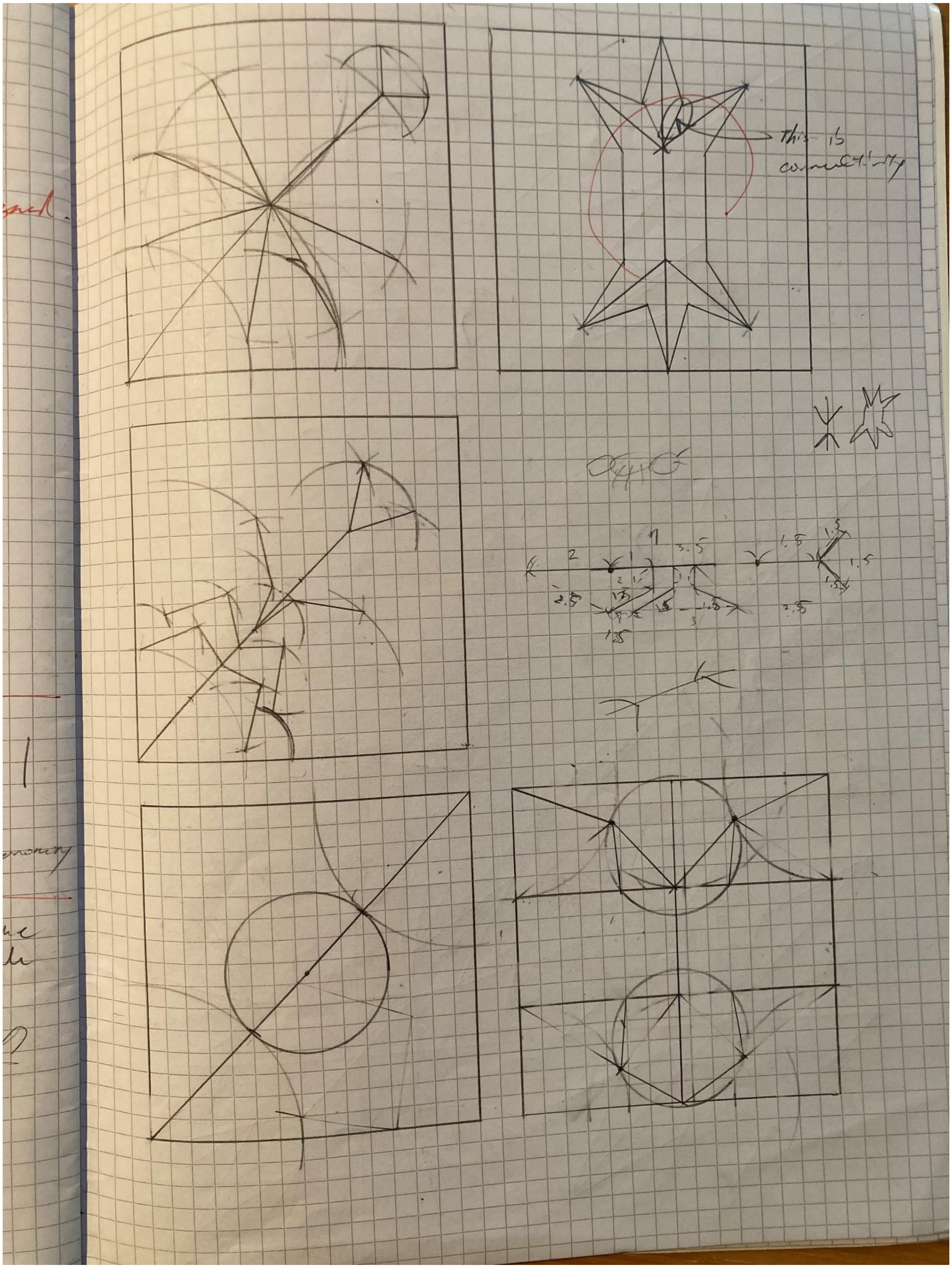
Make things rather 3d and make ~~展開~~ 展開図 which used as the base of the fragment

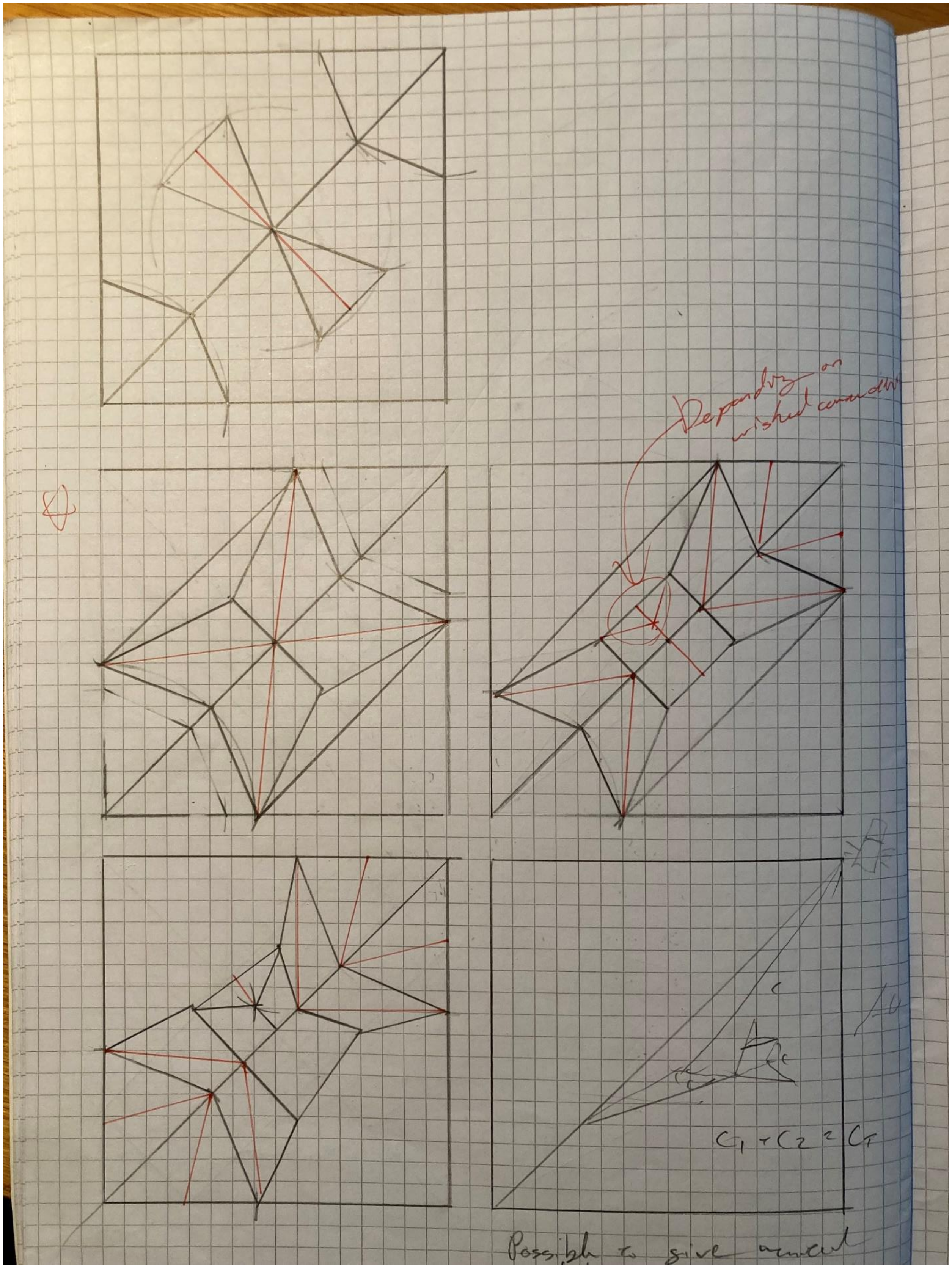






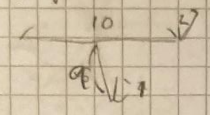
Outside the square meeting point match?


This is connectivity
2
1
3.5
1.5
1.5
1.5
1.5
2.5
2
1.5
1.5
1.5
1.5
3
1.35
2.5



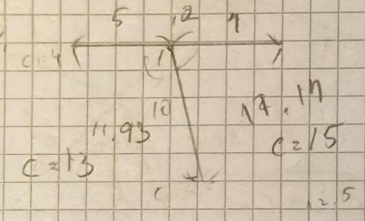
Possible to give amount

①.



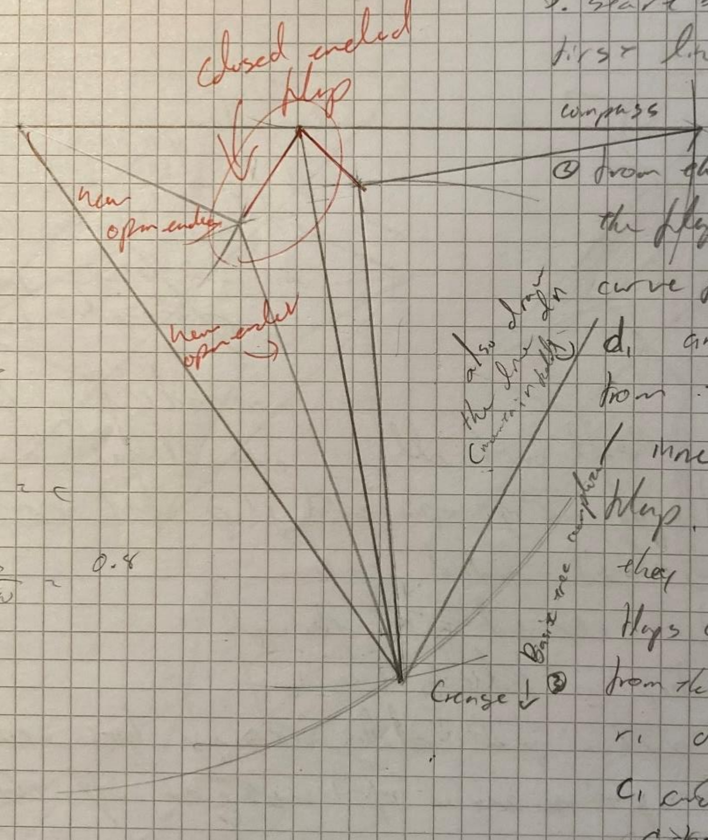
fail safe with angles

②



E Using cosine law convert angle into length and can also reverse to fail safe the angles (must add up to 360°)

③



①. start by drawing the first line, from there ease ⇒ length r₁

② from the outer end of the flap draw out a curve for the distance d₁ and distance r₂ from the crease center / inner side of the flap. The point where they meet is where the flaps outer side is

③ from the outer side of r₁ draw out length c₁ and c₂ from r₂ onto c → between r₁ and r₂ is decided.

$\sqrt{a^2+b^2}=c$

$\frac{13}{5+w}=0.8$

No percent rather for → connectivity easier to understand they are the same

$$\frac{c}{r_1+r_2}=n$$

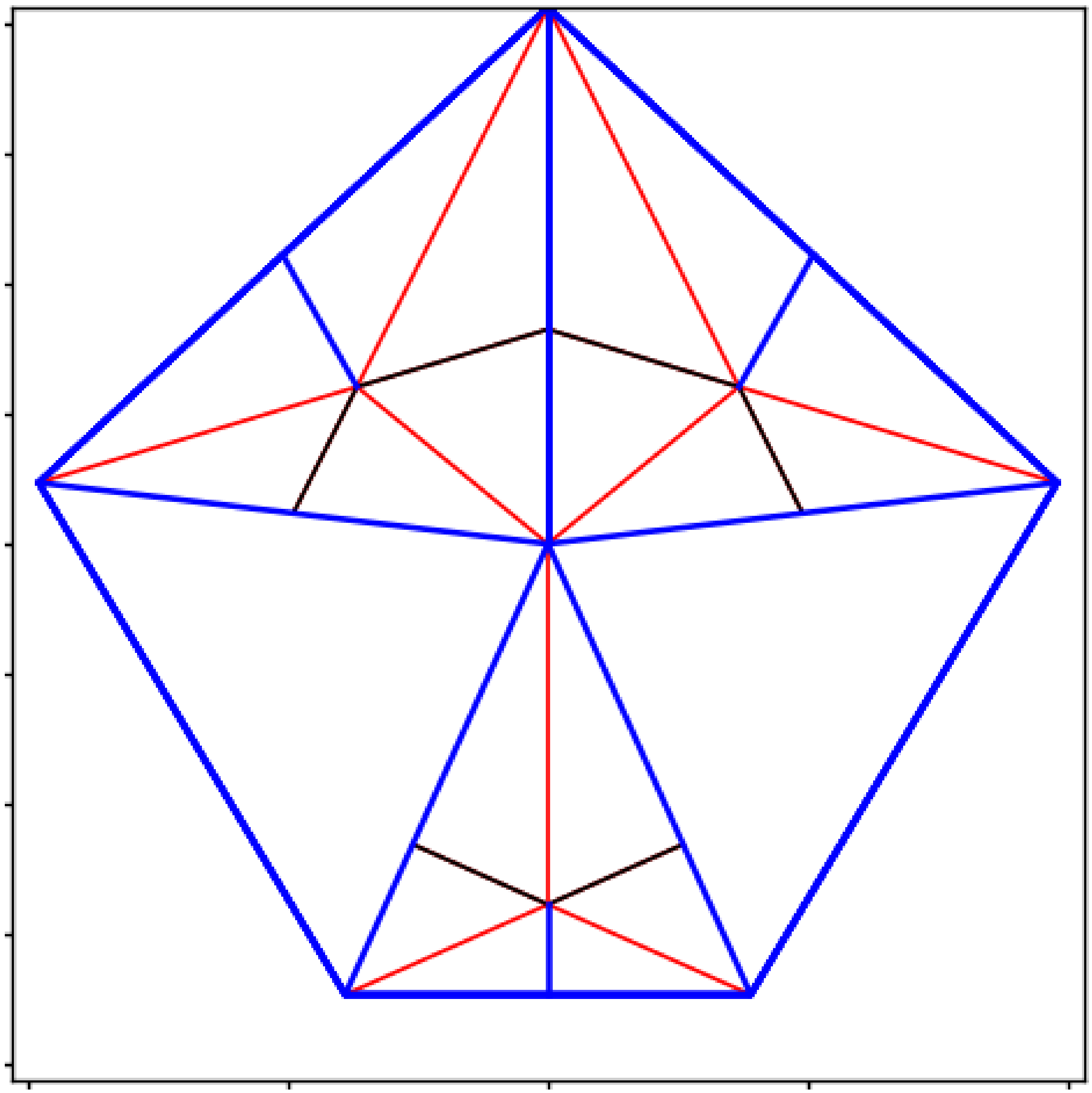
$c_1 = nr_1 \quad c_2 = nr_2$

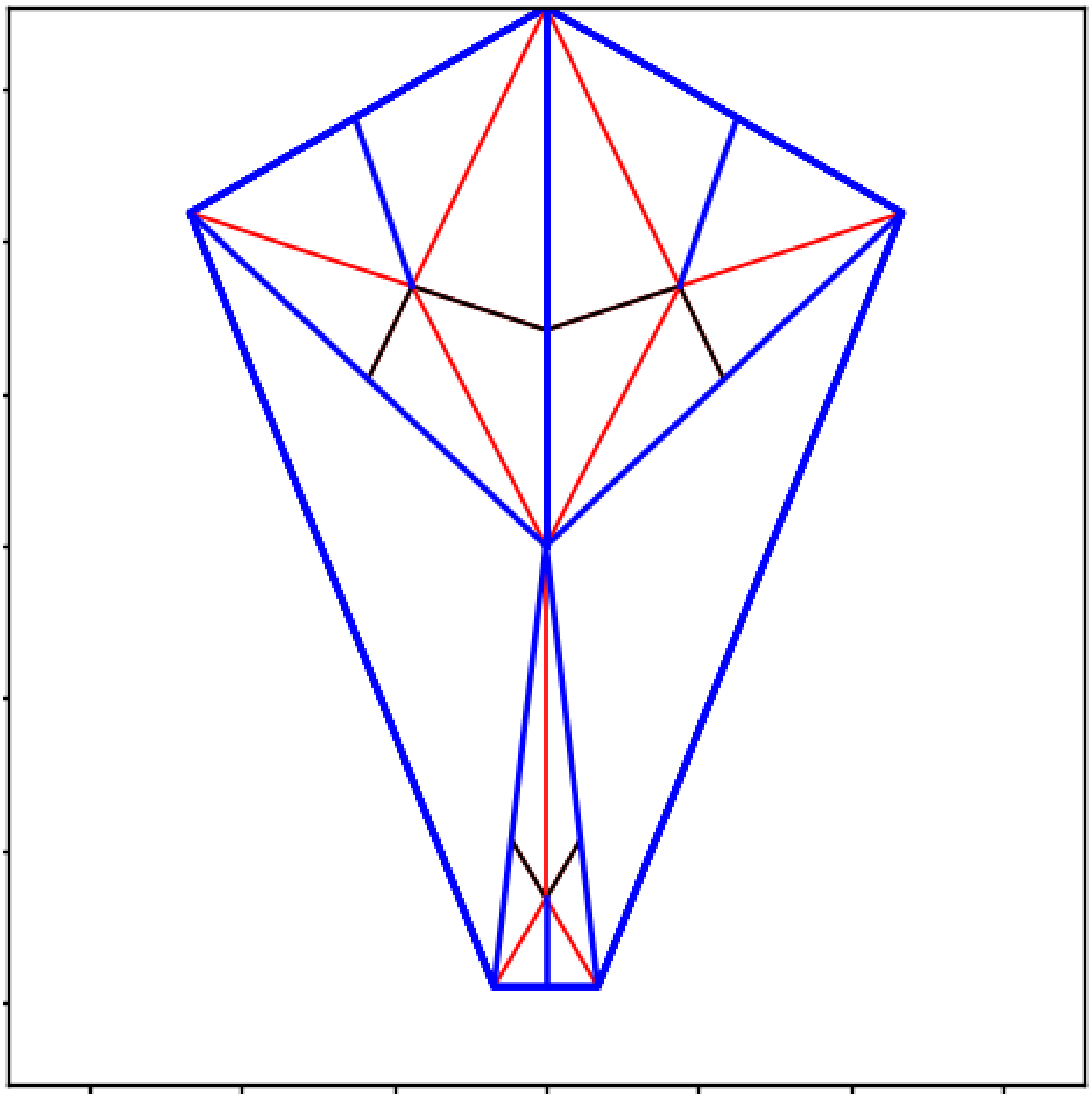
$c \nless r_1+r_2$

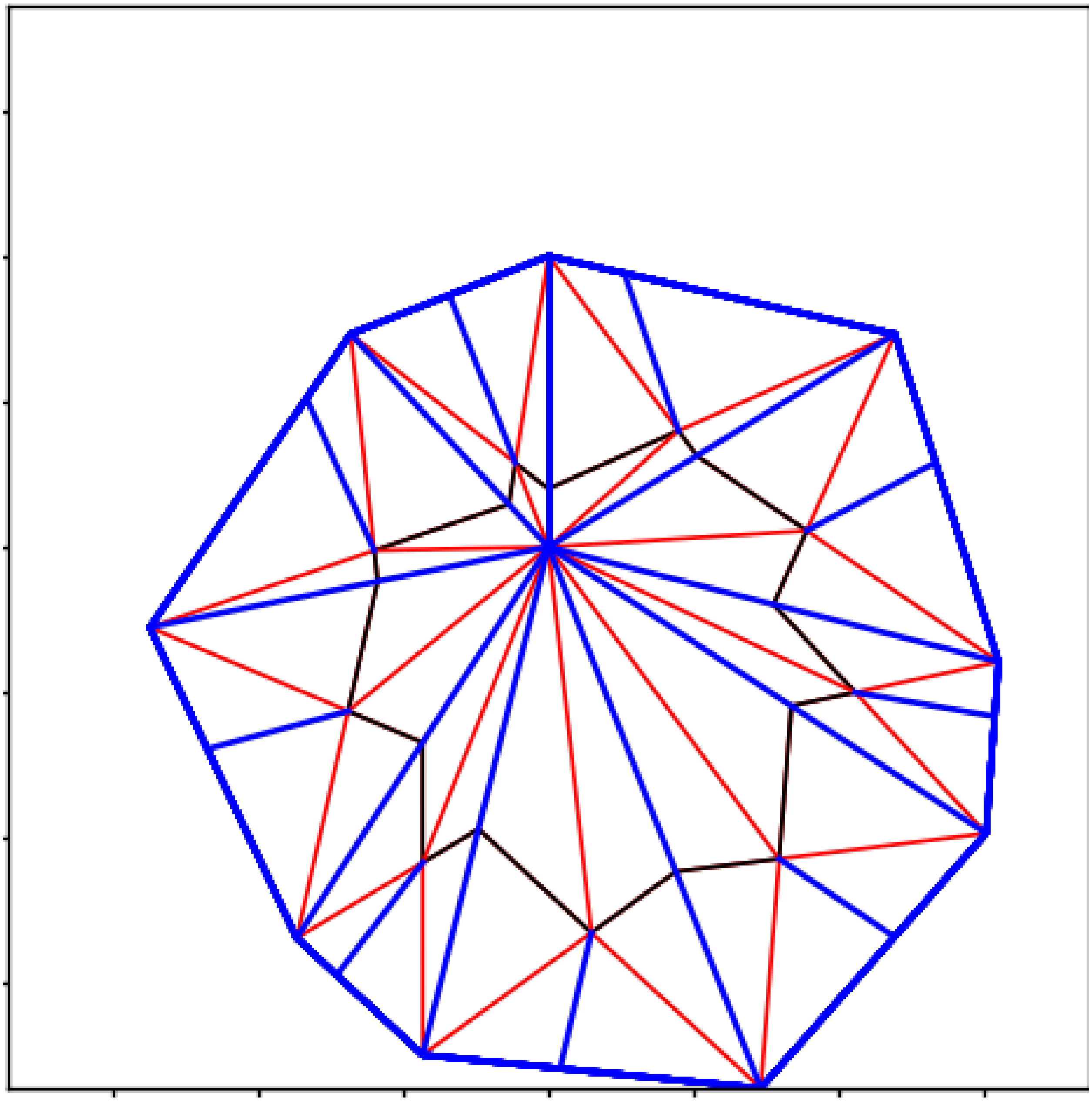
Appendix 8. Third stage Equation 5 calculator

```
import math
totalFlapNumber = 1+ int(input("Total number of flaps?: "))
n = 1

while n in range(totalFlapNumber):
    flapOpen1 = float(input("Open-ended flap length 1: "))
    flapOpen2 = float(input("Open-ended flap length 2: "))
    flapTotal1 = float(input("Total flap length 1: "))
    flapTotal2 = float(input("Total flap length 2: "))
    connectivity = float(input("Connectivity value between the two open-ended 
flaps: "))
    value = ((flapTotal1**2) * (connectivity**2) + (flapTotal2**2) * 
(connectivity**2) - flapOpen1**2 - flapOpen2**2 - 
2*flapOpen1*flapOpen2)/(2*flapTotal1*flapTotal2)
    angle = math.acos(value) * (180.0/math.pi)
    print(angle)
    n+=1
```

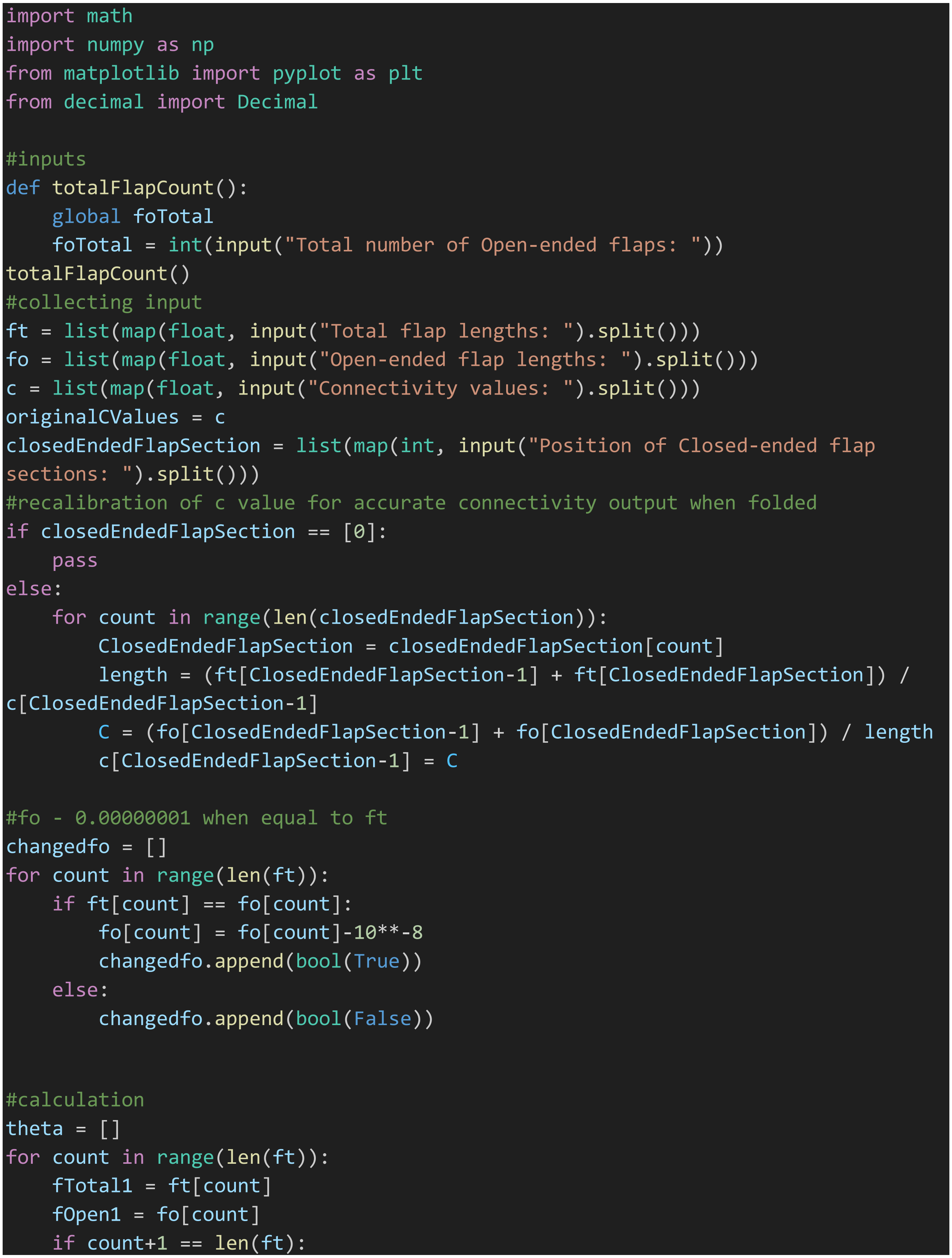
Appendix 9. Fourth stage additional mountain fold experiment



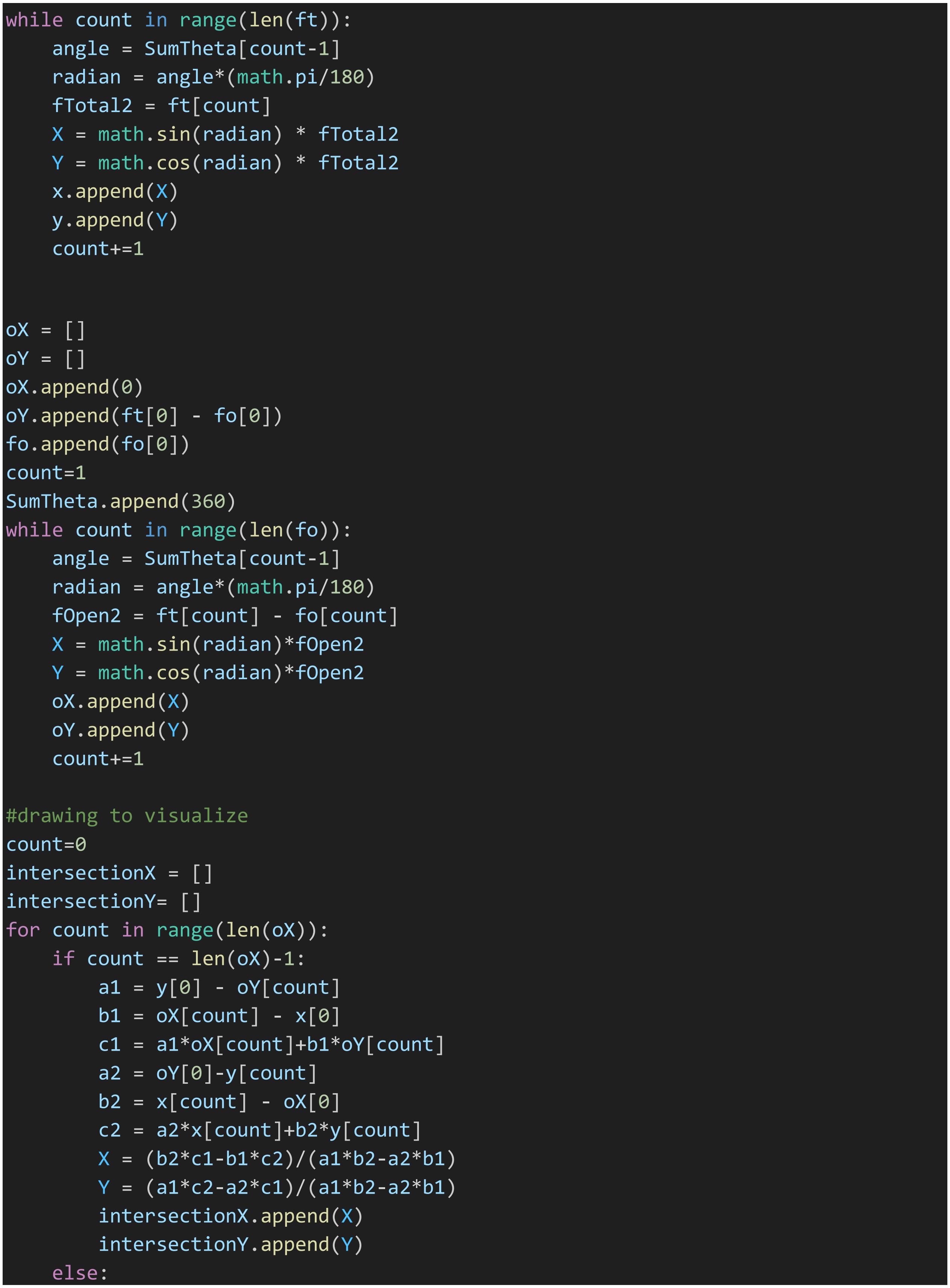




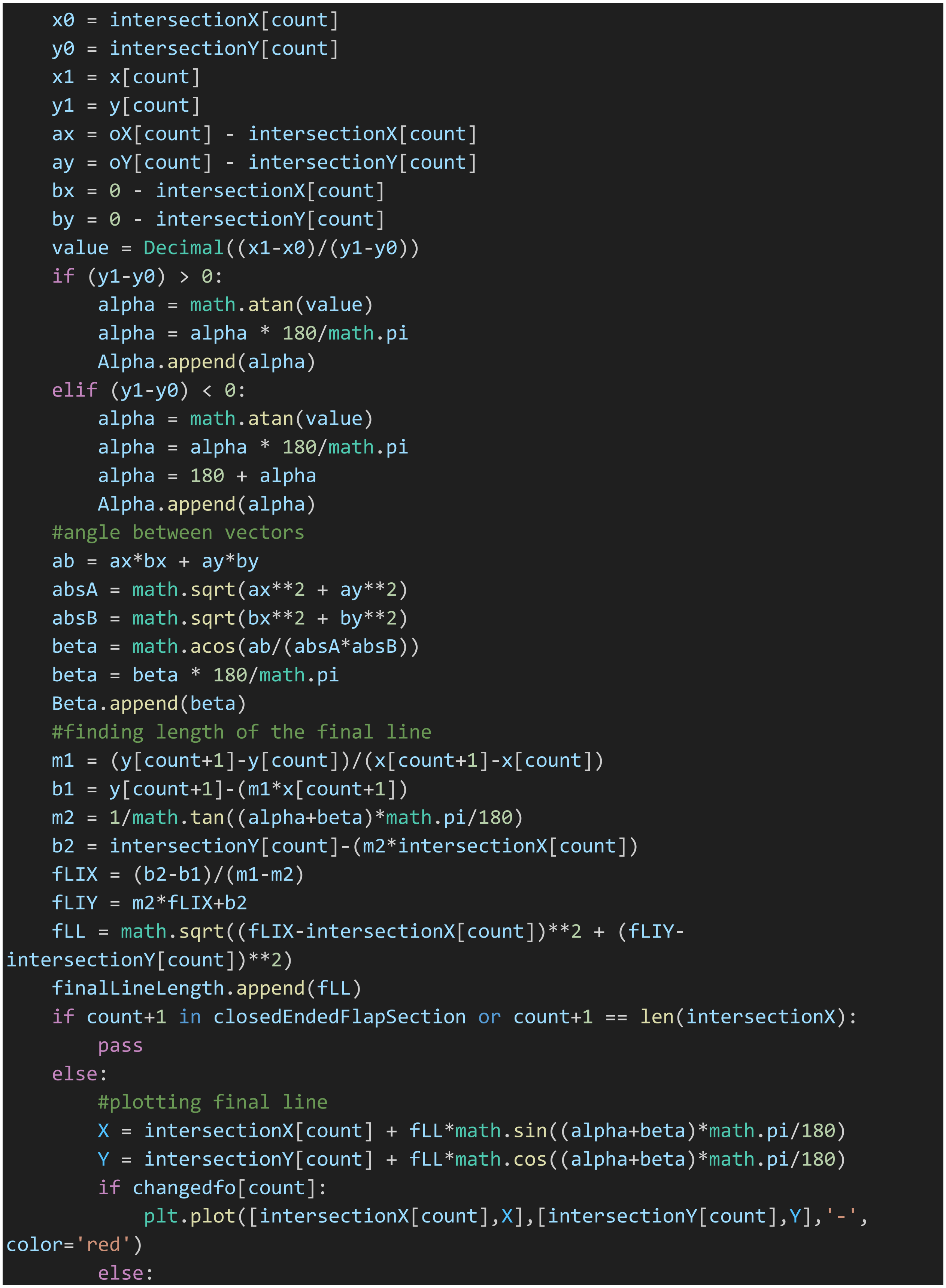
Appendix 10. The Angle-Ray computer program coded on Visual Studio Code



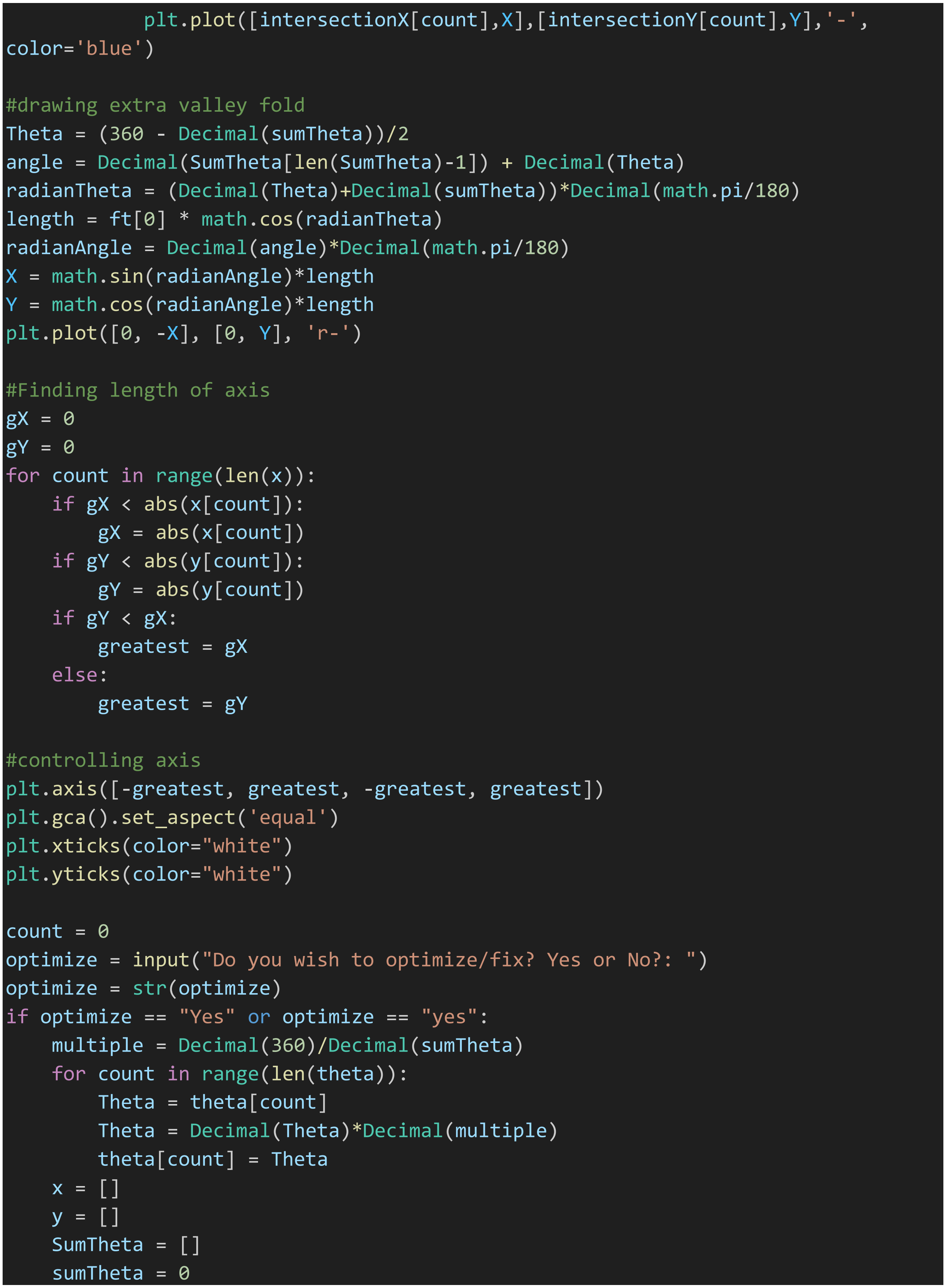
```
import math
import numpy as np
from matplotlib import pyplot as plt
from decimal import Decimal

#inputs
def totalFlapCount():
    global foTotal
    foTotal = int(input("Total number of Open-ended flaps: "))
totalFlapCount()
#collecting input
ft = list(map(float, input("Total flap lengths: ").split()))
fo = list(map(float, input("Open-ended flap lengths: ").split()))
c = list(map(float, input("Connectivity values: ").split()))
originalCValues = c
closedEndedFlapSection = list(map(int, input("Position of Closed-ended flap
sections: ").split()))
#recalibration of c value for accurate connectivity output when folded
if closedEndedFlapSection == [0]:
    pass
else:
    for count in range(len(closedEndedFlapSection)):
        ClosedEndedFlapSection = closedEndedFlapSection[count]
        length = (ft[ClosedEndedFlapSection-1] + ft[ClosedEndedFlapSection]) /
c[ClosedEndedFlapSection-1]
        C = (fo[ClosedEndedFlapSection-1] + fo[ClosedEndedFlapSection]) / length
        c[ClosedEndedFlapSection-1] = C

#fo - 0.00000001 when equal to ft
changedfo = []
for count in range(len(ft)):
    if ft[count] == fo[count]:
        fo[count] = fo[count]-10**-8
        changedfo.append(bool(True))
    else:
        changedfo.append(bool(False))


#calculation
theta = []
for count in range(len(ft)):
    fTotal1 = ft[count]
    fOpen1 = fo[count]
    if count+1 == len(ft):
        fTotal2 = ft[0]
    else:
        fTotal2 = ft[count+1]
    if count+1 == len(fo):
        fOpen2 = fo[0]
    else:
        fOpen2 = fo[count+1]
    connectivity = c[count]
    if connectivity == 0:
        theta.append(0)
    else:
        numerator = ((fTotal1**2)*(connectivity**2)) +
((fTotal2**2)*(connectivity**2)) - (fOpen1 + fOpen2)**2
        denominator = 2*fTotal1*fTotal2*connectivity**2
        value = min(1, max(-1, numerator/denominator))
        Theta = math.acos(value) * (180/math.pi)
        theta.append(Theta)

#sum of angles and optimality reply
SumTheta = []
sumTheta = 0
for count in range(len(theta)):
    sumTheta = sumTheta + theta[count]
    SumTheta.append(sumTheta)
if sumTheta > 360:
    print("Your specifications are impossible to fold. Unless it is
optimized/fixed.")
elif sumTheta == 360:
    print("Your specifications is at the highest optimization possible for this
method.")
elif sumTheta < 180:
    print("Your specifications is not foldable. Unless it is optimized/fixed.")
else:
    print("Your design is foldable, but is not of optimal design. Unless it is
optimized/fixed")


x = []
y = []

#need the initial line (vertical one in center)
x.append(0)
y.append(ft[0])
ft.append(ft[0])
count=1
```



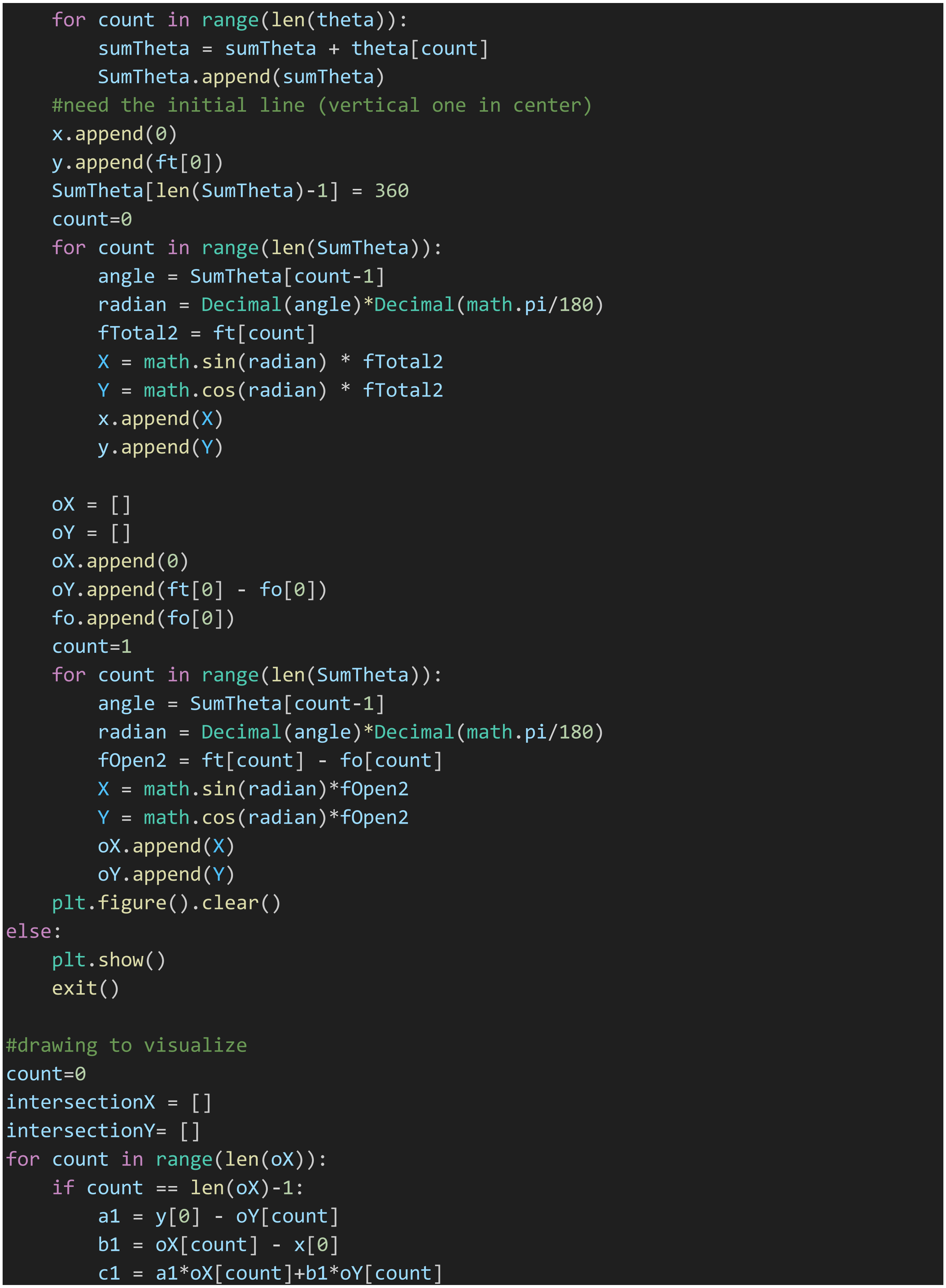
```
while count in range(len(ft)):
    angle = SumTheta[count-1]
    radian = angle*(math.pi/180)
    fTotal2 = ft[count]
    X = math.sin(radian) * fTotal2
    Y = math.cos(radian) * fTotal2
    x.append(X)
    y.append(Y)
    count+=1


oX = []
oY = []
oX.append(0)
oY.append(ft[0] - fo[0])
fo.append(fo[0])
count=1
SumTheta.append(360)
while count in range(len(fo)):
    angle = SumTheta[count-1]
    radian = angle*(math.pi/180)
    fOpen2 = ft[count] - fo[count]
    X = math.sin(radian)*fOpen2
    Y = math.cos(radian)*fOpen2
    oX.append(X)
    oY.append(Y)
    count+=1

#drawing to visualize
count=0
intersectionX = []
intersectionY= []
for count in range(len(oX)):
    if count == len(oX)-1:
        a1 = y[0] - oY[count]
        b1 = oX[count] - x[0]
        c1 = a1*oX[count]+b1*oY[count]
        a2 = oY[0]-y[count]
        b2 = x[count] - oX[0]
        c2 = a2*x[count]+b2*y[count]
        X = (b2*c1-b1*c2)/(a1*b2-a2*b1)
        Y = (a1*c2-a2*c1)/(a1*b2-a2*b1)
        intersectionX.append(X)
        intersectionY.append(Y)
    else:
        a1 = y[count+1] - oY[count]
        b1 = oX[count] - x[count+1]
        c1 = a1*oX[count]+b1*oY[count]
        a2 = oY[count+1]-y[count]
        b2 = x[count] - oX[count+1]
        c2 = a2*x[count]+b2*y[count]
        X = (b2*c1-b1*c2)/(a1*b2-a2*b1)
        Y = (a1*c2-a2*c1)/(a1*b2-a2*b1)
        intersectionX.append(X)
        intersectionY.append(Y)
    if count+1 in closedEndedFlapSection or count+1 == len(oX):
        pass
    else:
        if count == len(oX)-1:
            plt.plot([x[count], oX[0]], [y[count], oY[0]], 'r-', linewidth='1')
#left to right
            plt.plot([oX[count],x[0]], [oY[count],y[0]], 'r-', linewidth='1')
#right to left
            plt.plot([oX[count], X],[oY[count], Y],'-',
color='black',  linewidth='1',)
            plt.plot([X, oX[0]],[Y, oY[0]], '-', color='black', linewidth='1')
            plt.plot([0, X], [0, Y], 'r-', linewidth='1')
        else:
            plt.plot([x[count], oX[count+1]], [y[count], oY[count+1]], 'r-',
linewidth='1') #left to right
            plt.plot([oX[count],x[count+1]], [oY[count],y[count+1]], 'r-',
linewidth='1') #right to left
            plt.plot([oX[count], X],[oY[count], Y],'-',
color='black',  linewidth='1',)
            plt.plot([X, oX[count+1]],[Y, oY[count+1]], '-', color='black',
linewidth='1')
            plt.plot([0, X], [0, Y], 'r-', linewidth='1')
            plt.plot([0, X], [0, Y], 'r-', linewidth='1')
    x.append(0)
    y.append(ft[0])
    plt.plot([x[count],0], [y[count],0],'b-')
    plt.plot(x, y, 'b-')
    plt.plot([0,0],[0,ft[0]], 'b-')

#final line calculations
count = 0
Alpha = []
Beta = []
finalLineLength = []
for count in range(len(intersectionX)):
```



```
    x0 = intersectionX[count]
    y0 = intersectionY[count]
    x1 = x[count]
    y1 = y[count]
    ax = oX[count] - intersectionX[count]
    ay = oY[count] - intersectionY[count]
    bx = 0 - intersectionX[count]
    by = 0 - intersectionY[count]
    value = Decimal((x1-x0)/(y1-y0))
    if (y1-y0) > 0:
        alpha = math.atan(value)
        alpha = alpha * 180/math.pi
        Alpha.append(alpha)
    elif (y1-y0) < 0:
        alpha = math.atan(value)
        alpha = alpha * 180/math.pi
        alpha = 180 + alpha
        Alpha.append(alpha)
    #angle between vectors
    ab = ax*bx + ay*by
    absA = math.sqrt(ax**2 + ay**2)
    absB = math.sqrt(bx**2 + by**2)
    beta = math.acos(ab/(absA*absB))
    beta = beta * 180/math.pi
    Beta.append(beta)
    #finding length of the final line
    m1 = (y[count+1]-y[count])/(x[count+1]-x[count])
    b1 = y[count+1]-(m1*x[count+1])
    m2 = 1/math.tan((alpha+beta)*math.pi/180)
    b2 = intersectionY[count]-(m2*intersectionX[count])
    fLIX = (b2-b1)/(m1-m2)
    fLIY = m2*fLIX+b2
    fLL = math.sqrt((fLIX-intersectionX[count])**2 + (fLIY-
intersectionY[count])**2)
    finalLineLength.append(fLL)
    if count+1 in closedEndedFlapSection or count+1 == len(intersectionX):
        pass
    else:
        #plotting final line
        X = intersectionX[count] + fLL*math.sin((alpha+beta)*math.pi/180)
        Y = intersectionY[count] + fLL*math.cos((alpha+beta)*math.pi/180)
        if changedfo[count]:
            plt.plot([intersectionX[count],X],[intersectionY[count],Y],'-',
color='red')
        else:
```

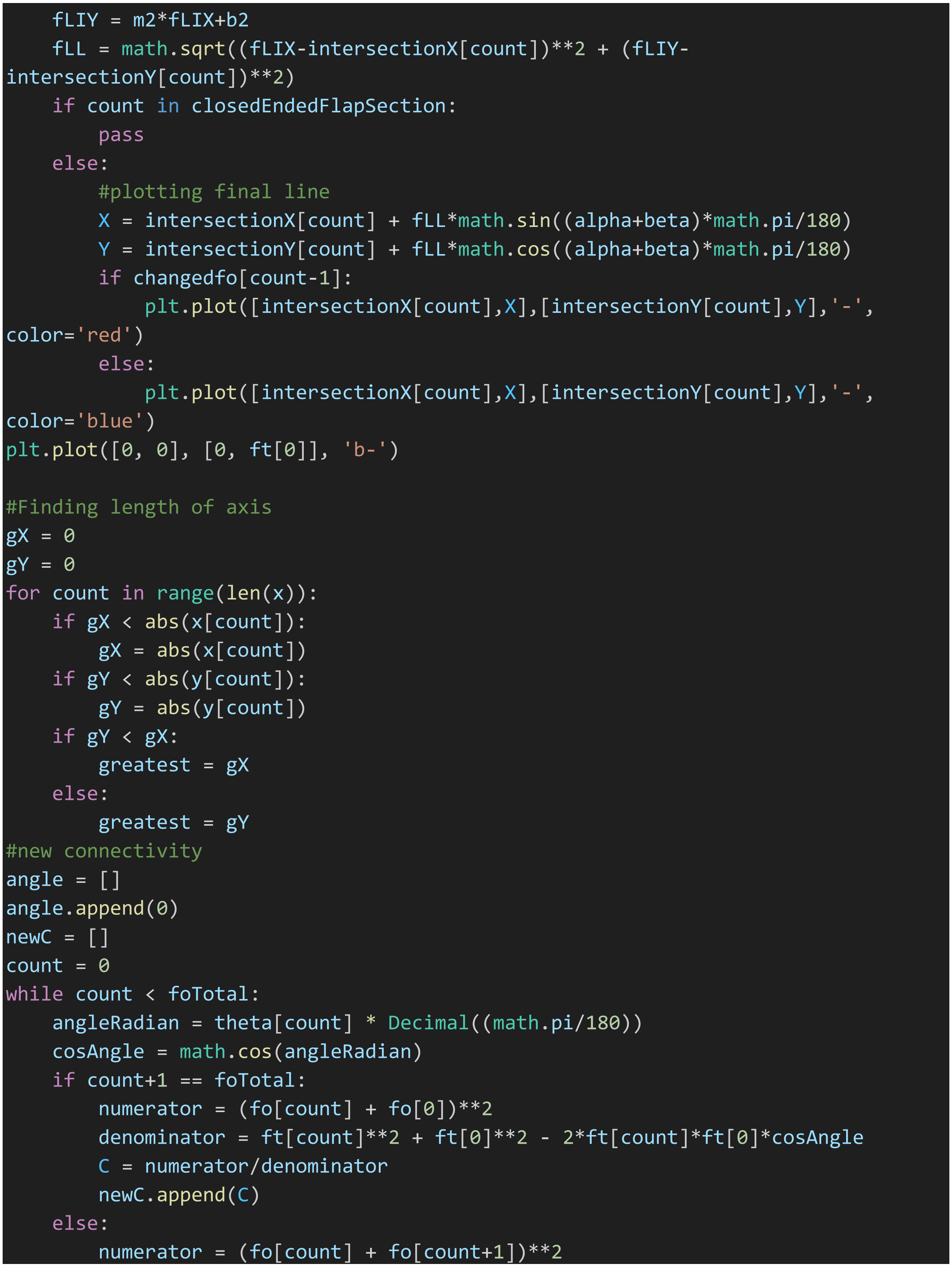


```
            plt.plot([intersectionX[count],X],[intersectionY[count],Y],'-',
color='blue')

#drawing extra valley fold
Theta = (360 - Decimal(sumTheta))/2
angle = Decimal(SumTheta[len(SumTheta)-1]) + Decimal(Theta)
radianTheta = (Decimal(Theta)+Decimal(sumTheta))*Decimal(math.pi/180)
length = ft[0] * math.cos(radianTheta)
radianAngle = Decimal(angle)*Decimal(math.pi/180)
X = math.sin(radianAngle)*length
Y = math.cos(radianAngle)*length
plt.plot([0, -X], [0, Y], 'r-')

#Finding length of axis
gX = 0
gY = 0
for count in range(len(x)):
    if gX < abs(x[count]):
        gX = abs(x[count])
    if gY < abs(y[count]):
        gY = abs(y[count])
    if gY < gX:
        greatest = gX
    else:
        greatest = gY

#controlling axis
plt.axis([-greatest, greatest, -greatest, greatest])
plt.gca().set_aspect('equal')
plt.xticks(color="white")
plt.yticks(color="white")

count = 0
optimize = input("Do you wish to optimize/fix? Yes or No?: ")
optimize = str(optimize)
if optimize == "Yes" or optimize == "yes":
    multiple = Decimal(360)/Decimal(sumTheta)
    for count in range(len(theta)):
        Theta = theta[count]
        Theta = Decimal(Theta)*Decimal(multiple)
        theta[count] = Theta
    x = []
    y = []
    SumTheta = []
    sumTheta = 0
```

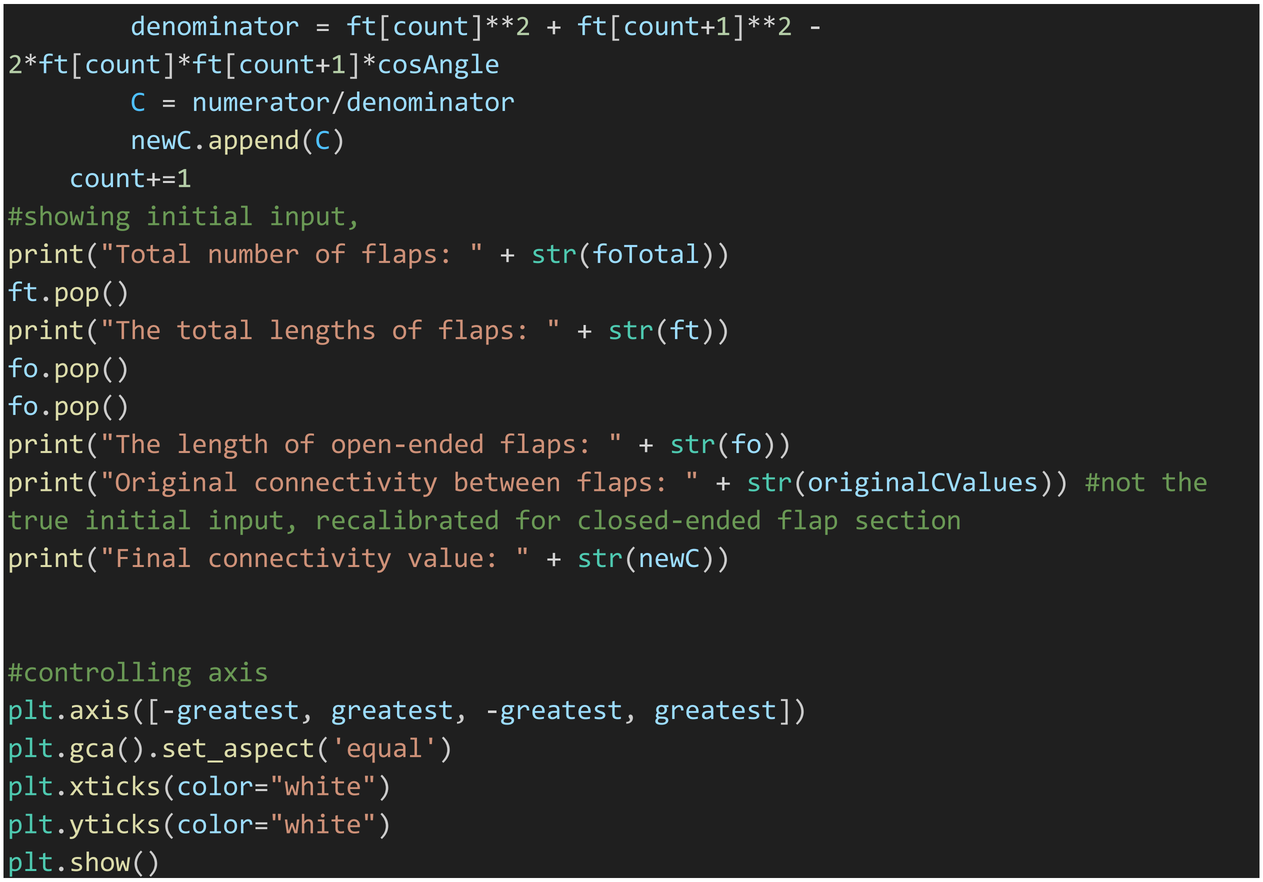


```
    for count in range(len(theta)):
        sumTheta = sumTheta + theta[count]
        SumTheta.append(sumTheta)
    #need the initial line (vertical one in center)
    x.append(0)
    y.append(ft[0])
    SumTheta[len(SumTheta)-1] = 360
    count=0
    for count in range(len(SumTheta)):
        angle = SumTheta[count-1]
        radian = Decimal(angle)*Decimal(math.pi/180)
        fTotal2 = ft[count]
        X = math.sin(radian) * fTotal2
        Y = math.cos(radian) * fTotal2
        x.append(X)
        y.append(Y)

    oX = []
    oY = []
    oX.append(0)
    oY.append(ft[0] - fo[0])
    fo.append(fo[0])
    count=1
    for count in range(len(SumTheta)):
        angle = SumTheta[count-1]
        radian = Decimal(angle)*Decimal(math.pi/180)
        fOpen2 = ft[count] - fo[count]
        X = math.sin(radian)*fOpen2
        Y = math.cos(radian)*fOpen2
        oX.append(X)
        oY.append(Y)
    plt.figure().clear()
else:
    plt.show()
    exit()

#drawing to visualize
count=0
intersectionX = []
intersectionY= []
for count in range(len(oX)):
    if count == len(oX)-1:
        a1 = y[0] - oY[count]
        b1 = oX[count] - x[0]
        c1 = a1*oX[count]+b1*oY[count]
        a2 = oY[0]-y[count]
        b2 = x[count] - oX[0]
        c2 = a2*x[count]+b2*y[count]
        X = (b2*c1-b1*c2)/(a1*b2-a2*b1)
        Y = (a1*c2-a2*c1)/(a1*b2-a2*b1)
        intersectionX.append(X)
        intersectionY.append(Y)
    else:
        a1 = y[count+1] - oY[count]
        b1 = oX[count] - x[count+1]
        c1 = a1*oX[count]+b1*oY[count]
        a2 = oY[count+1]-y[count]
        b2 = x[count] - oX[count+1]
        c2 = a2*x[count]+b2*y[count]
        X = (b2*c1-b1*c2)/(a1*b2-a2*b1)
        Y = (a1*c2-a2*c1)/(a1*b2-a2*b1)
        intersectionX.append(X)
        intersectionY.append(Y)
    if count in closedEndedFlapSection:
        pass
    else:
        if count == len(oX)-1:
            plt.plot([x[count], oX[0]], [y[count], oY[0]], 'r-', linewidth='1')
#left to right
            plt.plot([oX[count],x[0]], [oY[count],y[0]], 'r-', linewidth='1')
#right to left
            plt.plot([oX[count], X],[oY[count], Y],'-',
color='black',  linewidth='1',)
            plt.plot([X, oX[0]],[Y, oY[0]], '-', color='black', linewidth='1')
            plt.plot([0, X], [0, Y], 'r-', linewidth='1')
        else:
            plt.plot([x[count], oX[count+1]], [y[count], oY[count+1]], 'r-',
linewidth='1') #left to right
            plt.plot([oX[count],x[count+1]], [oY[count],y[count+1]], 'r-',
linewidth='1') #right to left
            plt.plot([oX[count], X],[oY[count], Y],'-',
color='black',  linewidth='1',)
            plt.plot([X, oX[count+1]],[Y, oY[count+1]], '-', color='black',
linewidth='1')
            plt.plot([0, X], [0, Y], 'r-', linewidth='1')
    x.append(0)
    y.append(ft[0])
    plt.plot([x[count],0], [y[count],0],'b-')
    plt.plot(x, y, 'b-')
    plt.plot([0,0],[0,ft[0]], 'b-')
```

```
#final line calculations
count = 0
Alpha = []
Beta = []
finalLineLength = []
for count in range(len(intersectionY)):
    x0 = intersectionX[count]
    y0 = intersectionY[count]
    x1 = x[count]
    y1 = y[count]
    ax = oX[count] - intersectionX[count]
    ay = oY[count] - intersectionY[count]
    bx = 0 - intersectionX[count]
    by = 0 - intersectionY[count]
    value = Decimal((x1-x0)/(y1-y0))
    if (y1-y0) > 0:
        alpha = math.atan(value)
        alpha = alpha * 180/math.pi
        Alpha.append(alpha)
    elif (y1-y0) < 0:
        alpha = math.atan(value)
        alpha = alpha * 180/math.pi
        alpha = 180 + alpha
        Alpha.append(alpha)
    #angle between vectors
    ab = ax*bx + ay*by
    absA = math.sqrt(ax**2 + ay**2)
    absB = math.sqrt(bx**2 + by**2)
    beta = math.acos(ab/(absA*absB))
    beta = beta * 180/math.pi
    Beta.append(beta)
    if count == len(x)-1:
        #finding length of the final line
        m1 = (y[0]-y[count])/(x[0]-x[count])
        b1 = y[0]-(m1*x[0])
        m2 = 1/math.tan((alpha+beta)*math.pi/180)
        b2 = intersectionY[count]-(m2*intersectionX[count])
    else:
        #finding length of the final line
        m1 = (y[count+1]-y[count])/(x[count+1]-x[count])
        b1 = y[count+1]-(m1*x[count+1])
        m2 = 1/math.tan((alpha+beta)*math.pi/180)
        b2 = intersectionY[count]-(m2*intersectionX[count])
    fLIX = (b2-b1)/(m1-m2)
```



```
    fLIY = m2*fLIX+b2
    fLL = math.sqrt((fLIX-intersectionX[count])**2 + (fLIY-
intersectionY[count])**2)
    if count in closedEndedFlapSection:
        pass
    else:
        #plotting final line
        X = intersectionX[count] + fLL*math.sin((alpha+beta)*math.pi/180)
        Y = intersectionY[count] + fLL*math.cos((alpha+beta)*math.pi/180)
        if changedfo[count-1]:
            plt.plot([intersectionX[count],X],[intersectionY[count],Y],'-',
color='red')
        else:
            plt.plot([intersectionX[count],X],[intersectionY[count],Y],'-',
color='blue')
plt.plot([0, 0], [0, ft[0]], 'b-')

#Finding length of axis
gX = 0
gY = 0
for count in range(len(x)):
    if gX < abs(x[count]):
        gX = abs(x[count])
    if gY < abs(y[count]):
        gY = abs(y[count])
    if gY < gX:
        greatest = gX
    else:
        greatest = gY
#new connectivity
angle = []
angle.append(0)
newC = []
count = 0
while count < foTotal:
    angleRadian = theta[count] * Decimal((math.pi/180))
    cosAngle = math.cos(angleRadian)
    if count+1 == foTotal:
        numerator = (fo[count] + fo[0])**2
        denominator = ft[count]**2 + ft[0]**2 - 2*ft[count]*ft[0]*cosAngle
        C = numerator/denominator
        newC.append(C)
    else:
        numerator = (fo[count] + fo[count+1])**2
```



```
        denominator = ft[count]**2 + ft[count+1]**2 -
2*ft[count]*ft[count+1]*cosAngle
        C = numerator/denominator
        newC.append(C)
    count+=1
#showing initial input,
print("Total number of flaps: " + str(foTotal))
ft.pop()
print("The total lengths of flaps: " + str(ft))
fo.pop()
fo.pop()
print("The length of open-ended flaps: " + str(fo))
print("Original connectivity between flaps: " + str(originalCValues)) #not the
true initial input, recalibrated for closed-ended flap section
print("Final connectivity value: " + str(newC))


#controlling axis
plt.axis([-greatest, greatest, -greatest, greatest])
plt.gca().set_aspect('equal')
plt.xticks(color="white")
plt.yticks(color="white")
plt.show()
```

Appendix 11. User guide on the Angle-Ray computer program

*The program was coded on Visual Studio code and has only been tested on this program.

1. Once the program is started the following texts will be printed.
2. "Total number of Open-ended flaps:", input the total number of flaps desired as an integer and press enter.
3. "Total flap lengths:", input all the total flap lengths in a clockwise order as float values. Remember to press space between every input and press enter once completed.
4. "Open-ended flap lengths:", input the open-ended flap lengths in the same order as the total flap lengths (Step 3) as float values. Remember to press space between every input and press enter once completed.
5. "Connectivity values:", input the connectivity between the flaps in the same order as the total flap lengths (Step 3) as float values. Remember to press space between every input and press enter once completed.
6. "Position of Closed-ended flaps sections:", input the flap number where a closed-ended flap section starts when going clockwise. The flap number is associated with the order of inputs from Steps 3, 4, and 5. Remember to press space between every input and press enter once completed. If no closed-ended flaps are desired input 0.
7. If the initial inputs are not foldable you will be asked to optimize/fix the design. When answering "yes" or "Yes" an optimized/fixed crease pattern appears together with a crease pattern with initial inputs, otherwise only a crease pattern with the initial inputs appears.
8. Once the crease pattern appears take a screenshot of it and paste it onto a document. From here you can rotate and enlarge the crease pattern to any paper shape for higher efficiency, or adjust the size to a desired one.
9. Lastly print out the document (Step 8) and fold. The blue crease lines are mountain folds, and the red crease lines are valley folds. The rest of the crease patterns follow naturally when folding.

Appendix 12. Explanation of the Angle-Ray computer program

1. Requesting the user to input all necessary values, (Appendix 13, colored in yellow) listed below.
    a. Total number of open-ended flaps (foTotal).
    b. The total flap lengths (ft).
        i. Which is recorded onto a list.
    c. The open-ended flap lengths (fo).
        i. Which is recorded onto a list.
        ii. Whenever $f_o = f_t$ on the same flap Equation 7 is executed.

$$f_{of} = f_{oi} - 10^{-8}$$

f_{oi} = initial length of the open-ended flap.

f_{of} = final length of the open-ended flap.

Equation 7. Equation to avoid Zero Division Errors when $f_o = f_t$

The calculation is done to avoid Zero Division Errors in the modeling system when $f_o = f_t$. The subtraction value is the smallest value possible to subtract without causing a Zero Division Error on Visual Studio. The value is also small enough to not influence the final crease pattern thanks to overlap of crease lines.

    d. The connectivity between the open-ended flaps (c)
        i. Which is recorded onto a list.
        ii. For closed-ended flap sections the total flaps are interpreted as open-ended flaps.
    e. The position of closed-ended flaps.
        i. If between flaps 2 and 3 the position is 2.
        ii. Based on the position the computer program recalibrates the inputted connectivity for this segment using Equation 8 in terms of the open-ended flap length for application in later calculations.

$$l = \frac{f_{t1} + f_{t2}}{c_i}$$

$$c_f = \frac{f_{o1} + f_{o2}}{l}$$

l = length between the ends of the open-ended flaps, $l \neq 0$.

f_{t1} = total flap length of one flap.

f_{t2} = total flap length of the other neighboring flap.

f_{o1} = open-ended flap length of one flap.

f_{o2} = open-ended flap length of the other neighboring flap.

c_i = the initial connectivity between the total flap lengths, $c_i \neq 0$.

c_f = the new connectivity between the open-ended flaps to take account of the reinterpretation of the total flap as an open-ended flap

Equation 8. Closed-ended flap section connectivity reinterpretation

2. Calculations with Equation 5 are in a loop repeating for all flaps, moving clockwise (Appendix 13, colored in green).

$$\theta = \cos^{-1}\frac{{f_{t1}}^2c^2 + {f_{t2}}^2c^2 - (f_{o1} + f_{o2})^2}{2f_{t1}f_{t2}c^2}$$

$$c, f_{t1}, f_{t2} \neq 0$$

3. Using inputs and angles calculated in Step 2 the program draws a crease pattern based on the method from the third stage (Appendix 13, colored in light blue).
   a. The crease pattern is drawn using linear functions, vector calculations, and Matplotlib treating it as a graph.
4. However, for some values the output of Step 3 is not foldable or not efficient. This is reported to the user, and they will get an option to optimize/fix their design which alters connectivity values (Appendix 13, colored in pink).
   a. Crease patterns are not foldable when $\sum\theta > 360$ or $\sum\theta < 180$
5. If the user answers "yes" or "Yes" in Step 4 the computer program will run all output of Equation 5 calculated in Step 2 through Equation 6 recalibrating the angles (Appendix 13, colored in red).

$$\varphi = \theta * \frac{360}{\sum \theta}$$

$$\sum \theta \neq 0$$

If the user answers anything other than “yes” or “Yes” they will be presented with the crease pattern generated in Step 3 and the program terminates.

6. Following Step 5 the program draws a new crease pattern using the new angles (φ) from Step 5 and the user is presented with the new crease pattern and the initial crease pattern (Step 3) (Appendix 13¸ colored in gray).
7. Finally, the program prints out all the initial inputs together with the calibrated connectivity values and terminates.

Appendix 13. Color-coded computer program by segments

```
import math

import numpy as np

from matplotlib import pyplot as plt

from decimal import Decimal



#inputs

def totalFlapCount():

    global foTotal

    foTotal = int(input("Total number of Open-ended flaps: "))

totalFlapCount()

#collecting input

ft = list(map(float, input("Total flap lengths: ").split()))

fo = list(map(float, input("Open-ended flap lengths: ").split()))

c = list(map(float, input("Connectivity values: ").split()))

originalCValues = c

closedEndedFlapSection = list(map(int, input("Position of Closed-ended flap sections: ").split()))

#recalibration of c value for accurate connectivity output when folded

if closedEndedFlapSection == [0]:

    pass

else:

    for count in range(len(closedEndedFlapSection)):

        ClosedEndedFlapSection = closedEndedFlapSection[count]

        length = (ft[ClosedEndedFlapSection-1] + ft[ClosedEndedFlapSection]) / c[ClosedEndedFlapSection-1]
        C = (fo[ClosedEndedFlapSection-1] + fo[ClosedEndedFlapSection]) / length
        c[ClosedEndedFlapSection-1] = C


#fo - 0.00000001 when equal to ft
changedfo = []
for count in range(len(ft)):
    if ft[count] == fo[count]:
        fo[count] = fo[count]-10**-8
        changedfo.append(bool(True))
    else:
        changedfo.append(bool(False))



#calculation
theta = []
for count in range(len(ft)):
    fTotal1 = ft[count]
    fOpen1 = fo[count]
    if count+1 == len(ft):
        fTotal2 = ft[0]
    else:
        fTotal2 = ft[count+1]
    if count+1 == len(fo):
        fOpen2 = fo[0]
```

```
  else:
    fOpen2 = fo[count+1]
  connectivity = c[count]
  if connectivity == 0:
    theta.append(0)
  else:
    numerator = ((fTotal1**2)*(connectivity**2)) + ((fTotal2**2)*(connectivity**2)) - (fOpen1 + fOpen2)**2
    denominator = 2*fTotal1*fTotal2*connectivity**2
    value = min(1, max(-1, numerator/denominator))
    Theta = math.acos(value) * (180/math.pi)
    theta.append(Theta)

#sum of angles and optimality reply
SumTheta = []
sumTheta = 0
for count in range(len(theta)):
  sumTheta = sumTheta + theta[count]
  SumTheta.append(sumTheta)
if sumTheta > 360:
  print("Your specifications are impossible to fold. Unless it is optimized/fixed.")
elif sumTheta == 360:
  print("Your specifications is at the highest optimization possible for this method.")
elif sumTheta < 180:
  print("Your specifications is not foldable. Unless it is optimized/fixed.")
```

```
else:
    print("Your design is foldable, but is not of optimal design. Unless it is optimized/fixed")



x = []
y = []

#need the initial line (vertical one in center)
x.append(0)
y.append(ft[0])
ft.append(ft[0])
count=1
while count in range(len(ft)):
    angle = SumTheta[count-1]
    radian = angle*(math.pi/180)
    fTotal2 = ft[count]
    X = math.sin(radian) * fTotal2
    Y = math.cos(radian) * fTotal2
    x.append(X)
    y.append(Y)
    count+=1



oX = []
```

```
oY = []
oX.append(0)
oY.append(ft[0] - fo[0])
fo.append(fo[0])
count=1
SumTheta.append(360)
while count in range(len(fo)):
    angle = SumTheta[count-1]
    radian = angle*(math.pi/180)
    fOpen2 = ft[count] - fo[count]
    X = math.sin(radian)*fOpen2
    Y = math.cos(radian)*fOpen2
    oX.append(X)
    oY.append(Y)
    count+=1

#drawing to visualize
count=0
intersectionX = []
intersectionY= []
for count in range(len(oX)):
    if count == len(oX)-1:
        a1 = y[0] - oY[count]
        b1 = oX[count] - x[0]
        c1 = a1*oX[count]+b1*oY[count]
        a2 = oY[0]-y[count]
        b2 = x[count] - oX[0]
        c2 = a2*x[count]+b2*y[count]
        X = (b2*c1-b1*c2)/(a1*b2-a2*b1)
        Y = (a1*c2-a2*c1)/(a1*b2-a2*b1)
        intersectionX.append(X)
        intersectionY.append(Y)
    else:
        a1 = y[count+1] - oY[count]
        b1 = oX[count] - x[count+1]
        c1 = a1*oX[count]+b1*oY[count]
        a2 = oY[count+1]-y[count]
        b2 = x[count] - oX[count+1]
        c2 = a2*x[count]+b2*y[count]
        X = (b2*c1-b1*c2)/(a1*b2-a2*b1)
        Y = (a1*c2-a2*c1)/(a1*b2-a2*b1)
        intersectionX.append(X)
        intersectionY.append(Y)
    if count+1 in closedEndedFlapSection or count+1 == len(oX):
        pass
    else:
        if count == len(oX)-1:
            plt.plot([x[count], oX[0]], [y[count], oY[0]], 'r-', linewidth='1') #left to right
```

```
      plt.plot([oX[count],x[0]], [oY[count],y[0]], 'r-', linewidth='1') #right to left
      plt.plot([oX[count], X],[oY[count], Y],'-', color='black',  linewidth='1',)
      plt.plot([X, oX[0]],[Y, oY[0]], '-', color='black', linewidth='1')
      plt.plot([0, X], [0, Y], 'r-', linewidth='1')
    else:
      plt.plot([x[count], oX[count+1]], [y[count], oY[count+1]], 'r-', linewidth='1') #left to right
      plt.plot([oX[count],x[count+1]], [oY[count],y[count+1]], 'r-', linewidth='1') #right to left
      plt.plot([oX[count], X],[oY[count], Y],'-', color='black',  linewidth='1',)
      plt.plot([X, oX[count+1]],[Y, oY[count+1]], '-', color='black', linewidth='1')
      plt.plot([0, X], [0, Y], 'r-', linewidth='1')
      plt.plot([0, X], [0, Y], 'r-', linewidth='1')
  x.append(0)
  y.append(ft[0])
  plt.plot([x[count],0], [y[count],0],'b-')
  plt.plot(x, y, 'b-')
  plt.plot([0,0],[0,ft[0]], 'b-')


#final line calculations
count = 0
Alpha = []
Beta = []
finalLineLength = []
for count in range(len(intersectionX)):
  x0 = intersectionX[count]
```

```
y0 = intersectionY[count]
x1 = x[count]
y1 = y[count]
ax = oX[count] - intersectionX[count]
ay = oY[count] - intersectionY[count]
bx = 0 - intersectionX[count]
by = 0 - intersectionY[count]
value = Decimal((x1-x0)/(y1-y0))
if (y1-y0) > 0:
    alpha = math.atan(value)
    alpha = alpha * 180/math.pi
    Alpha.append(alpha)
elif (y1-y0) < 0:
    alpha = math.atan(value)
    alpha = alpha * 180/math.pi
    alpha = 180 + alpha
    Alpha.append(alpha)
#angle between vectors
ab = ax*bx + ay*by
absA = math.sqrt(ax**2 + ay**2)
absB = math.sqrt(bx**2 + by**2)
beta = math.acos(ab/(absA*absB))
beta = beta * 180/math.pi
Beta.append(beta)
```

```
    #finding length of the final line
    m1 = (y[count+1]-y[count])/(x[count+1]-x[count])
    b1 = y[count+1]-(m1*x[count+1])
    m2 = 1/math.tan((alpha+beta)*math.pi/180)
    b2 = intersectionY[count]-(m2*intersectionX[count])
    fLIX = (b2-b1)/(m1-m2)
    fLIY = m2*fLIX+b2
    fLL = math.sqrt((fLIX-intersectionX[count])**2 + (fLIY-intersectionY[count])**2)
    finalLineLength.append(fLL)
    if count+1 in closedEndedFlapSection or count+1 == len(intersectionX):
        pass
    else:
        #plotting final line
        X = intersectionX[count] + fLL*math.sin((alpha+beta)*math.pi/180)
        Y = intersectionY[count] + fLL*math.cos((alpha+beta)*math.pi/180)
        if changedfo[count]:
            plt.plot([intersectionX[count],X],[intersectionY[count],Y],'-', color='red')
        else:
            plt.plot([intersectionX[count],X],[intersectionY[count],Y],'-', color='blue')


#drawing extra valley fold
Theta = (360 - Decimal(sumTheta))/2
angle = Decimal(SumTheta[len(SumTheta)-1]) + Decimal(Theta)
radianTheta = (Decimal(Theta)+Decimal(sumTheta))*Decimal(math.pi/180)
```

```
length = ft[0] * math.cos(radianTheta)
radianAngle = Decimal(angle)*Decimal(math.pi/180)
X = math.sin(radianAngle)*length
Y = math.cos(radianAngle)*length
plt.plot([0, -X], [0, Y], 'r-')

#Finding length of axis
gX = 0
gY = 0
for count in range(len(x)):
    if gX < abs(x[count]):
        gX = abs(x[count])
    if gY < abs(y[count]):
        gY = abs(y[count])
    if gY < gX:
        greatest = gX
    else:
        greatest = gY

#controlling axis
plt.axis([-greatest, greatest, -greatest, greatest])
plt.gca().set_aspect('equal')
plt.xticks(color="white")
plt.yticks(color="white")
```

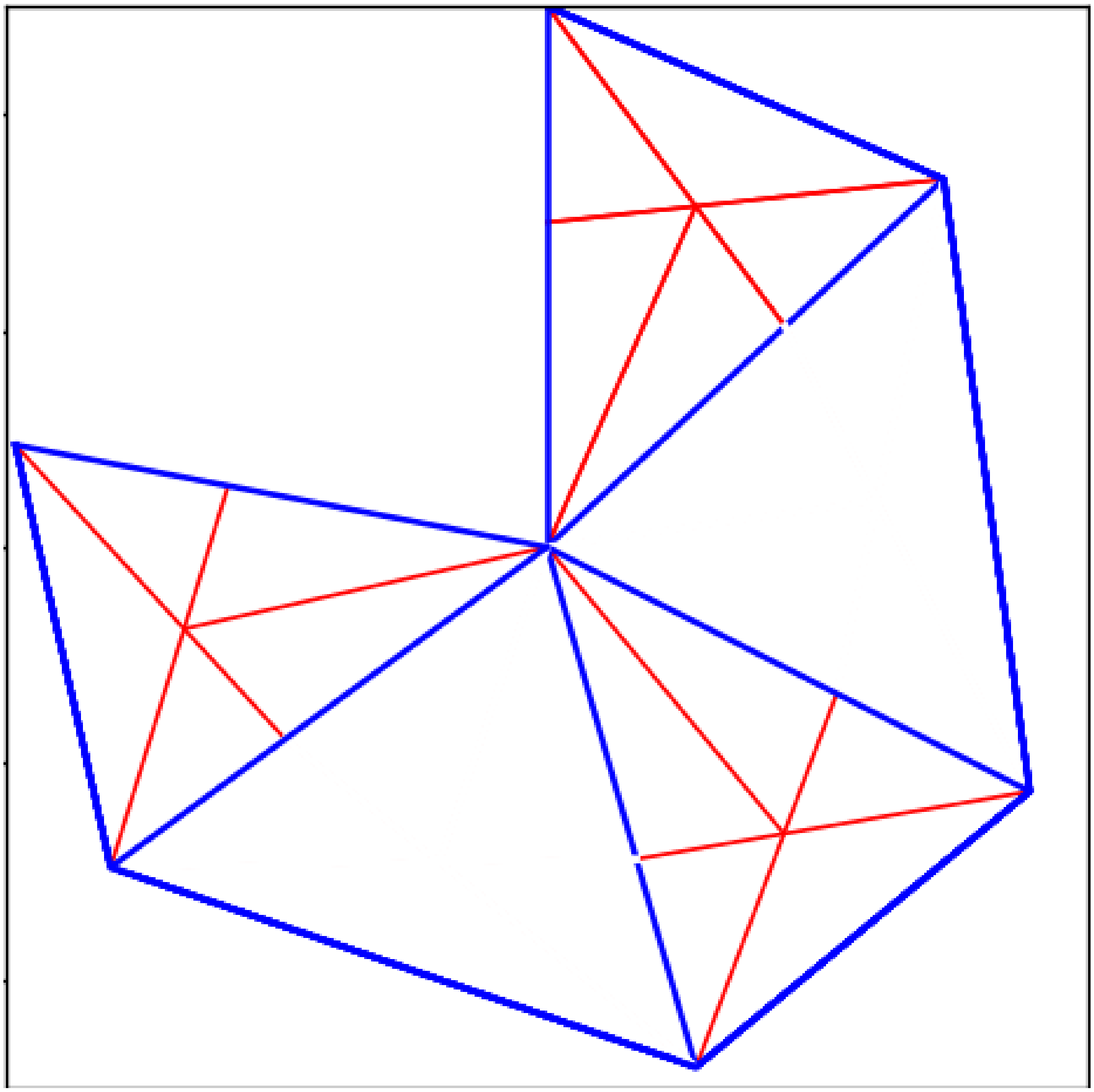
```
count = 0
optimize = input("Do you wish to optimize/fix? Yes or No?: ")
optimize = str(optimize)
if optimize == "Yes" or optimize == "yes":
    multiple = Decimal(360)/Decimal(sumTheta)
    for count in range(len(theta)):
        Theta = theta[count]
        Theta = Decimal(Theta)*Decimal(multiple)
        theta[count] = Theta
    x = []
    y = []
    SumTheta = []
    sumTheta = 0
    for count in range(len(theta)):
        sumTheta = sumTheta + theta[count]
        SumTheta.append(sumTheta)
    #need the initial line (vertical one in center)
    x.append(0)
    y.append(ft[0])
    SumTheta[len(SumTheta)-1] = 360
    count=0
    for count in range(len(SumTheta)):
        angle = SumTheta[count-1]
        radian = Decimal(angle)*Decimal(math.pi/180)
        fTotal2 = ft[count]
        X = math.sin(radian) * fTotal2
        Y = math.cos(radian) * fTotal2
        x.append(X)
        y.append(Y)

    oX = []
    oY = []
    oX.append(0)
    oY.append(ft[0] - fo[0])
    fo.append(fo[0])
    count=1
    for count in range(len(SumTheta)):
        angle = SumTheta[count-1]
        radian = Decimal(angle)*Decimal(math.pi/180)
        fOpen2 = ft[count] - fo[count]
        X = math.sin(radian)*fOpen2
        Y = math.cos(radian)*fOpen2
        oX.append(X)
        oY.append(Y)
    plt.figure().clear()
else:
    plt.show()
    exit()


#drawing to visualize
count=0
intersectionX = []
intersectionY= []
for count in range(len(oX)):
    if count == len(oX)-1:
        a1 = y[0] - oY[count]
        b1 = oX[count] - x[0]
        c1 = a1*oX[count]+b1*oY[count]
        a2 = oY[0]-y[count]
        b2 = x[count] - oX[0]
        c2 = a2*x[count]+b2*y[count]
        X = (b2*c1-b1*c2)/(a1*b2-a2*b1)
        Y = (a1*c2-a2*c1)/(a1*b2-a2*b1)
        intersectionX.append(X)
        intersectionY.append(Y)
    else:
        a1 = y[count+1] - oY[count]
        b1 = oX[count] - x[count+1]
        c1 = a1*oX[count]+b1*oY[count]
        a2 = oY[count+1]-y[count]
        b2 = x[count] - oX[count+1]
        c2 = a2*x[count]+b2*y[count]

        X = (b2*c1-b1*c2)/(a1*b2-a2*b1)

        Y = (a1*c2-a2*c1)/(a1*b2-a2*b1)

        intersectionX.append(X)

        intersectionY.append(Y)

    if count in closedEndedFlapSection:

        pass

    else:

        if count == len(oX)-1:

            plt.plot([x[count], oX[0]], [y[count], oY[0]], 'r-', linewidth='1') #left to right

            plt.plot([oX[count],x[0]], [oY[count],y[0]], 'r-', linewidth='1') #right to left

            plt.plot([oX[count], X],[oY[count], Y],'-', color='black',  linewidth='1',)

            plt.plot([X, oX[0]],[Y, oY[0]], '-', color='black', linewidth='1')

            plt.plot([0, X], [0, Y], 'r-', linewidth='1')

        else:

            plt.plot([x[count], oX[count+1]], [y[count], oY[count+1]], 'r-', linewidth='1') #left to right

            plt.plot([oX[count],x[count+1]], [oY[count],y[count+1]], 'r-', linewidth='1') #right to left

            plt.plot([oX[count], X],[oY[count], Y],'-', color='black',  linewidth='1',)

            plt.plot([X, oX[count+1]],[Y, oY[count+1]], '-', color='black', linewidth='1')

            plt.plot([0, X], [0, Y], 'r-', linewidth='1')

    x.append(0)

    y.append(ft[0])

    plt.plot([x[count],0], [y[count],0],'b-')

    plt.plot(x, y, 'b-')
    plt.plot([0,0],[0,ft[0]], 'b-')


#final line calculations

count = 0

Alpha = []

Beta = []

finalLineLength = []

for count in range(len(intersectionY)):

    x0 = intersectionX[count]

    y0 = intersectionY[count]

    x1 = x[count]

    y1 = y[count]

    ax = oX[count] - intersectionX[count]

    ay = oY[count] - intersectionY[count]

    bx = 0 - intersectionX[count]

    by = 0 - intersectionY[count]

    value = Decimal((x1-x0)/(y1-y0))

    if (y1-y0) > 0:

        alpha = math.atan(value)

        alpha = alpha * 180/math.pi

        Alpha.append(alpha)

    elif (y1-y0) < 0:

        alpha = math.atan(value)

        alpha = alpha * 180/math.pi
        alpha = 180 + alpha

        Alpha.append(alpha)

    #angle between vectors

    ab = ax*bx + ay*by

    absA = math.sqrt(ax**2 + ay**2)

    absB = math.sqrt(bx**2 + by**2)

    beta = math.acos(ab/(absA*absB))

    beta = beta * 180/math.pi

    Beta.append(beta)

    if count == len(x)-1:

        #finding length of the final line

        m1 = (y[0]-y[count])/(x[0]-x[count])

        b1 = y[0]-(m1*x[0])

        m2 = 1/math.tan((alpha+beta)*math.pi/180)

        b2 = intersectionY[count]-(m2*intersectionX[count])

    else:

        #finding length of the final line

        m1 = (y[count+1]-y[count])/(x[count+1]-x[count])

        b1 = y[count+1]-(m1*x[count+1])

        m2 = 1/math.tan((alpha+beta)*math.pi/180)

        b2 = intersectionY[count]-(m2*intersectionX[count])

    fLIX = (b2-b1)/(m1-m2)

    fLIY = m2*fLIX+b2

    fLL = math.sqrt((fLIX-intersectionX[count])**2 + (fLIY-intersectionY[count])**2)
```

```
    if count in closedEndedFlapSection:
        pass
    else:
        #plotting final line
        X = intersectionX[count] + fLL*math.sin((alpha+beta)*math.pi/180)
        Y = intersectionY[count] + fLL*math.cos((alpha+beta)*math.pi/180)
        if changedfo[count-1]:
            plt.plot([intersectionX[count],X],[intersectionY[count],Y],'-', color='red')
        else:
            plt.plot([intersectionX[count],X],[intersectionY[count],Y],'-', color='blue')
plt.plot([0, 0], [0, ft[0]], 'b-')


#Finding length of axis
gX = 0
gY = 0
for count in range(len(x)):
    if gX < abs(x[count]):
        gX = abs(x[count])
    if gY < abs(y[count]):
        gY = abs(y[count])
    if gY < gX:
        greatest = gX
    else:
        greatest = gY
```

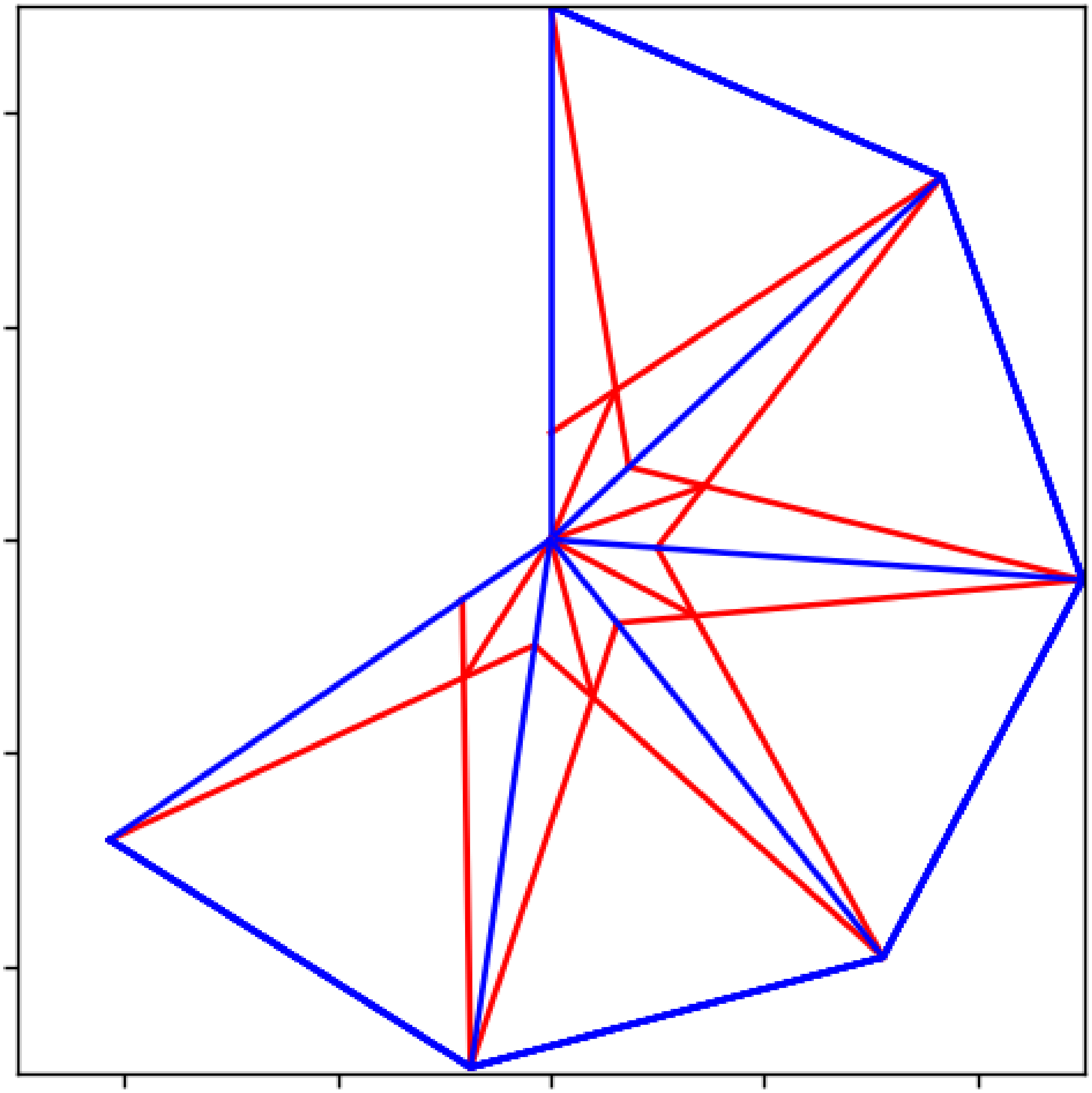
```
#new connectivity
angle = []
angle.append(0)
newC = []
count = 0
while count < foTotal:
   angleRadian = theta[count] * Decimal((math.pi/180))
   cosAngle = math.cos(angleRadian)
   if count+1 == foTotal:
      numerator = (fo[count] + fo[0])**2
      denominator = ft[count]**2 + ft[0]**2 - 2*ft[count]*ft[0]*cosAngle
      C = numerator/denominator
      newC.append(C)
   else:
      numerator = (fo[count] + fo[count+1])**2
      denominator = ft[count]**2 + ft[count+1]**2 - 2*ft[count]*ft[count+1]*cosAngle
      C = numerator/denominator
      newC.append(C)
   count+=1
#showing initial input,
print("Total number of flaps: " + str(foTotal))
ft.pop()
print("The total lengths of flaps: " + str(ft))
fo.pop()
fo.pop()
print("The length of open-ended flaps: " + str(fo))
print("Original connectivity between flaps: " + str(originalCValues)) #not the true initial value, perhaps fine. If bodered
print("Final connectivity value: " + str(newC))

#controlling axis
plt.axis([-greatest, greatest, -greatest, greatest])
plt.gca().set_aspect('equal')
plt.xticks(color="white")
plt.yticks(color="white")
plt.show()
```

Appendix 14. First test of the computer program testing with human inputs

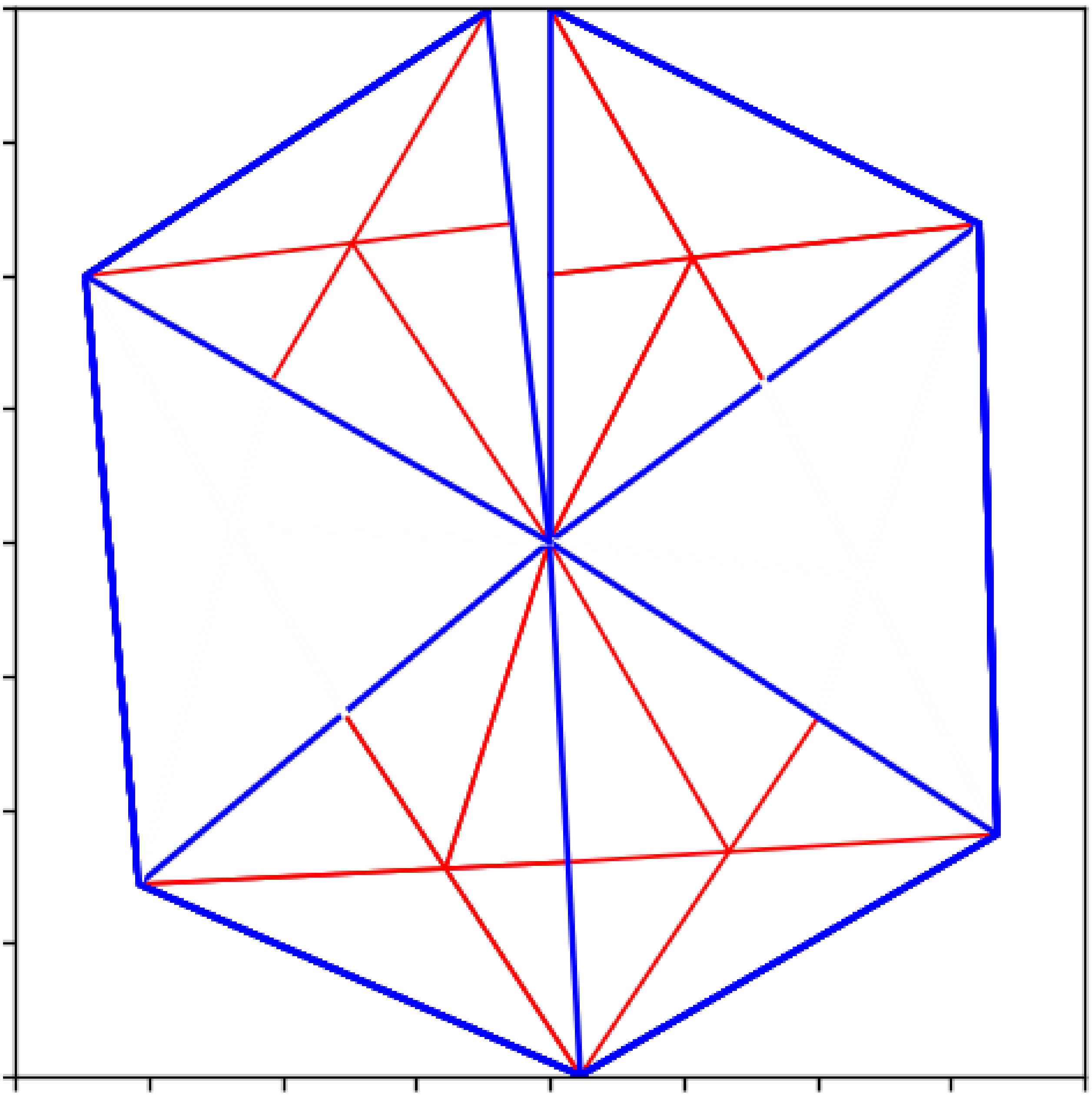
Not all tests were well-recorded, hence only a selection is attached



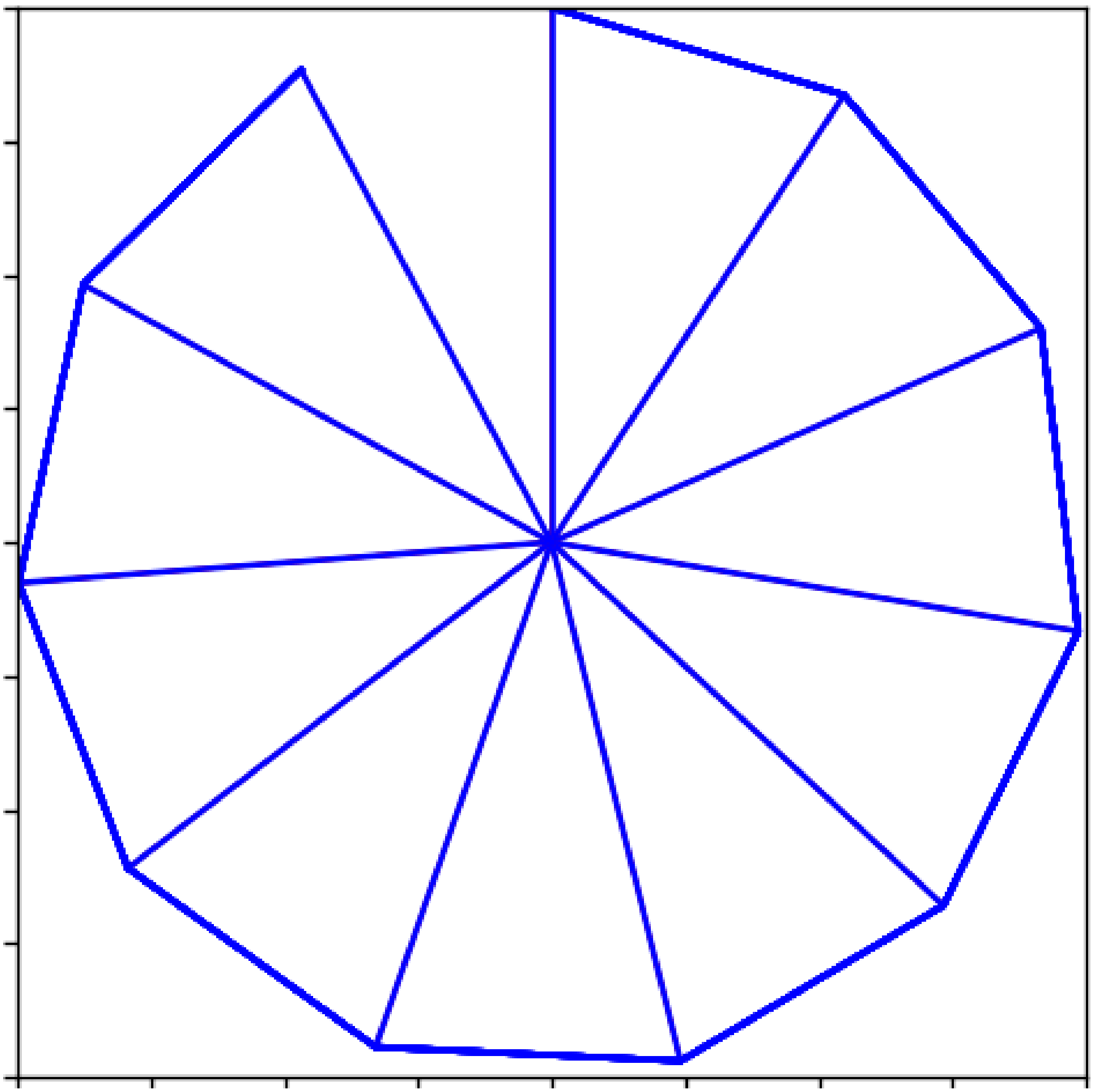
```
5

[5.0, 5.0, 5.0, 5.0, 5.0, 5.0]

[2.0, 2.0, 2.0, 2.0, 2.0, 2.0]

[1.0, 0.7, 1.0, 0.7, 1.0]

280.86868918539693
```



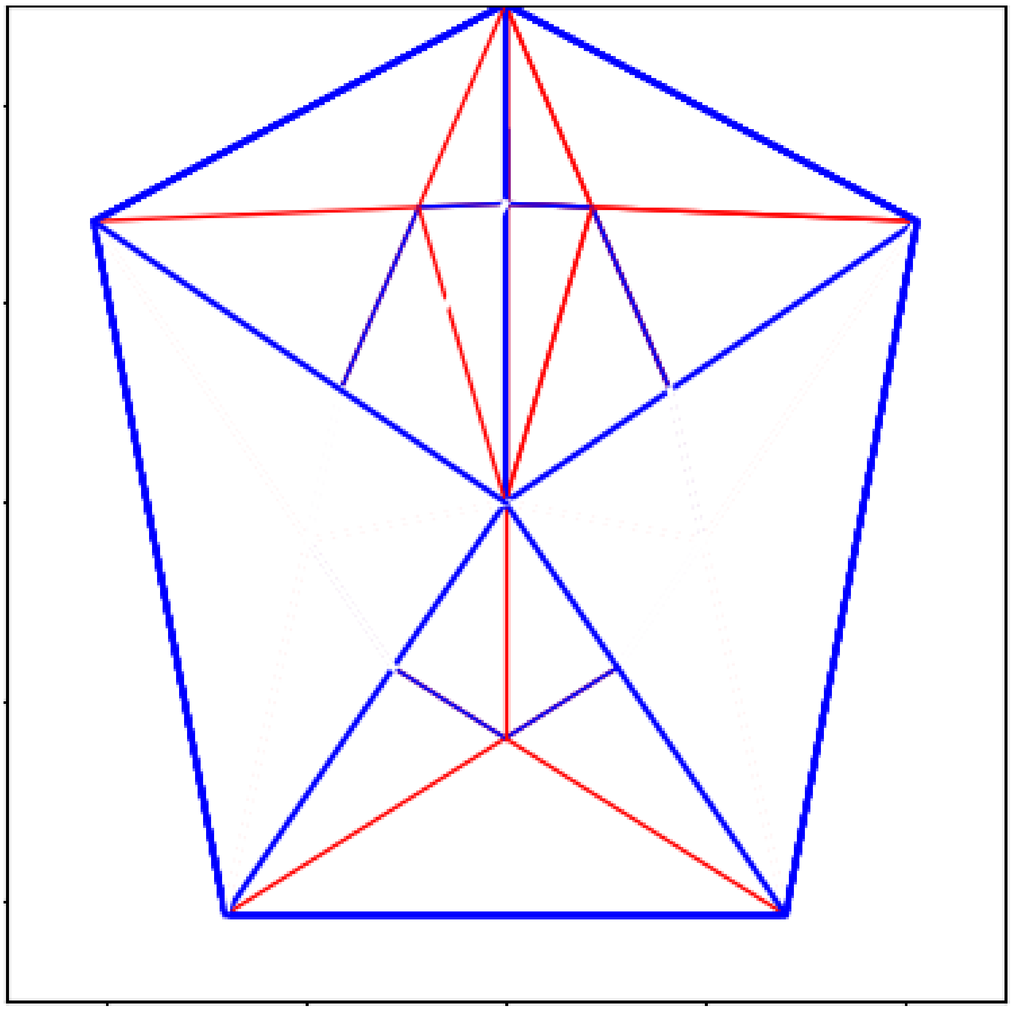
```
[5.0, 5.0, 5.0, 5.0, 5.0, 5.0]

[4.0, 4.0, 4.0, 4.0, 4.0, 4.0]

[2.0, 2.0, 2.0, 2.0, 2.0]
```



```
6

[10.0, 10.0, 10.0, 10.0, 10.0, 10.0, 10.0]

[5.0, 4.0, 4.0, 5.0, 4.0, 4.0, 5.0]


[1.0, 0.7, 1.0, 1.0, 0.7, 1.0]

353.34908991941006
```



```
10

[1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0]

[1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0]

[3.5, 3.5, 3.5, 3.5, 3.5, 3.5, 3.5, 3.5, 3.5, 3.5]

332.03099198040474
```



1.6

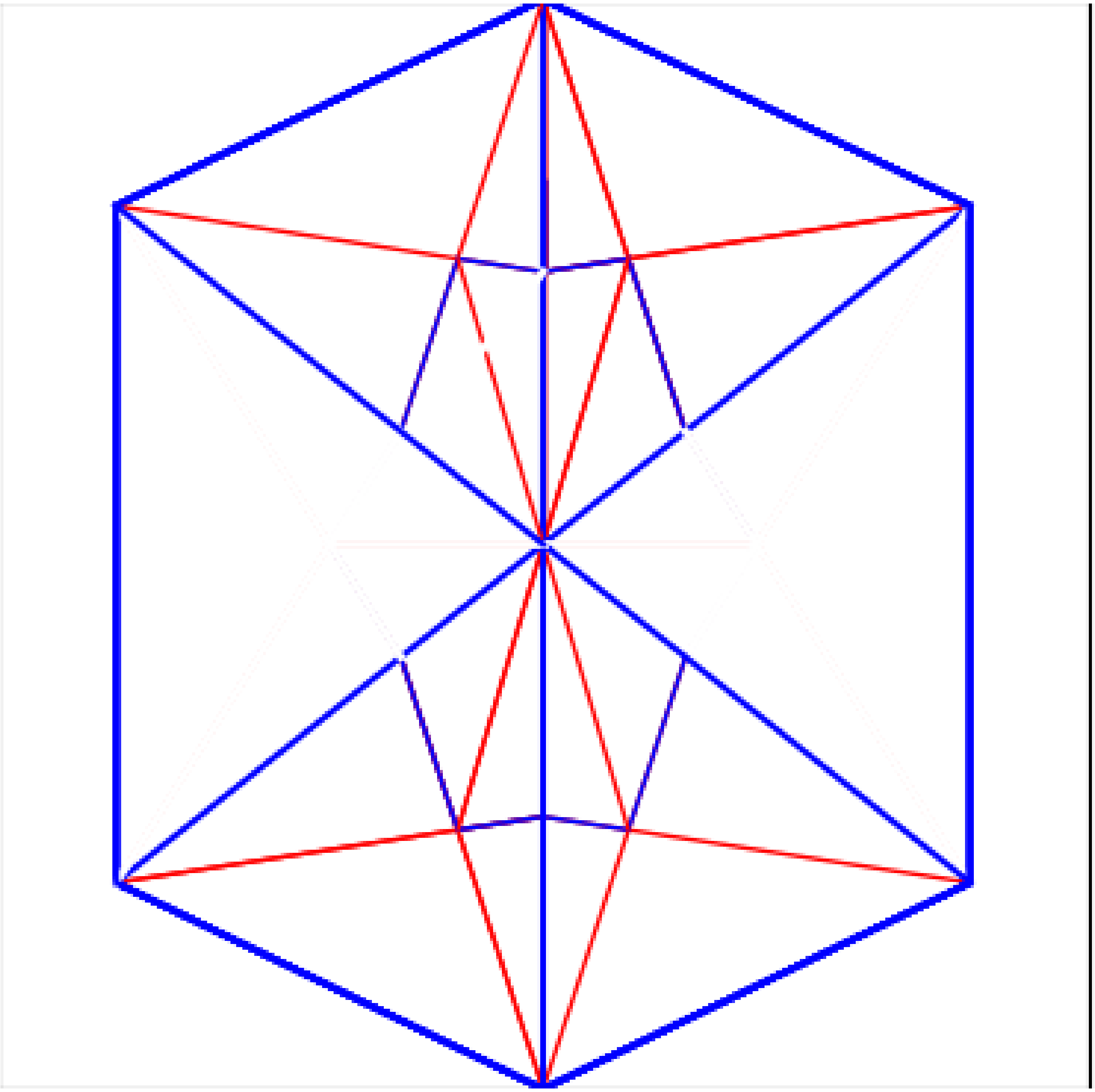
Total number of flaps: 5

The total lengths of flaps: [5.0, 5.0, 5.0, 5.0, 5.0]

The length of open-ended flaps: [2.0, 3.0, 3.0, 3.0, 3.0]

Original connectivity between flaps: [1.0, 1.0, 1.0, 1.0, 1.0]

Final connectivity value: [1.1472269810229754, 0.7181927942875258, 1.1394502211052069, 0.7181927942875258, 1.1472269810229754]



1.7

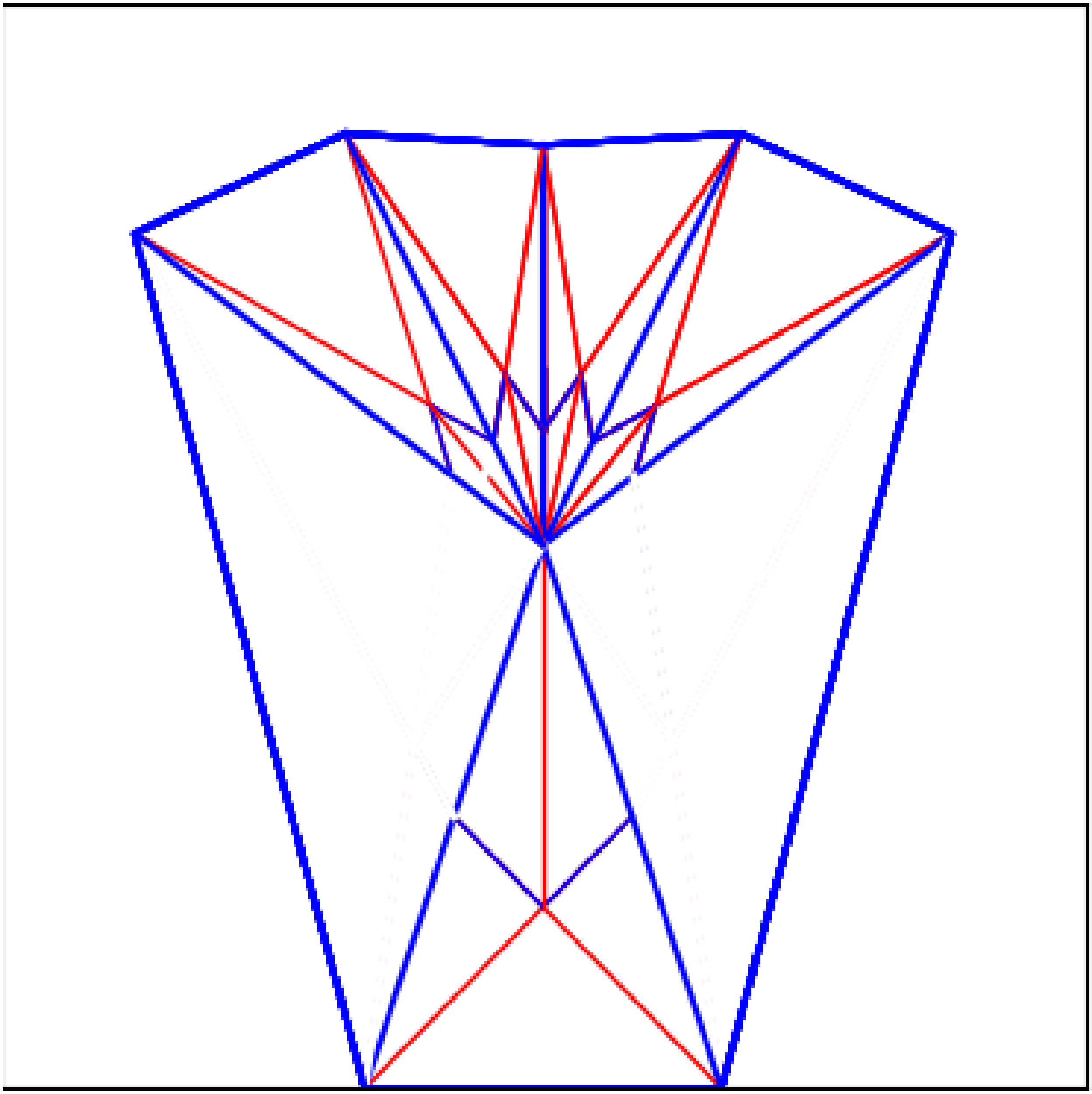
Total number of flaps: 6

The total lengths of flaps: [6.0, 6.0, 6.0, 6.0, 6.0, 6.0]

The length of open-ended flaps: [3.0, 4.0, 4.0, 3.0, 4.0, 4.0]

Original connectivity between flaps: [1.0, 2.0, 1.0, 1.0, 2.0, 1.0]

Final connectivity value: [1.7968198242362217, 1.15157571120294, 1.7968198242362217, 1.7968198242362217, 1.15157571120294, 1.7968198242362217]



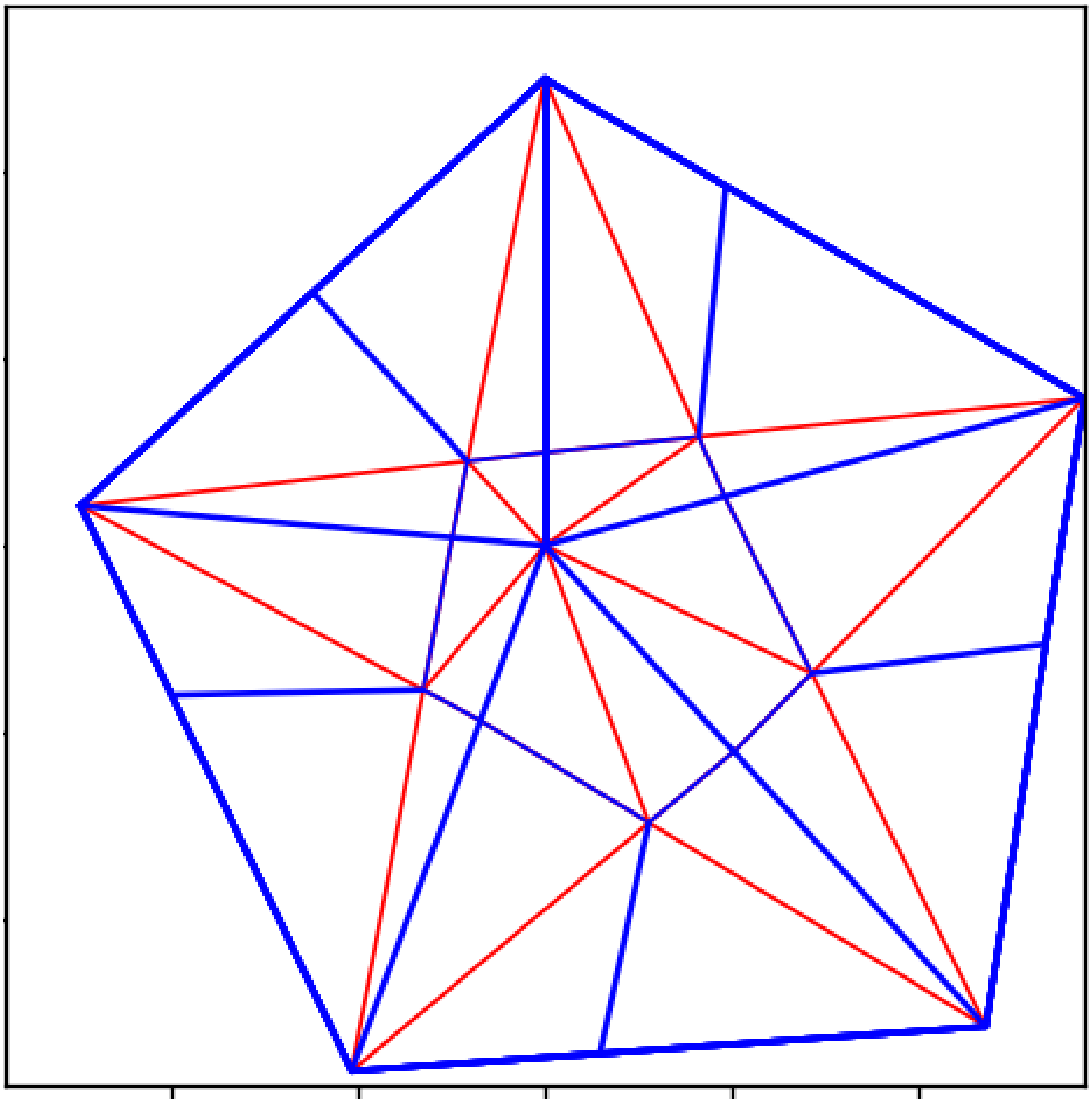
Total number of flaps: 7

The total lengths of flaps: [7.0, 8.0, 9.0, 10.0, 10.0, 9.0, 8.0]

The length of open-ended flaps: [5.0, 6.0, 7.0, 5.0, 5.0, 7.0, 6.0]

Original connectivity between flaps: [2.0, 2.0, 1.0, 1.0, 1.0, 2.0, 2.0]

Final connectivity value: [10.02359966086807, 10.143907382277579, 0.5998927888350684, 2.5650482517504, 0.5998927888350684, 10.143907382277579, 10.02359966086807]



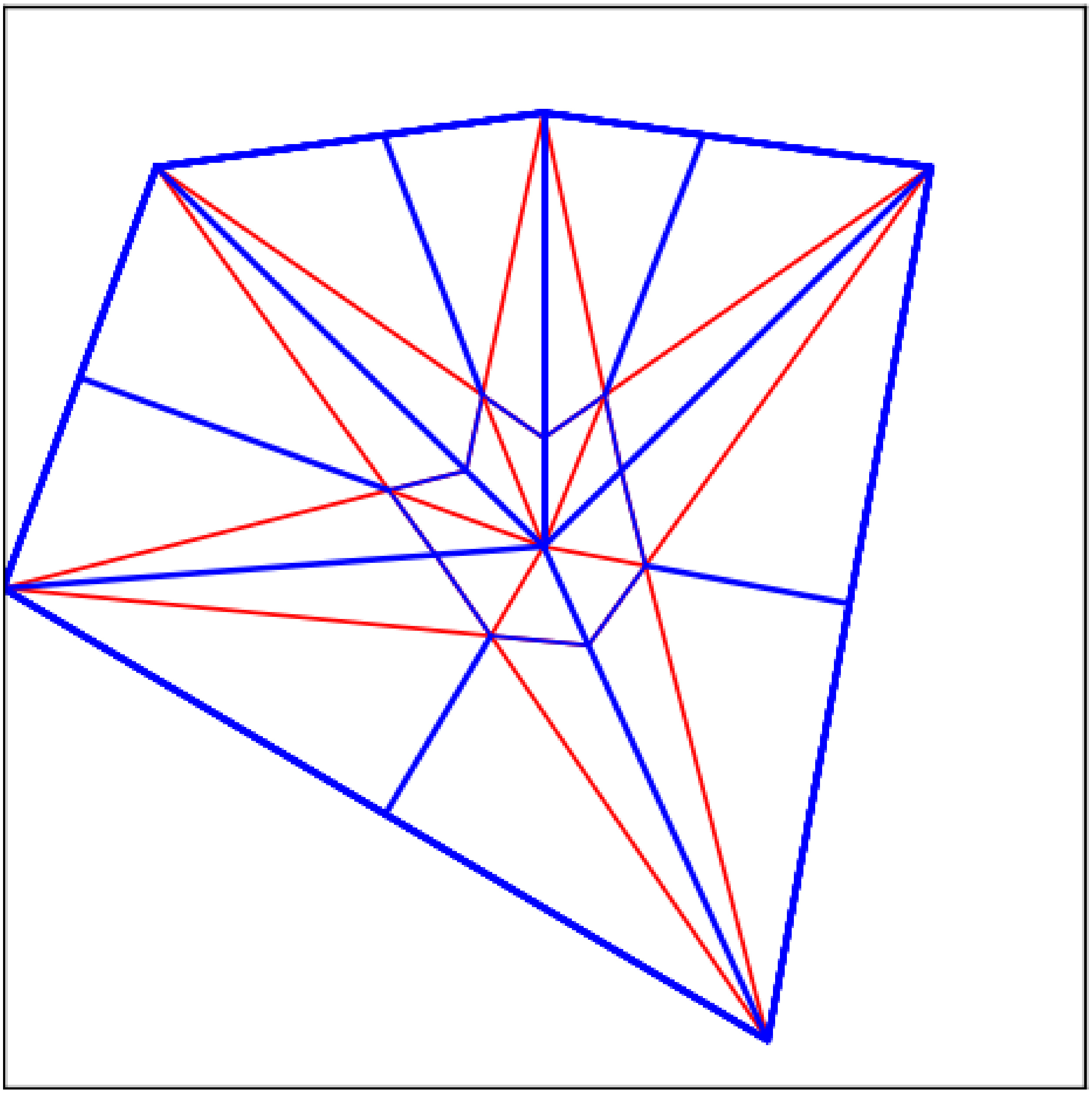
Total number of flaps: 5

The total lengths of flaps: [5.0, 6.0, 7.0, 6.0, 5.0]

The length of open-ended flaps: [4.0, 4.0, 4.0, 4.0, 4.0]

Original connectivity between flaps: [2.0, 2.0, 2.0, 2.0, 2.0]

Final connectivity value: [1.4164112101142428, 1.3755789187085947, 1.3755789187085947, 1.4164112101142428, 1.399921022532155]



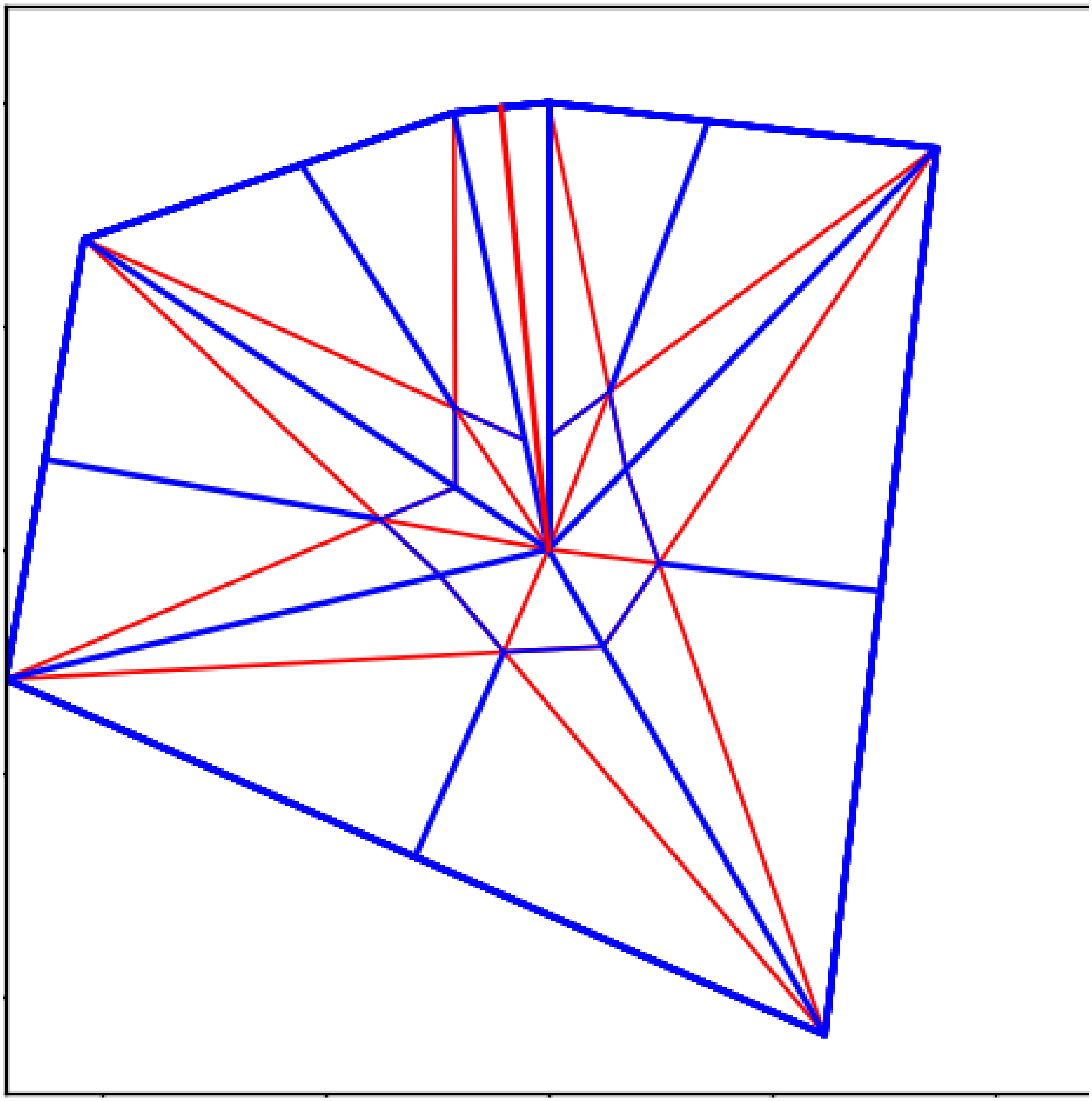
Total number of flaps: 5

The total lengths of flaps: [4.0, 5.0, 5.0, 5.0, 5.0]

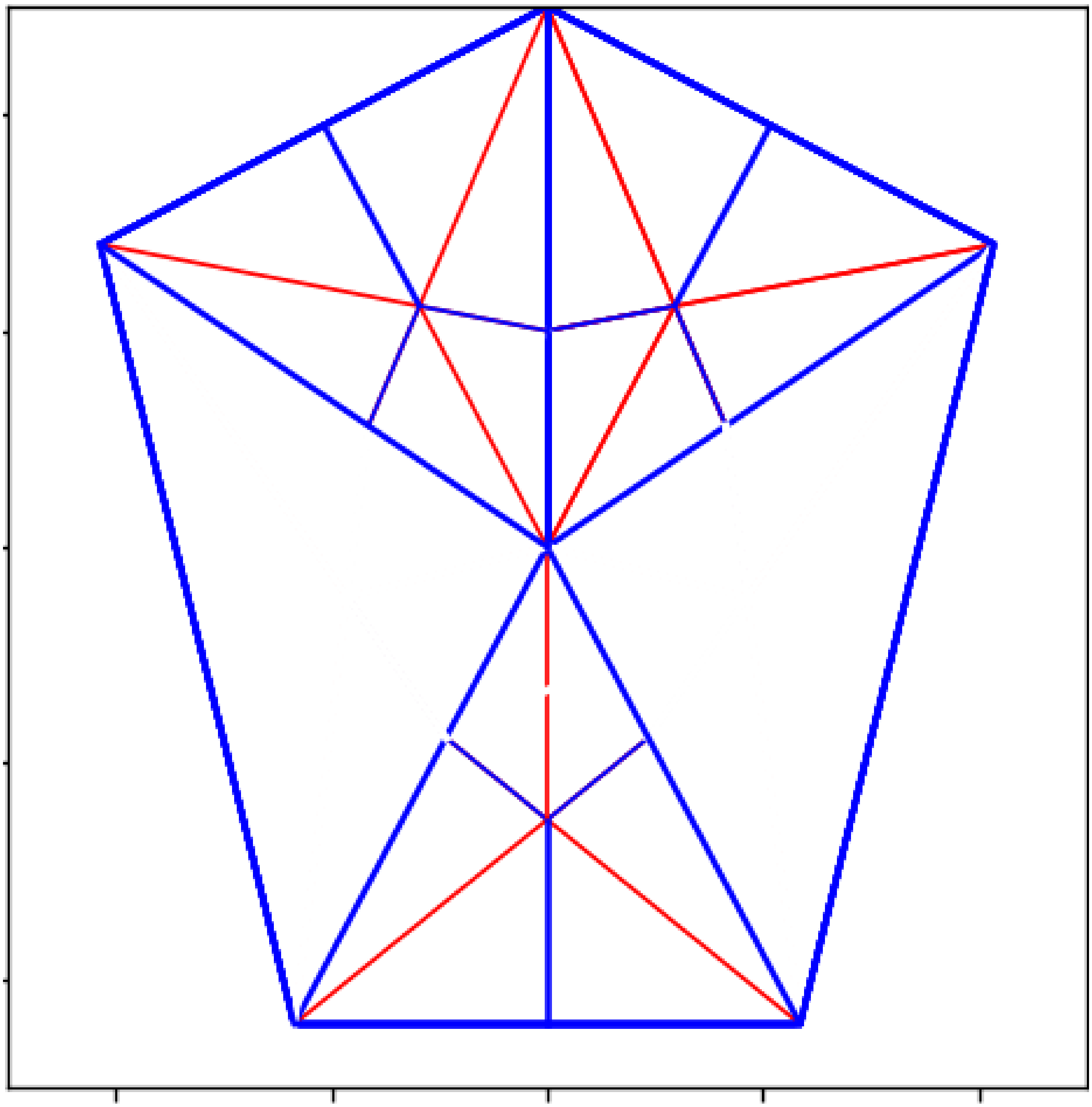
The length of open-ended flaps: [3.0, 4.0, 4.0, 4.0, 4.0]

Original connectivity between flaps: [2.0, 1.0, 1.0, 2.0, 2.0]

Final connectivity value: [3.765741215338876, 0.9538351956674005, 0.9538351956674005, 3.748211468822247, 3.765741215338876]



Same as one earlier, but not optimized



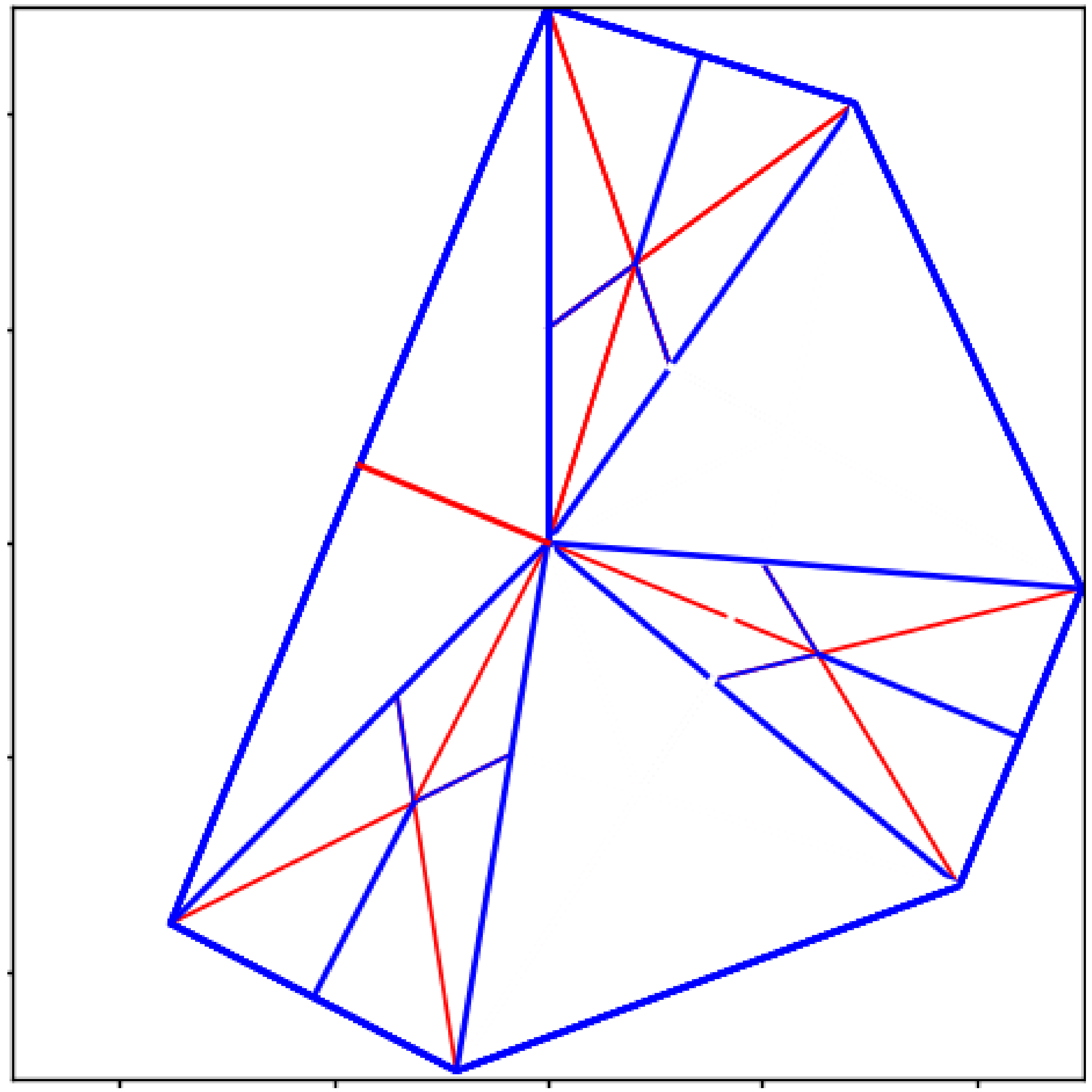
Total number of flaps: 5

The total lengths of flaps: [5.0, 5.0, 5.0, 5.0, 5.0]

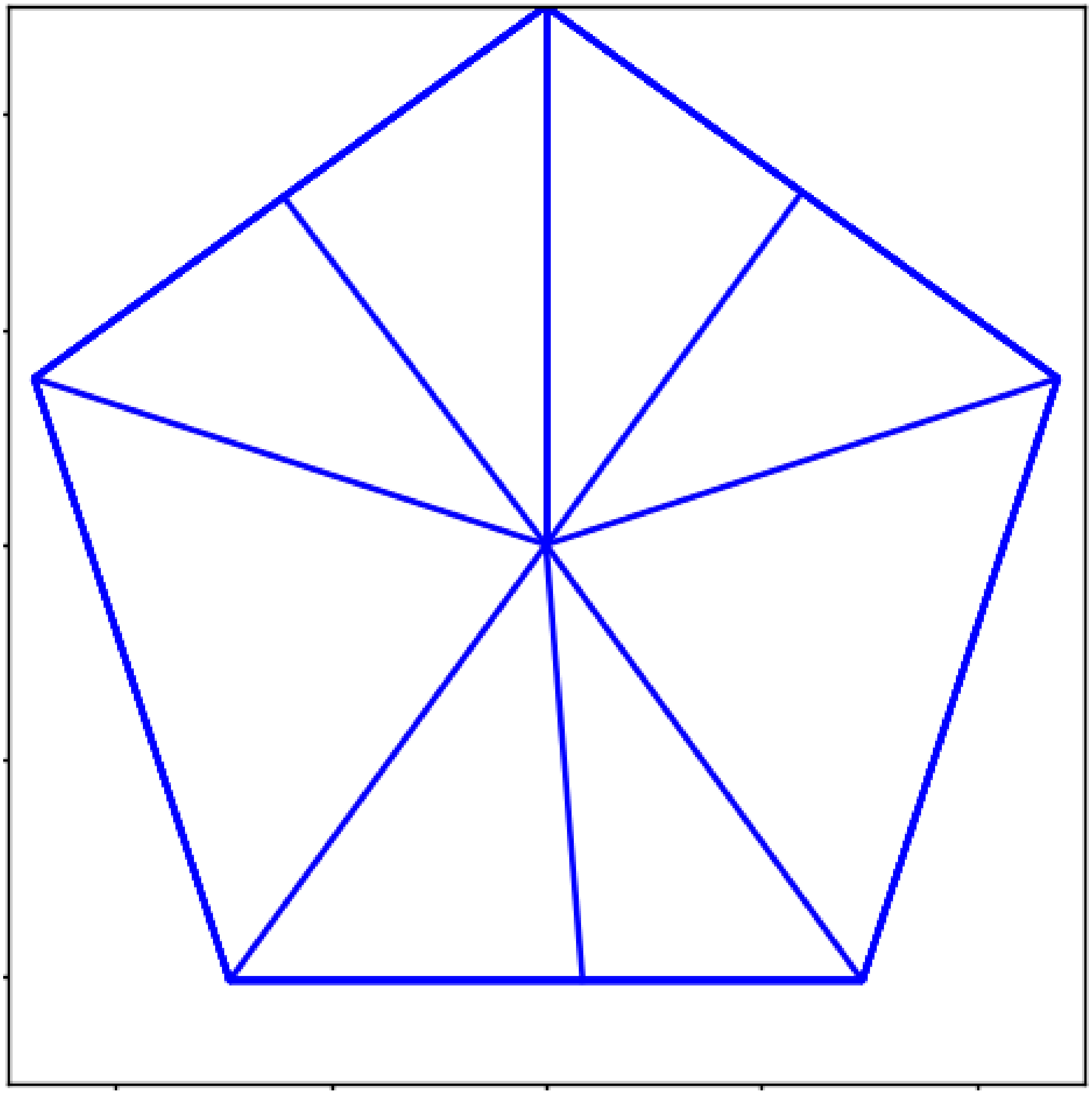
The length of open-ended flaps: [3.0, 3.0, 3.0, 3.0, 3.0]

Original connectivity between flaps: [2.0, 2.0, 2.0, 2.0, 2.0]

Final connectivity value: [1.6372589605444714, 0.6507507847943713, 1.6372589605444714, 0.6507507847943713, 1.6372589605444714]



Same as earlier one, just not optimized



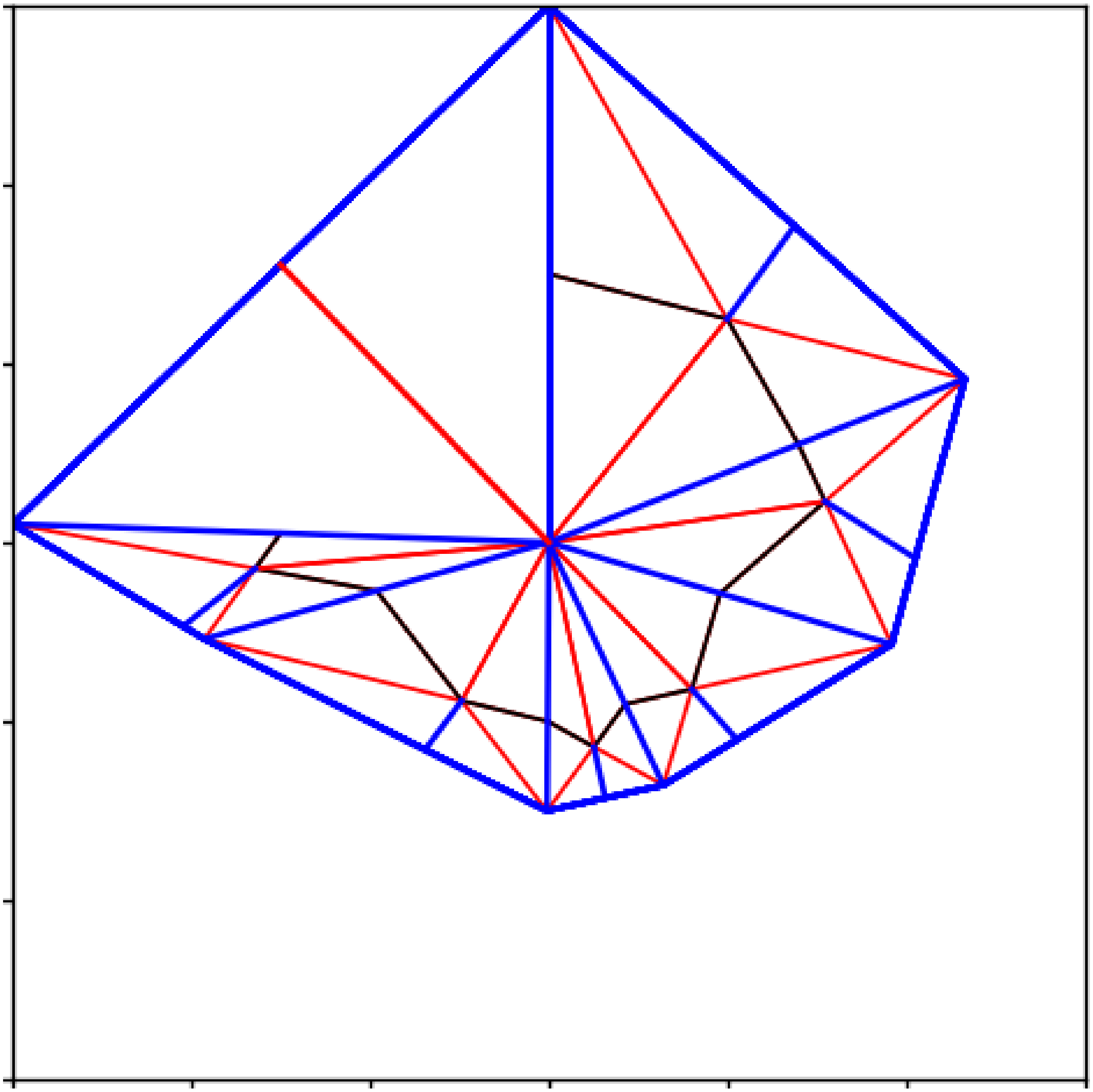
Total number of flaps: 5

The total lengths of flaps: [5.0, 5.0, 5.0, 5.0, 5.0]

The length of open-ended flaps: [4.999999999999999, 4.999999999999999, 4.999999999999999, 4.999999999999999, 4 .999999999999999]

Original connectivity between flaps: [2.0, 2.0, 2.0, 2.0, 2.0]

Final connectivity value: [2.8944271909999144, 2.8944271909999144, 2.8944271909999144, 2.8944271909999144, 2.8944271909999144]

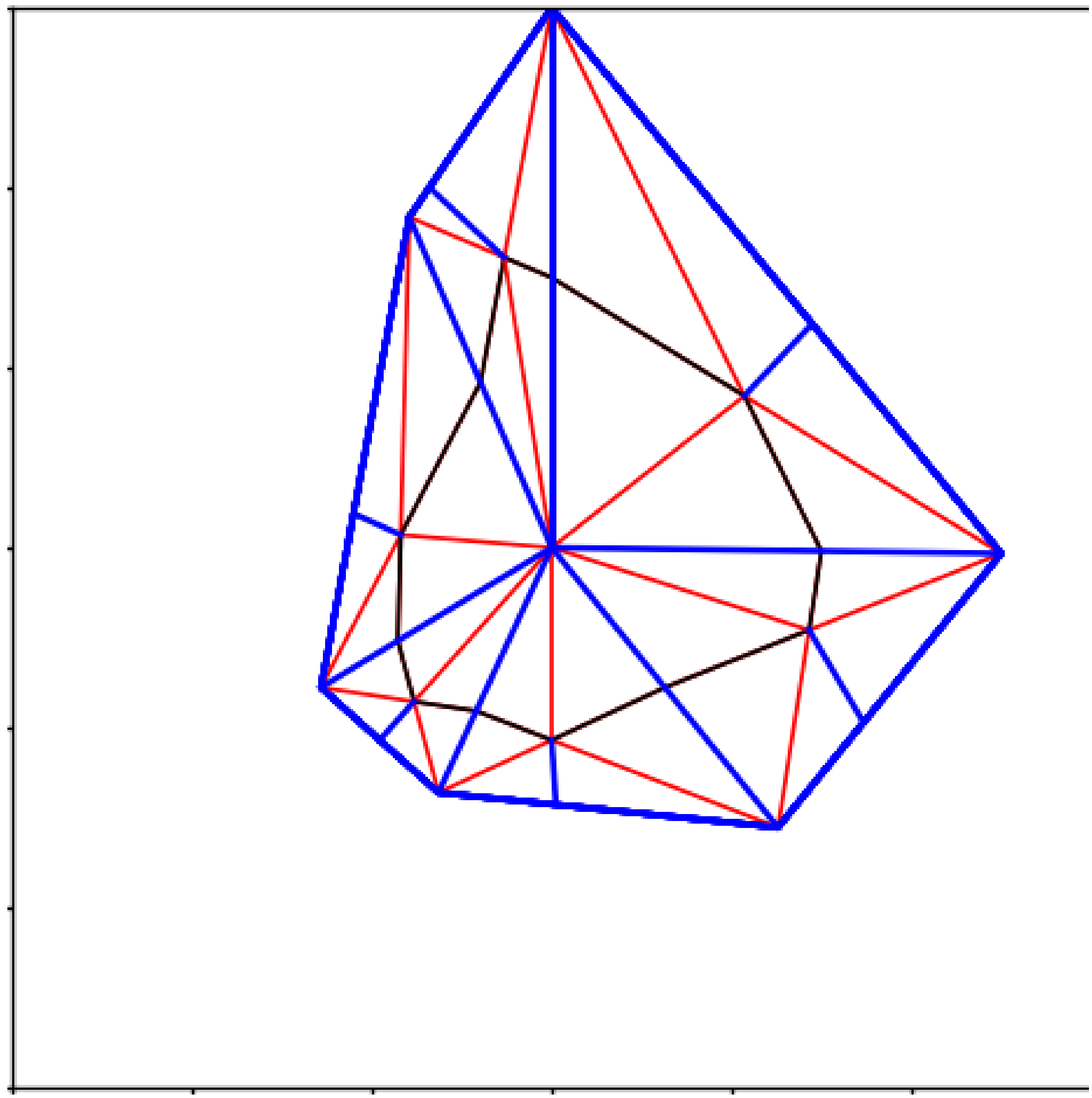


Total number of flaps: 6

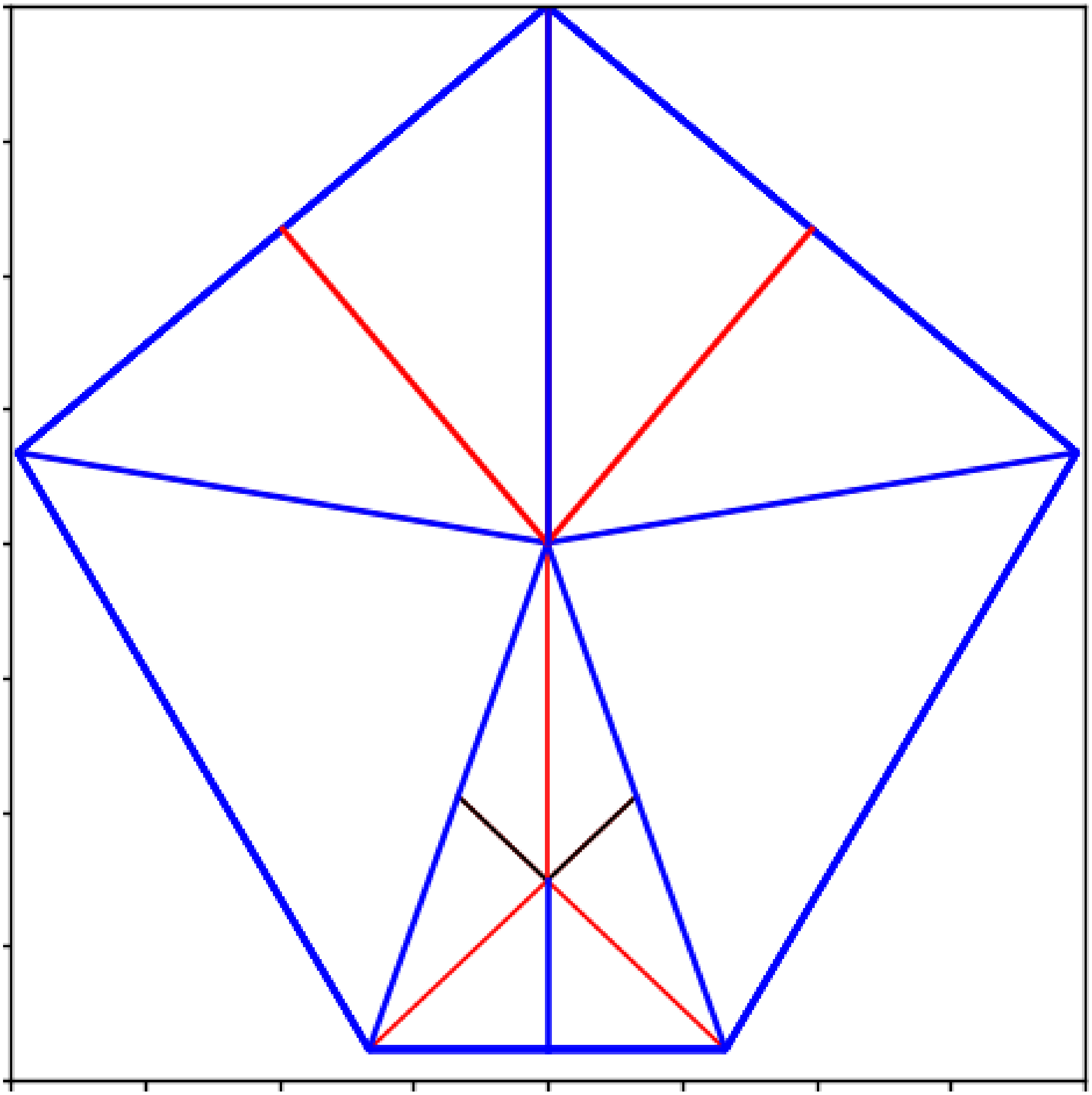
The total lengths of flaps: [6.0, 5.0, 4.0, 3.0, 3.0, 4.0]

The length of open-ended flaps: [3.0, 2.0, 2.0, 1.0, 1.0, 2.0]

Original connectivity between flaps: [0.8, 1.3, 1.0, 1.5, 0.7, 2.0]



Same as earlier, just optimized



Total number of flaps: 5

The total lengths of flaps: [1.0, 1.0, 1.0, 1.0, 1.0]

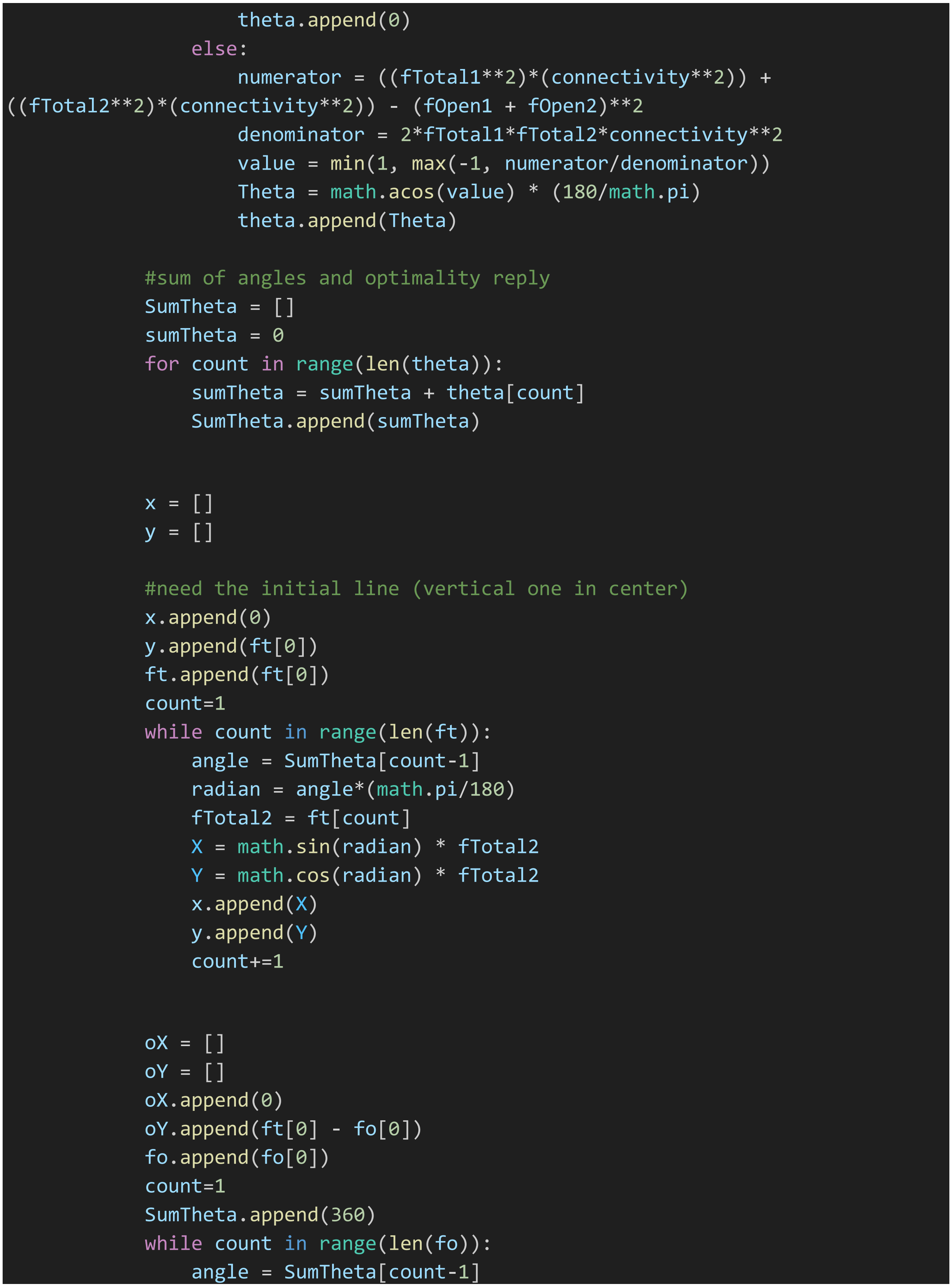
The length of open-ended flaps: [0.99999999, 0.99999999, 0.5, 0.5, 0.99999999]

Original connectivity between flaps: [2.0, 2.0, 2.0, 2.0, 2.0]

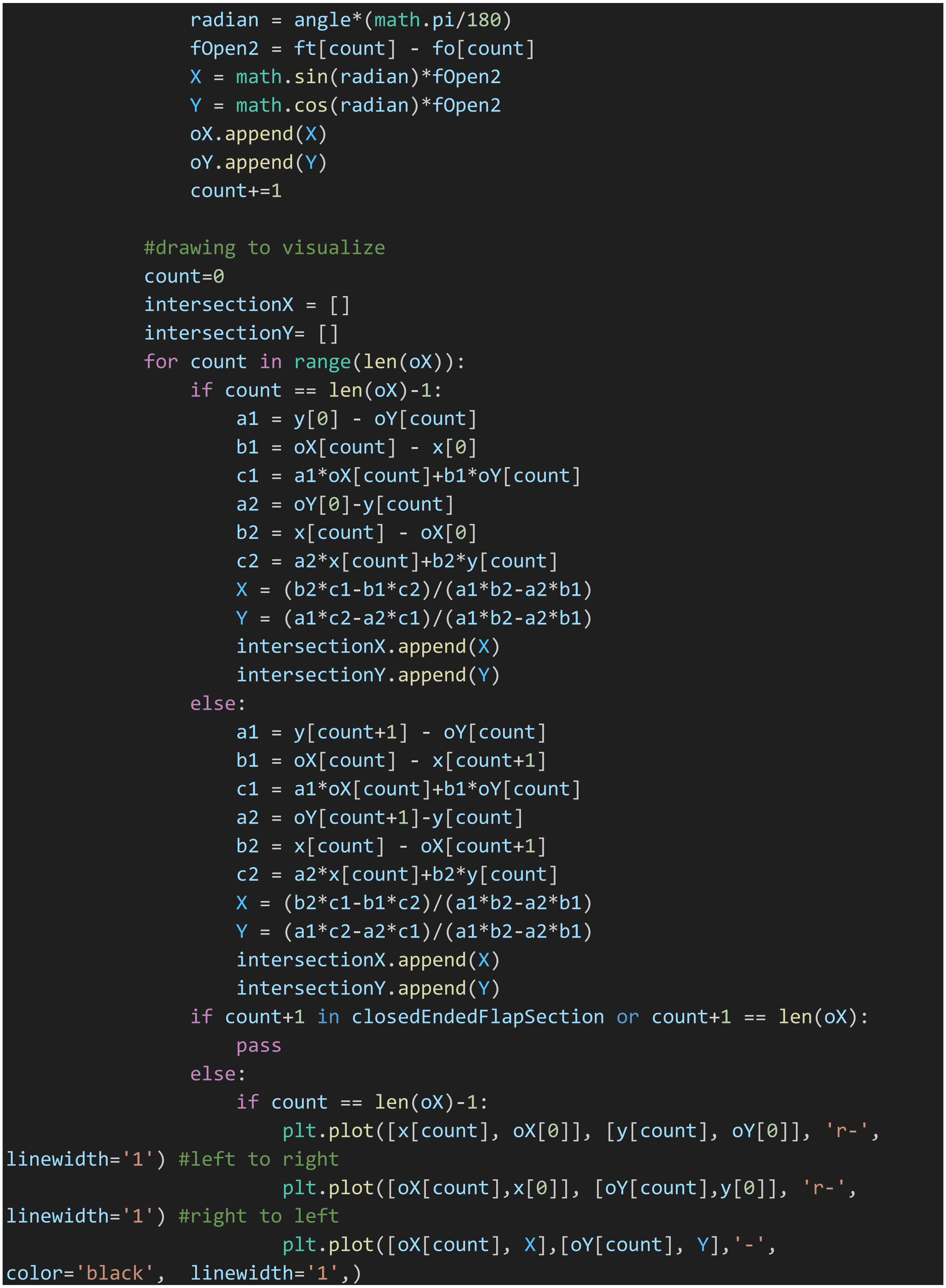
Final connectivity value: [2.4047222771414805, 1.3526562816502945, 2.270779539114497, 1.3526562816502945, 2.4047222771414805]

Appendix 15. The computer program used in the Second and Third test

```
import math
import numpy as np
from matplotlib import pyplot as plt
from decimal import Decimal
import random
import openpyxl
import random
import sys, os
import traceback

# Create a workbook and select the active worksheet
wb = openpyxl.Workbook()
worksheet = wb.active

#headers to the worksheet
worksheet.append(["Flap number", "FT length", "FO length", "c-values", "Error"])

#Generate the variables

repetitions = int(input())
def generateRandomValues(repetitions):
    for i in range(repetitions):
        foTotal = random.randrange(3,10)
        ft = []
        fo = []
        c = []
        oft = []
        ofo = []
        oc = []
        closedEndedFlapSection = [0]
        for j in range(foTotal):
            FT = round(random.uniform(1,10), 2)
            FO = round(random.uniform(1,FT), 2)
            C = round(random.uniform(0.01,1), 2)
            ft.append(FT)
            fo.append(FO)
            c.append(C)
            oft.append(FT)
            ofo.append(FO)
            oc.append(C)
        print(foTotal, len(ft), len(fo), len(c))
        #NClosedEndedFlapSection = random.randrange(0, foTotal)
        #if NClosedEndedFlapSection == 0:
        #    closedEndedFlapSection.append(0)
        #    pass
        #else:
        #    for i in range(NClosedEndedFlapSection):
        #        ClosedEndedFlapSection = random.randrange(1, foTotal)
        #        closedEndedFlapSection.append(ClosedEndedFlapSection)

        try:
            #recalibration of c value for accurate connectivity output when
folded
            if closedEndedFlapSection == [0]:
                pass
            else:
                for count in range(len(closedEndedFlapSection)):
                    ClosedEndedFlapSection = closedEndedFlapSection[count]
                    length = (ft[ClosedEndedFlapSection-1] +
ft[ClosedEndedFlapSection]) / c[ClosedEndedFlapSection-1]
                    C = (fo[ClosedEndedFlapSection-1] +
fo[ClosedEndedFlapSection]) / length
                    c[ClosedEndedFlapSection-1] = C

            #fo - 0.00000001 when equal to ft
            changedfo = []
            for count in range(len(ft)):
                if ft[count] == fo[count]:
                    fo[count] = fo[count]-10**-8
                    changedfo.append(bool(True))
                else:
                    changedfo.append(bool(False))


            #calculation
            theta = []
            for count in range(len(ft)):
                fTotal1 = ft[count]
                fOpen1 = fo[count]
                if count+1 == len(ft):
                    fTotal2 = ft[0]
                else:
                    fTotal2 = ft[count+1]
                if count+1 == len(fo):
                    fOpen2 = fo[0]
                else:
                    fOpen2 = fo[count+1]
                connectivity = c[count]
                if connectivity == 0:
```

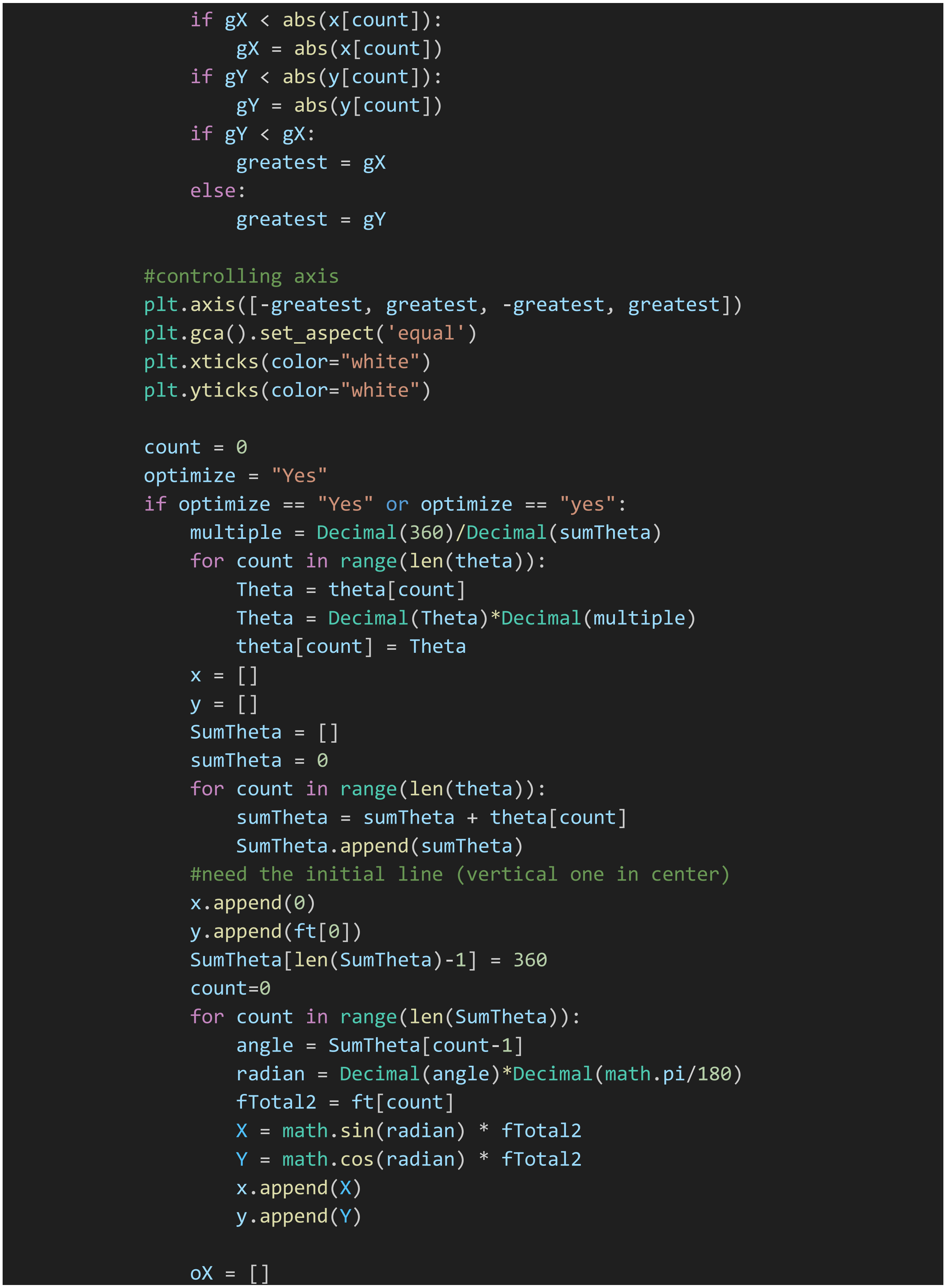


```
                    theta.append(0)
                else:
                    numerator = ((fTotal1**2)*(connectivity**2)) +
((fTotal2**2)*(connectivity**2)) - (fOpen1 + fOpen2)**2
                    denominator = 2*fTotal1*fTotal2*connectivity**2
                    value = min(1, max(-1, numerator/denominator))
                    Theta = math.acos(value) * (180/math.pi)
                    theta.append(Theta)

            #sum of angles and optimality reply
            SumTheta = []
            sumTheta = 0
            for count in range(len(theta)):
                sumTheta = sumTheta + theta[count]
                SumTheta.append(sumTheta)


            x = []
            y = []

            #need the initial line (vertical one in center)
            x.append(0)
            y.append(ft[0])
            ft.append(ft[0])
            count=1
            while count in range(len(ft)):
                angle = SumTheta[count-1]
                radian = angle*(math.pi/180)
                fTotal2 = ft[count]
                X = math.sin(radian) * fTotal2
                Y = math.cos(radian) * fTotal2
                x.append(X)
                y.append(Y)
                count+=1


            oX = []
            oY = []
            oX.append(0)
            oY.append(ft[0] - fo[0])
            fo.append(fo[0])
            count=1
            SumTheta.append(360)
            while count in range(len(fo)):
                angle = SumTheta[count-1]
```

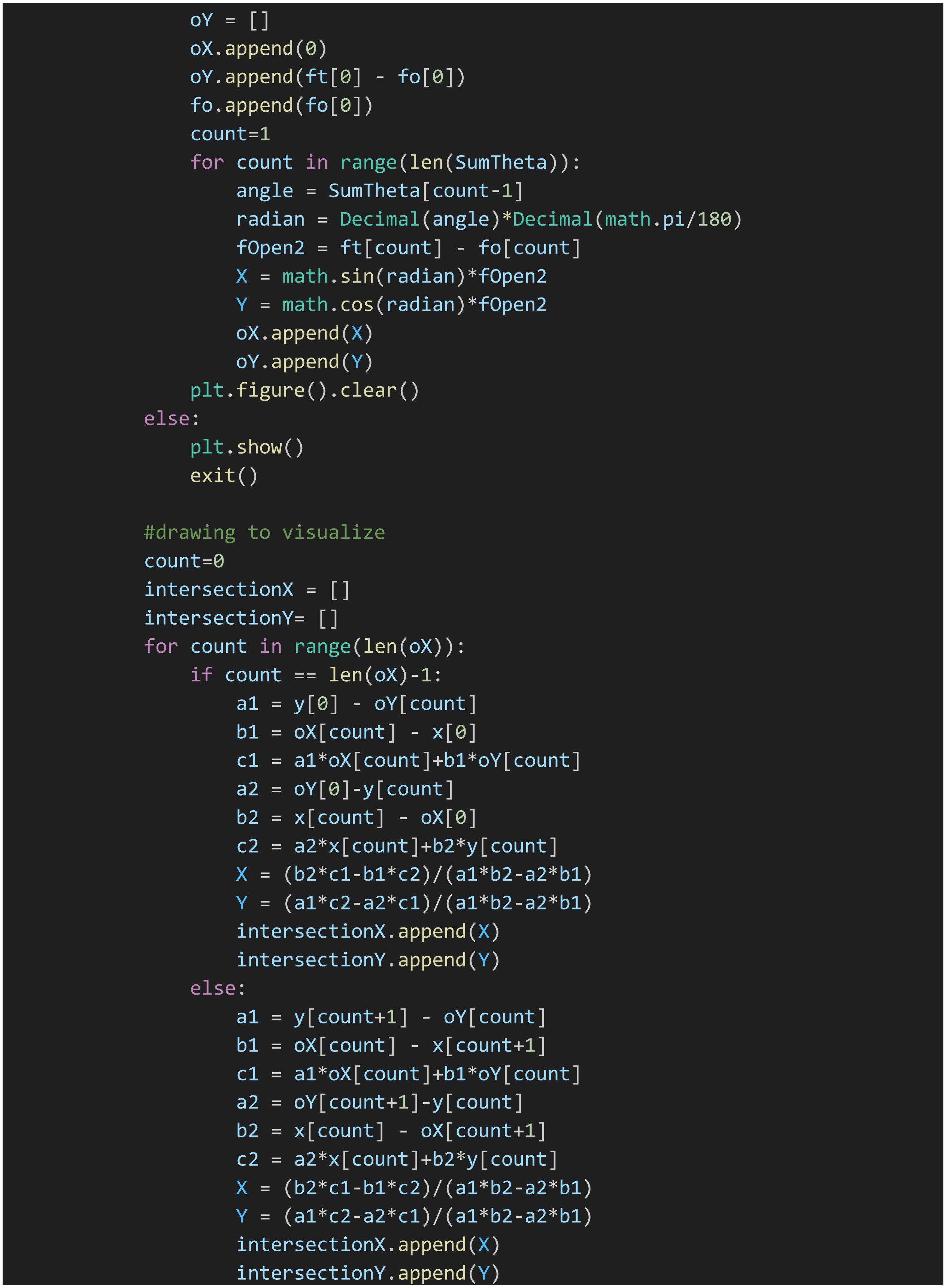


```
                radian = angle*(math.pi/180)
                fOpen2 = ft[count] - fo[count]
                X = math.sin(radian)*fOpen2
                Y = math.cos(radian)*fOpen2
                oX.append(X)
                oY.append(Y)
                count+=1

            #drawing to visualize
            count=0
            intersectionX = []
            intersectionY= []
            for count in range(len(oX)):
                if count == len(oX)-1:
                    a1 = y[0] - oY[count]
                    b1 = oX[count] - x[0]
                    c1 = a1*oX[count]+b1*oY[count]
                    a2 = oY[0]-y[count]
                    b2 = x[count] - oX[0]
                    c2 = a2*x[count]+b2*y[count]
                    X = (b2*c1-b1*c2)/(a1*b2-a2*b1)
                    Y = (a1*c2-a2*c1)/(a1*b2-a2*b1)
                    intersectionX.append(X)
                    intersectionY.append(Y)
                else:
                    a1 = y[count+1] - oY[count]
                    b1 = oX[count] - x[count+1]
                    c1 = a1*oX[count]+b1*oY[count]
                    a2 = oY[count+1]-y[count]
                    b2 = x[count] - oX[count+1]
                    c2 = a2*x[count]+b2*y[count]
                    X = (b2*c1-b1*c2)/(a1*b2-a2*b1)
                    Y = (a1*c2-a2*c1)/(a1*b2-a2*b1)
                    intersectionX.append(X)
                    intersectionY.append(Y)
                if count+1 in closedEndedFlapSection or count+1 == len(oX):
                    pass
                else:
                    if count == len(oX)-1:
                        plt.plot([x[count], oX[0]], [y[count], oY[0]], 'r-',
linewidth='1') #left to right
                        plt.plot([oX[count],x[0]], [oY[count],y[0]], 'r-',
linewidth='1') #right to left
                        plt.plot([oX[count], X],[oY[count], Y],'-',
color='black',  linewidth='1',)
```

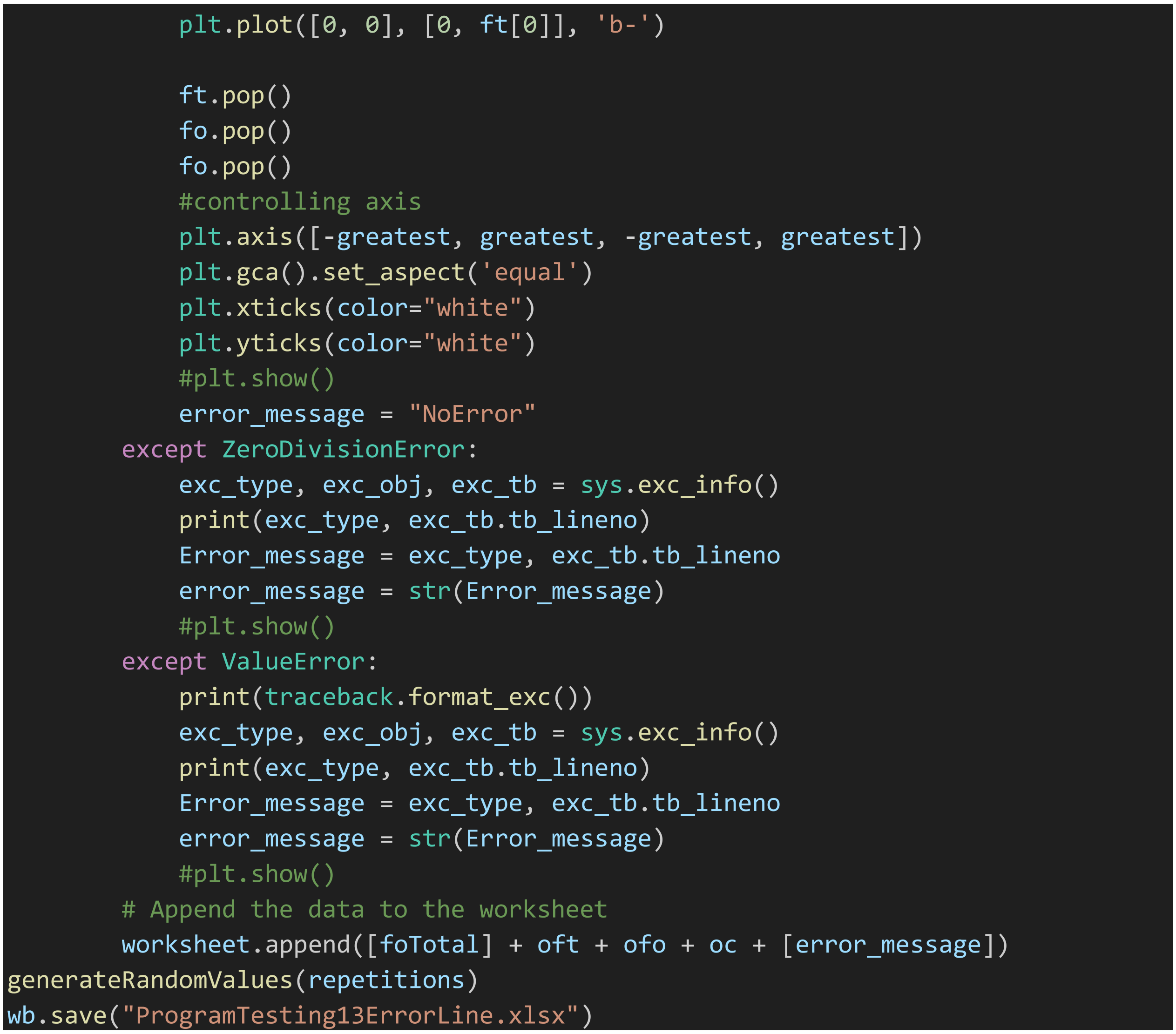
```
                        plt.plot([X, oX[0]],[Y, oY[0]], '-', color='black', linewidth='1')
                        plt.plot([0, X], [0, Y], 'r-', linewidth='1')
                    else:
                        plt.plot([x[count], oX[count+1]], [y[count], oY[count+1]], 'r-', linewidth='1') #left to right
                        plt.plot([oX[count],x[count+1]], [oY[count],y[count+1]], 'r-', linewidth='1') #right to left
                        plt.plot([oX[count], X],[oY[count], Y],'-', color='black',  linewidth='1',)
                        plt.plot([X, oX[count+1]],[Y, oY[count+1]], '-', color='black', linewidth='1')
                        plt.plot([0, X], [0, Y], 'r-', linewidth='1')
                        plt.plot([0, X], [0, Y], 'r-', linewidth='1')
                x.append(0)
                y.append(ft[0])
                plt.plot([x[count],0], [y[count],0],'b-')
                plt.plot(x, y, 'b-')
                plt.plot([0,0],[0,ft[0]], 'b-')

            #final line calculations
            count = 0
            Alpha = []
            Beta = []
            finalLineLength = []
            for count in range(len(intersectionX)):
                x0 = intersectionX[count]
                y0 = intersectionY[count]
                x1 = x[count]
                y1 = y[count]
                ax = oX[count] - intersectionX[count]
                ay = oY[count] - intersectionY[count]
                bx = 0 - intersectionX[count]
                by = 0 - intersectionY[count]
                value = Decimal((x1-x0)/(y1-y0))
                if (y1-y0) > 0:
                    alpha = math.atan(value)
                    alpha = alpha * 180/math.pi
                    Alpha.append(alpha)
                elif (y1-y0) < 0:
                    alpha = math.atan(value)
                    alpha = alpha * 180/math.pi
                    alpha = 180 + alpha
                    Alpha.append(alpha)
                #angle between vectors
                ab = ax*bx + ay*by
                absA = math.sqrt(ax**2 + ay**2)
                absB = math.sqrt(bx**2 + by**2)
                beta = math.acos(ab/(absA*absB))
                beta = beta * 180/math.pi
                Beta.append(beta)
                #finding length of the final line
                m1 = (y[count+1]-y[count])/(x[count+1]-x[count])
                b1 = y[count+1]-(m1*x[count+1])
                m2 = 1/math.tan((alpha+beta)*math.pi/180)
                b2 = intersectionY[count]-(m2*intersectionX[count])
                fLIX = (b2-b1)/(m1-m2)
                fLIY = m2*fLIX+b2
                fLL = math.sqrt((fLIX-intersectionX[count])**2 + (fLIY-
intersectionY[count])**2)
                finalLineLength.append(fLL)
                if count+1 in closedEndedFlapSection or count+1 ==
len(intersectionX):
                    pass
                else:
                    #plotting final line
                    X = intersectionX[count] +
fLL*math.sin((alpha+beta)*math.pi/180)
                    Y = intersectionY[count] +
fLL*math.cos((alpha+beta)*math.pi/180)
                    if changedfo[count]:
                        plt.plot([intersectionX[count],X],[intersectionY[count],Y
],'-', color='red')
                    else:
                        plt.plot([intersectionX[count],X],[intersectionY[count],Y
],'-', color='blue')
            #drawing extra valley fold
            Theta = (360 - Decimal(sumTheta))/2
            angle = Decimal(SumTheta[len(SumTheta)-1]) + Decimal(Theta)
            radianTheta = (Decimal(Theta)+Decimal(sumTheta))*Decimal(math.pi/180)
            length = ft[0] * math.cos(radianTheta)
            radianAngle = Decimal(angle)*Decimal(math.pi/180)
            X = math.sin(radianAngle)*length
            Y = math.cos(radianAngle)*length
            plt.plot([0, -X], [0, Y], 'r-')

            #Finding length of axis
            gX = 0
            gY = 0
            for count in range(len(x)):
```



```
            if gX < abs(x[count]):
                gX = abs(x[count])
            if gY < abs(y[count]):
                gY = abs(y[count])
            if gY < gX:
                greatest = gX
            else:
                greatest = gY

        #controlling axis
        plt.axis([-greatest, greatest, -greatest, greatest])
        plt.gca().set_aspect('equal')
        plt.xticks(color="white")
        plt.yticks(color="white")

        count = 0
        optimize = "Yes"
        if optimize == "Yes" or optimize == "yes":
            multiple = Decimal(360)/Decimal(sumTheta)
            for count in range(len(theta)):
                Theta = theta[count]
                Theta = Decimal(Theta)*Decimal(multiple)
                theta[count] = Theta
            x = []
            y = []
            SumTheta = []
            sumTheta = 0
            for count in range(len(theta)):
                sumTheta = sumTheta + theta[count]
                SumTheta.append(sumTheta)
            #need the initial line (vertical one in center)
            x.append(0)
            y.append(ft[0])
            SumTheta[len(SumTheta)-1] = 360
            count=0
            for count in range(len(SumTheta)):
                angle = SumTheta[count-1]
                radian = Decimal(angle)*Decimal(math.pi/180)
                fTotal2 = ft[count]
                X = math.sin(radian) * fTotal2
                Y = math.cos(radian) * fTotal2
                x.append(X)
                y.append(Y)

            oX = []
```



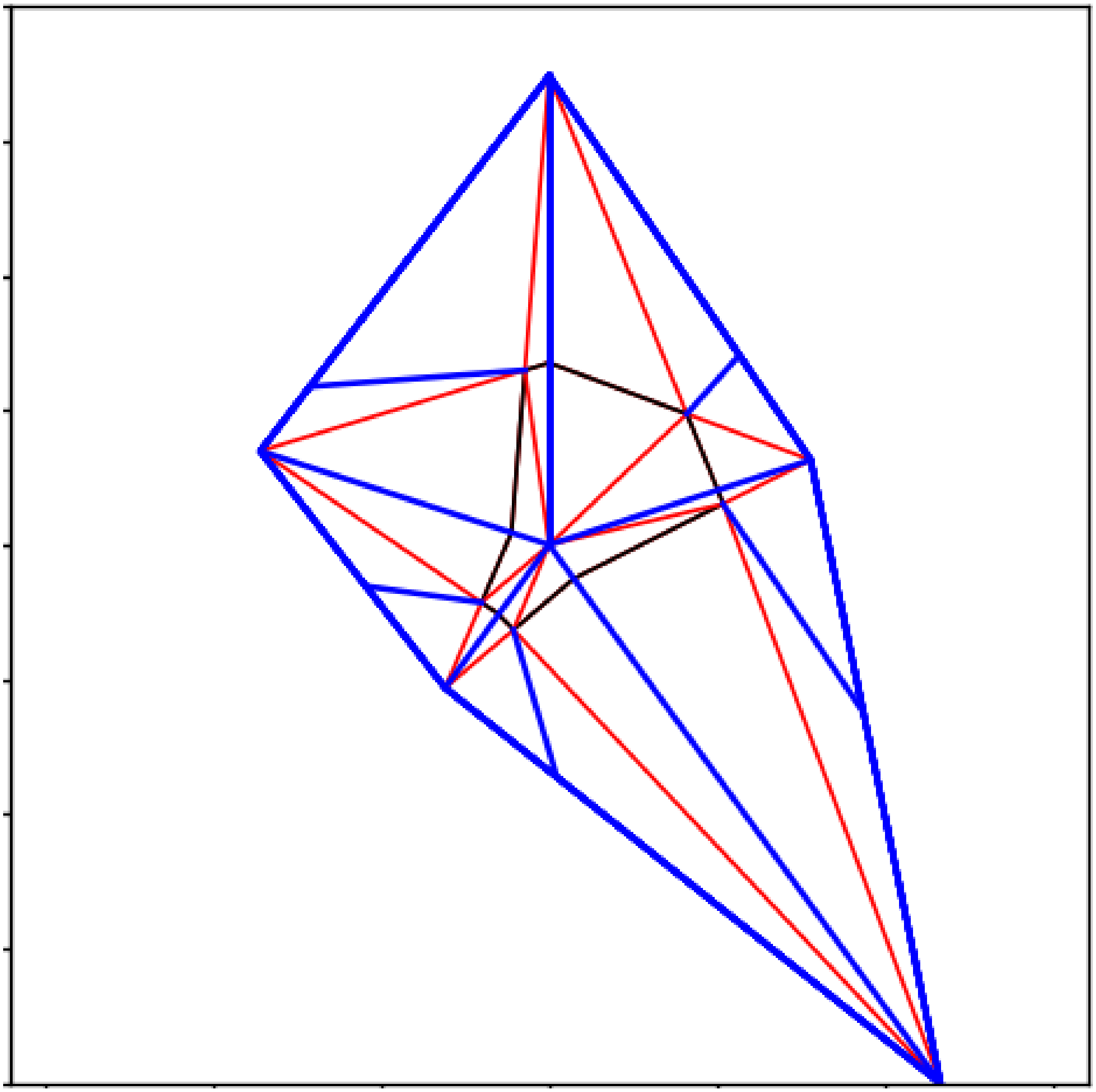
```
            oY = []
            oX.append(0)
            oY.append(ft[0] - fo[0])
            fo.append(fo[0])
            count=1
            for count in range(len(SumTheta)):
                angle = SumTheta[count-1]
                radian = Decimal(angle)*Decimal(math.pi/180)
                fOpen2 = ft[count] - fo[count]
                X = math.sin(radian)*fOpen2
                Y = math.cos(radian)*fOpen2
                oX.append(X)
                oY.append(Y)
            plt.figure().clear()
        else:
            plt.show()
            exit()

        #drawing to visualize
        count=0
        intersectionX = []
        intersectionY= []
        for count in range(len(oX)):
            if count == len(oX)-1:
                a1 = y[0] - oY[count]
                b1 = oX[count] - x[0]
                c1 = a1*oX[count]+b1*oY[count]
                a2 = oY[0]-y[count]
                b2 = x[count] - oX[0]
                c2 = a2*x[count]+b2*y[count]
                X = (b2*c1-b1*c2)/(a1*b2-a2*b1)
                Y = (a1*c2-a2*c1)/(a1*b2-a2*b1)
                intersectionX.append(X)
                intersectionY.append(Y)
            else:
                a1 = y[count+1] - oY[count]
                b1 = oX[count] - x[count+1]
                c1 = a1*oX[count]+b1*oY[count]
                a2 = oY[count+1]-y[count]
                b2 = x[count] - oX[count+1]
                c2 = a2*x[count]+b2*y[count]
                X = (b2*c1-b1*c2)/(a1*b2-a2*b1)
                Y = (a1*c2-a2*c1)/(a1*b2-a2*b1)
                intersectionX.append(X)
                intersectionY.append(Y)
                if count in closedEndedFlapSection:
                    pass
                else:
                    if count == len(oX)-1:
                        plt.plot([x[count], oX[0]], [y[count], oY[0]], 'r-',
linewidth='1') #left to right
                        plt.plot([oX[count],x[0]], [oY[count],y[0]], 'r-',
linewidth='1') #right to left
                        plt.plot([oX[count], X],[oY[count], Y],'-',
color='black',  linewidth='1',)
                        plt.plot([X, oX[0]],[Y, oY[0]], '-', color='black',
linewidth='1')
                        plt.plot([0, X], [0, Y], 'r-', linewidth='1')
                    else:
                        plt.plot([x[count], oX[count+1]], [y[count],
oY[count+1]], 'r-', linewidth='1') #left to right
                        plt.plot([oX[count],x[count+1]], [oY[count],y[count+1]],
'r-', linewidth='1') #right to left
                        plt.plot([oX[count], X],[oY[count], Y],'-',
color='black',  linewidth='1',)
                        plt.plot([X, oX[count+1]],[Y, oY[count+1]], '-',
color='black', linewidth='1')
                        plt.plot([0, X], [0, Y], 'r-', linewidth='1')
                x.append(0)
                y.append(ft[0])
                plt.plot([x[count],0], [y[count],0],'b-')
                plt.plot(x, y, 'b-')
                plt.plot([0,0],[0,ft[0]], 'b-')

            #final line calculations
            count = 0
            Alpha = []
            Beta = []
            finalLineLength = []
            for count in range(len(intersectionY)):
                x0 = intersectionX[count]
                y0 = intersectionY[count]
                x1 = x[count]
                y1 = y[count]
                ax = oX[count] - intersectionX[count]
                ay = oY[count] - intersectionY[count]
                bx = 0 - intersectionX[count]
                by = 0 - intersectionY[count]
                value = Decimal((x1-x0)/(y1-y0))
                if (y1-y0) > 0:
                    alpha = math.atan(value)
                    alpha = alpha * 180/math.pi
                    Alpha.append(alpha)
                elif (y1-y0) < 0:
                    alpha = math.atan(value)
                    alpha = alpha * 180/math.pi
                    alpha = 180 + alpha
                    Alpha.append(alpha)
                #angle between vectors
                ab = ax*bx + ay*by
                absA = math.sqrt(ax**2 + ay**2)
                absB = math.sqrt(bx**2 + by**2)
                beta = math.acos(ab/(absA*absB))
                beta = beta * 180/math.pi
                Beta.append(beta)
                if count == len(x)-1:
                    #finding length of the final line
                    m1 = (y[0]-y[count])/(x[0]-x[count])
                    b1 = y[0]-(m1*x[0])
                    m2 = 1/math.tan((alpha+beta)*math.pi/180)
                    b2 = intersectionY[count]-(m2*intersectionX[count])
                else:
                    #finding length of the final line
                    m1 = (y[count+1]-y[count])/(x[count+1]-x[count])
                    b1 = y[count+1]-(m1*x[count+1])
                    m2 = 1/math.tan((alpha+beta)*math.pi/180)
                    b2 = intersectionY[count]-(m2*intersectionX[count])
                fLIX = (b2-b1)/(m1-m2)
                fLIY = m2*fLIX+b2
                fLL = math.sqrt((fLIX-intersectionX[count])**2 + (fLIY-
intersectionY[count])**2)
                if count in closedEndedFlapSection:
                    pass
                else:
                    #plotting final line
                    X = intersectionX[count] +
fLL*math.sin((alpha+beta)*math.pi/180)
                    Y = intersectionY[count] +
fLL*math.cos((alpha+beta)*math.pi/180)
                    if changedfo[count-1]:
                        plt.plot([intersectionX[count],X],[intersectionY[count],Y
],'-', color='red')
                    else:
                        plt.plot([intersectionX[count],X],[intersectionY[count],Y
],'-', color='blue')
```



```
            plt.plot([0, 0], [0, ft[0]], 'b-')

            ft.pop()
            fo.pop()
            fo.pop()
            #controlling axis
            plt.axis([-greatest, greatest, -greatest, greatest])
            plt.gca().set_aspect('equal')
            plt.xticks(color="white")
            plt.yticks(color="white")
            #plt.show()
            error_message = "NoError"
        except ZeroDivisionError:
            exc_type, exc_obj, exc_tb = sys.exc_info()
            print(exc_type, exc_tb.tb_lineno)
            Error_message = exc_type, exc_tb.tb_lineno
            error_message = str(Error_message)
            #plt.show()
        except ValueError:
            print(traceback.format_exc())
            exc_type, exc_obj, exc_tb = sys.exc_info()
            print(exc_type, exc_tb.tb_lineno)
            Error_message = exc_type, exc_tb.tb_lineno
            error_message = str(Error_message)
            #plt.show()
        # Append the data to the worksheet
        worksheet.append([foTotal] + oft + ofo + oc + [error_message])
generateRandomValues(repetitions)
wb.save("ProgramTesting13ErrorLine.xlsx")
```

Appendix 16. Second test on randomly generated crease patterns

Appendix 16.1



Total number of flaps: 5

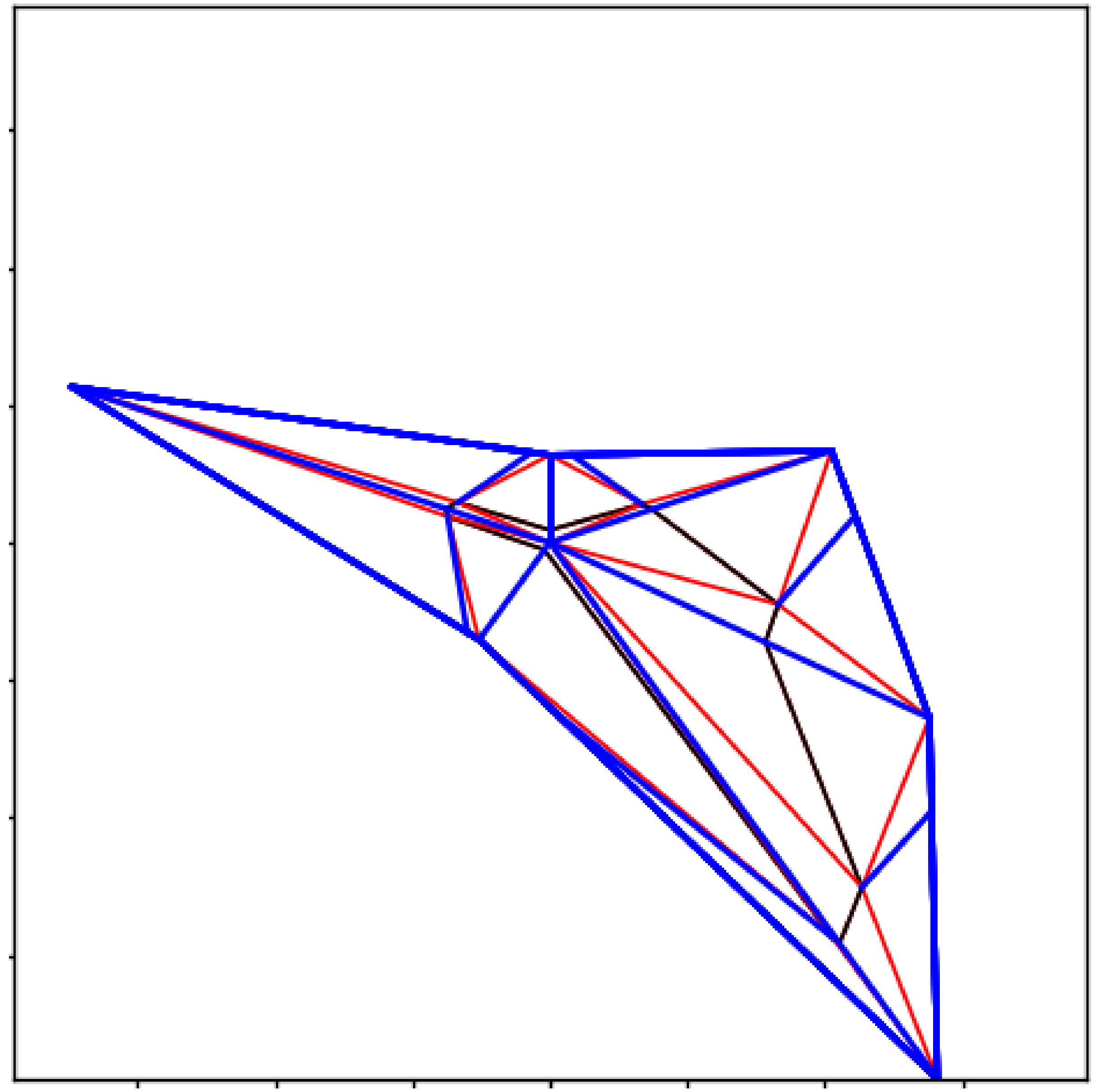
The total lengths of flaps: [6.968573053411594, 4.084843362518074, 9.895733385630354, 2.6271432969399013, 4.51111905492084]

The length of open-ended flaps: [4.26374124280005, 1.450761527049666, 9.274753419747833, 1.3533069581666544, 3.9225258614707204]

Original connectivity between flaps:

Final connectivity value: [0.6852592117223886, 1.2834761091022513, 1.2725954253992882, 1.39678140886816, 1.3543124443666401]

Appendix 16.2



Total number of flaps: 6

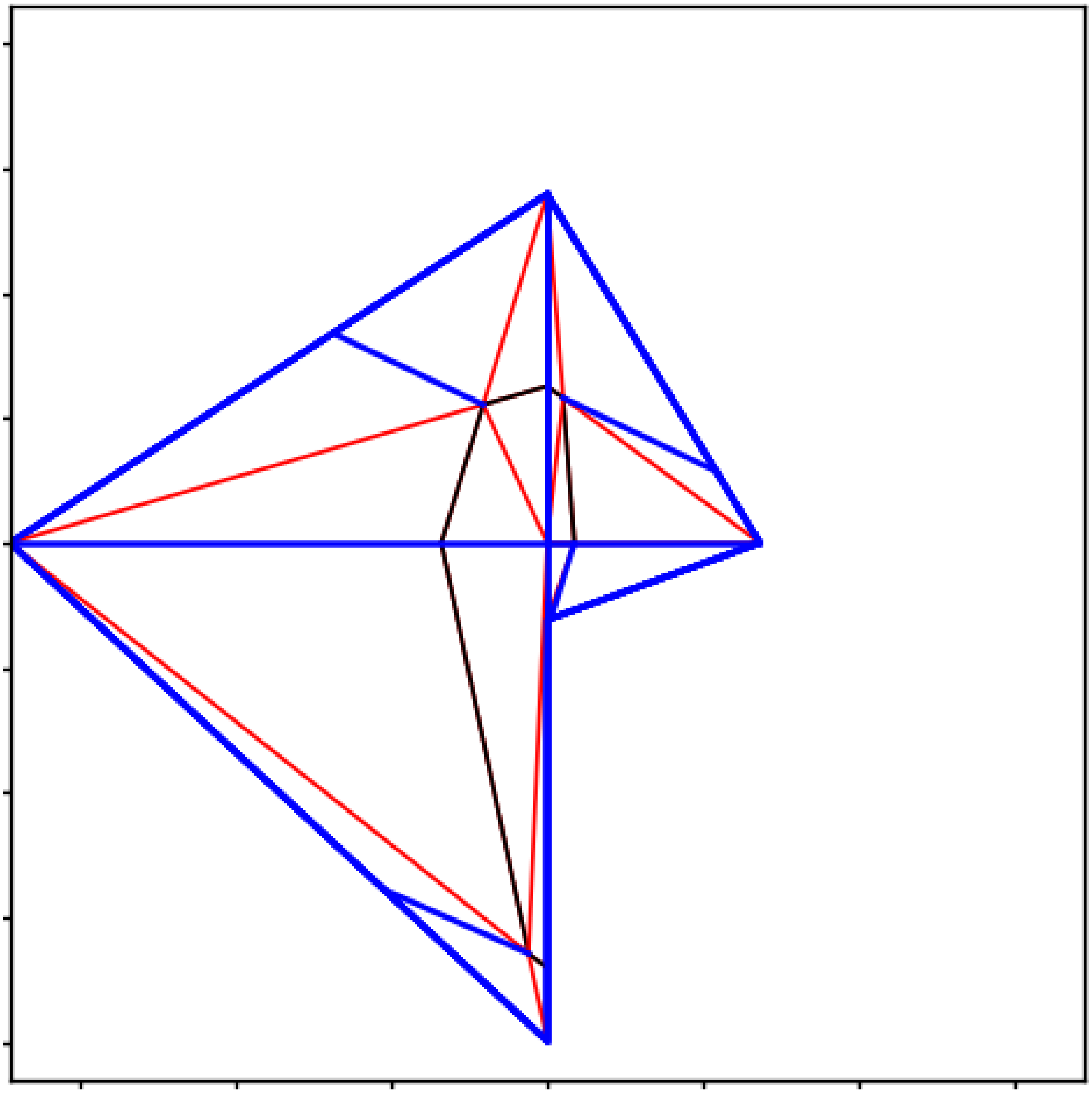
The total lengths of flaps: [1.273516338960313, 4.312490873388468, 6.087277894222673, 9.631853620193693, 1.7480667330405266, 7.324272655279679]

The length of open-ended flaps: [1.0897591610229087, 2.749995613679151, 2.633137310708558, 2.4336728200664197, 1.6160249601075463, 5.72995512329334]

Original connectivity between flaps:

Final connectivity value: [0.8767199514968809, 1.6886903359406715, 0.9302290256055756, 0.19204456525703667, 1.1065204038134053, 0.9397929767056392]

Appendix 16.3



Total number of flaps: 5

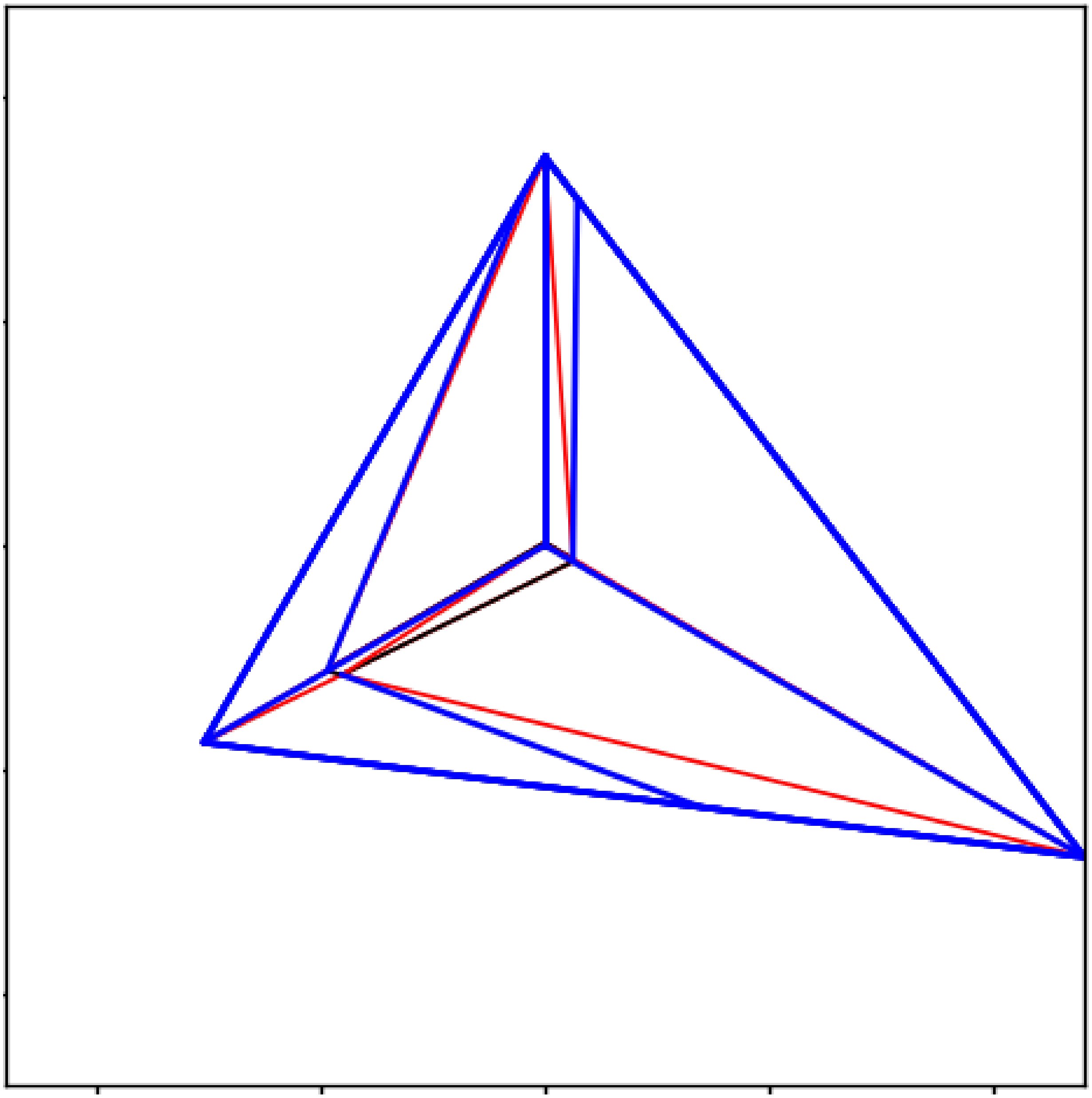
The total lengths of flaps: [5.582971313180651, 3.4064955586271295, 1.2318008582303626, 7.973629276127891, 8.600909184600969]

The length of open-ended flaps: [3.072032640007576, 2.9683713694534095, 1.2141996890702833, 1.1671697386121278, 6.9071451093363105]

Original connectivity between flaps: [0.3997683211083244, 0.06453650878503558, 0.5445513858156292, 0.02602325127049676, 0.8527854205129118]

Final connectivity value: [0.8530104195128021, 1.3332195407828427, 0.12476654729196729, 0.4739547336543806, 0.9471091544204192]

Appendix 16.4



Total number of flaps: 3

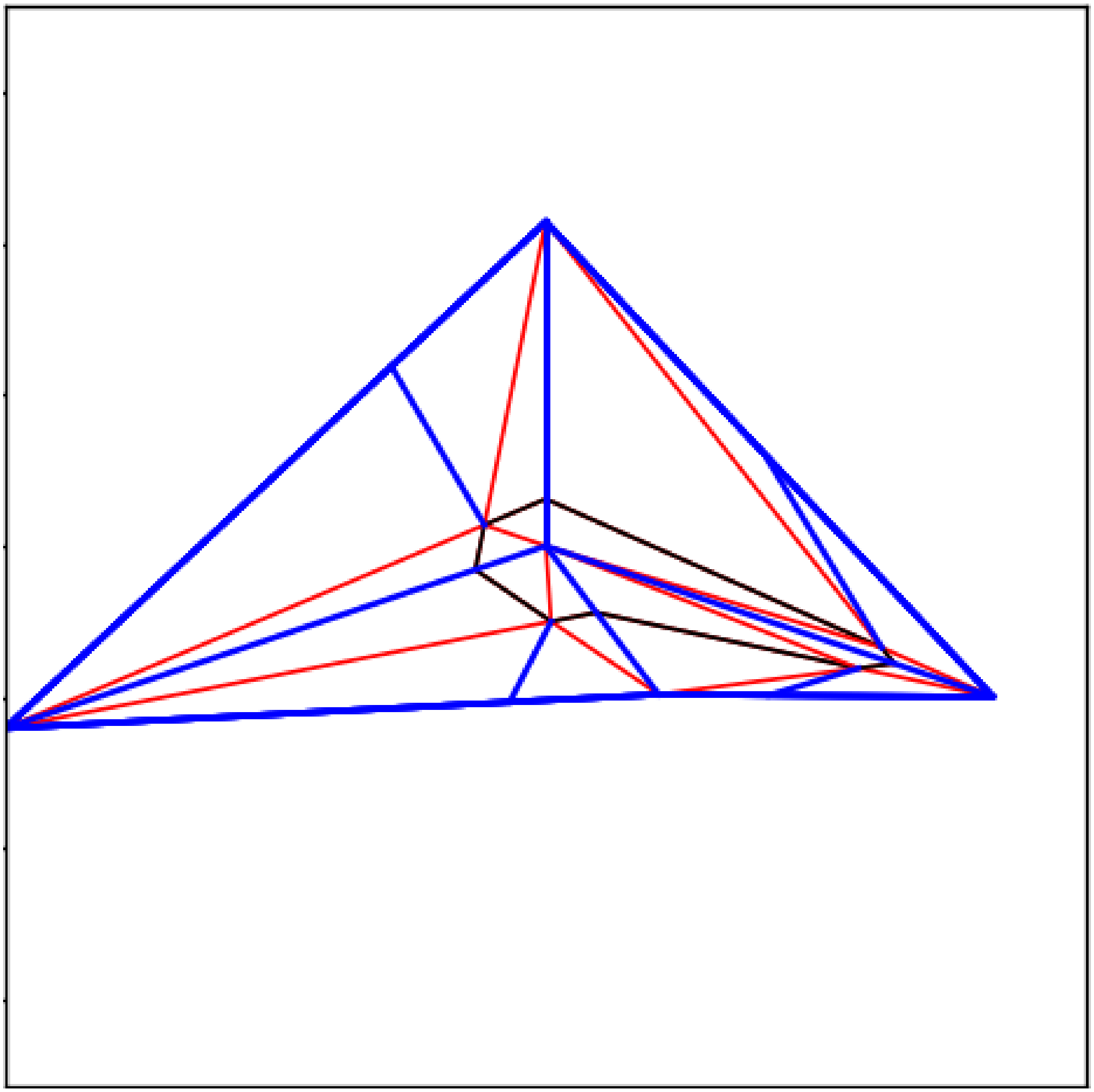
The total lengths of flaps: [3.464470402135396, 5.553452375610049, 3.512416096115799]

The length of open-ended flaps: [3.4381955399990076, 5.26293136479214, 1.2721508625675515]

Original connectivity between flaps: [0.6638554146832885, 0.5051272102050278, 0.7040450880657435]

Final connectivity value: [1.219487049433334, 0.6813117154614303, 0.6077350409712303]

Appendix 16.5



Total number of flaps: 4

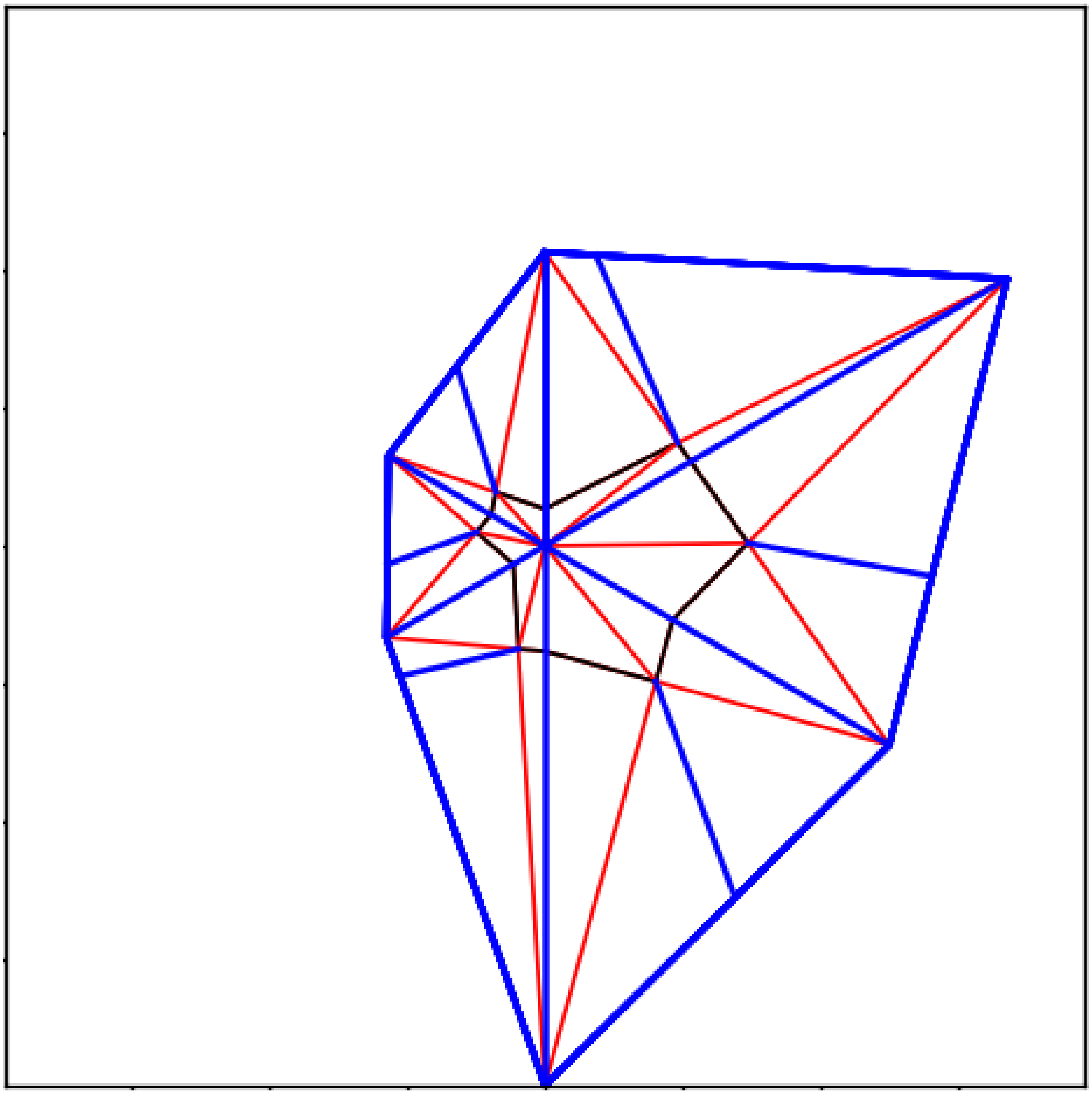
The total lengths of flaps: [4.282746436445714, 6.231407611233215, 2.4633665409542767, 7.530375062881111]

The length of open-ended flaps: [3.6618404174393886, 1.3742384609195133, 1.3591048957814738, 6.54779834406157]

Original connectivity between flaps: [0.3449502721206801, 0.517099409439606, 0.4999944538069857, 0.08686217507345129]

Final connectivity value: [0.34166321487220924, 0.3836942455006062, 0.837697574652114, 1.0896199357364507]

Appendix 16.6



Total number of flaps: 6

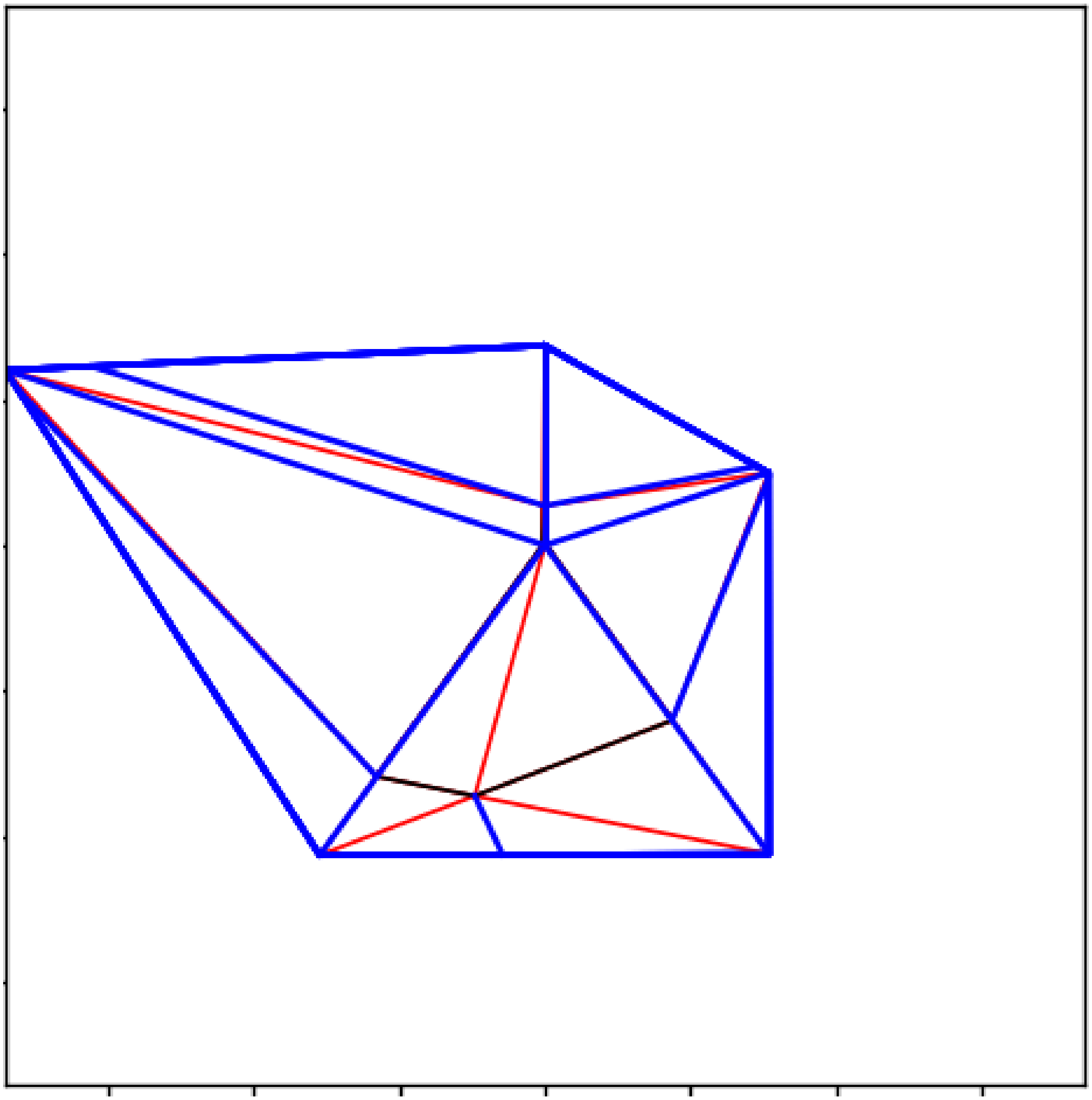
The total lengths of flaps: [4.255777963775013, 7.7432943310925575, 5.765953097730162, 7.820112260613582, 2.6538115670730855, 2.618405710296501]

The length of open-ended flaps: [3.7204529568503597, 5.2992473782993335, 3.6251695155373684, 6.2937690822203, 2.1251797302930466, 1.7192978286076213]

Original connectivity between flaps: [0.26464283222415985, 0.37343277188042057, 0.33292365424602643, 0.14369435790920182, 0.23190981751183692, 0.28120212259944977]

Final connectivity value: [1.8032199498953356, 1.6402298812097045, 1.9952423850108727, 1.4939517506156386, 2.126617078777873, 2.1404917702808905]

Appendix 16.7



Total number of flaps: 5

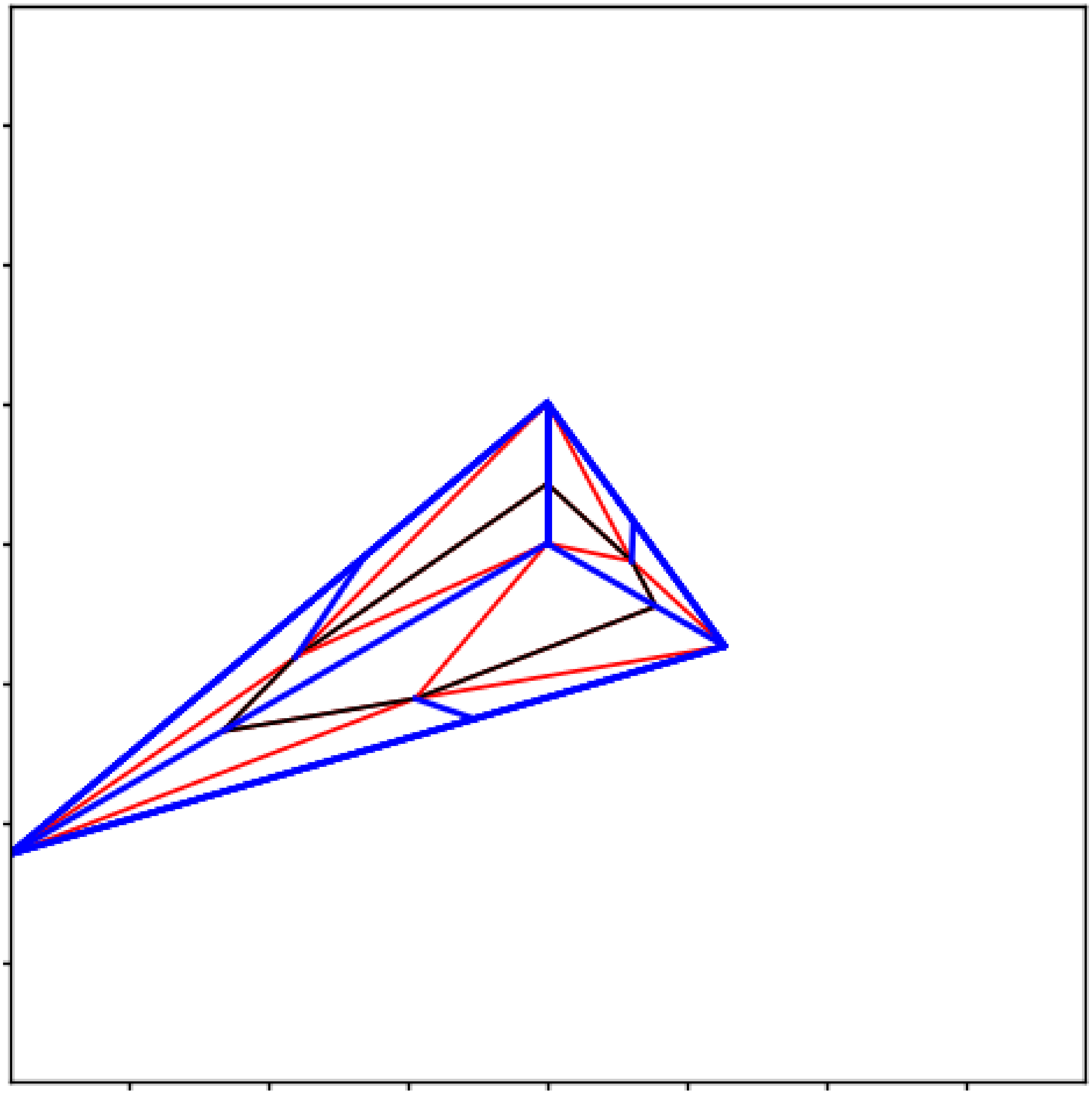
The total lengths of flaps: [3.4346681076382723, 4.041480600965569, 6.5535865745033215, 6.580155281748957, 9.746219805828762]

The length of open-ended flaps: [2.7546904998808497, 4.014203334390521, 2.8324186671531137, 1.6672885877945705, 9.7084718462198]

Original connectivity between flaps: [0.21942586571206557, 0.4026466222572944, 0.22848759772450722, 0.2592118831737529, 0.3043298426522313]

Final connectivity value: [2.3434490469607763, 1.0923373741793443, 0.33974324513549997, 1.3117656474140296, 1.8041319130917417]

Appendix 16.8



Total number of flaps: 3

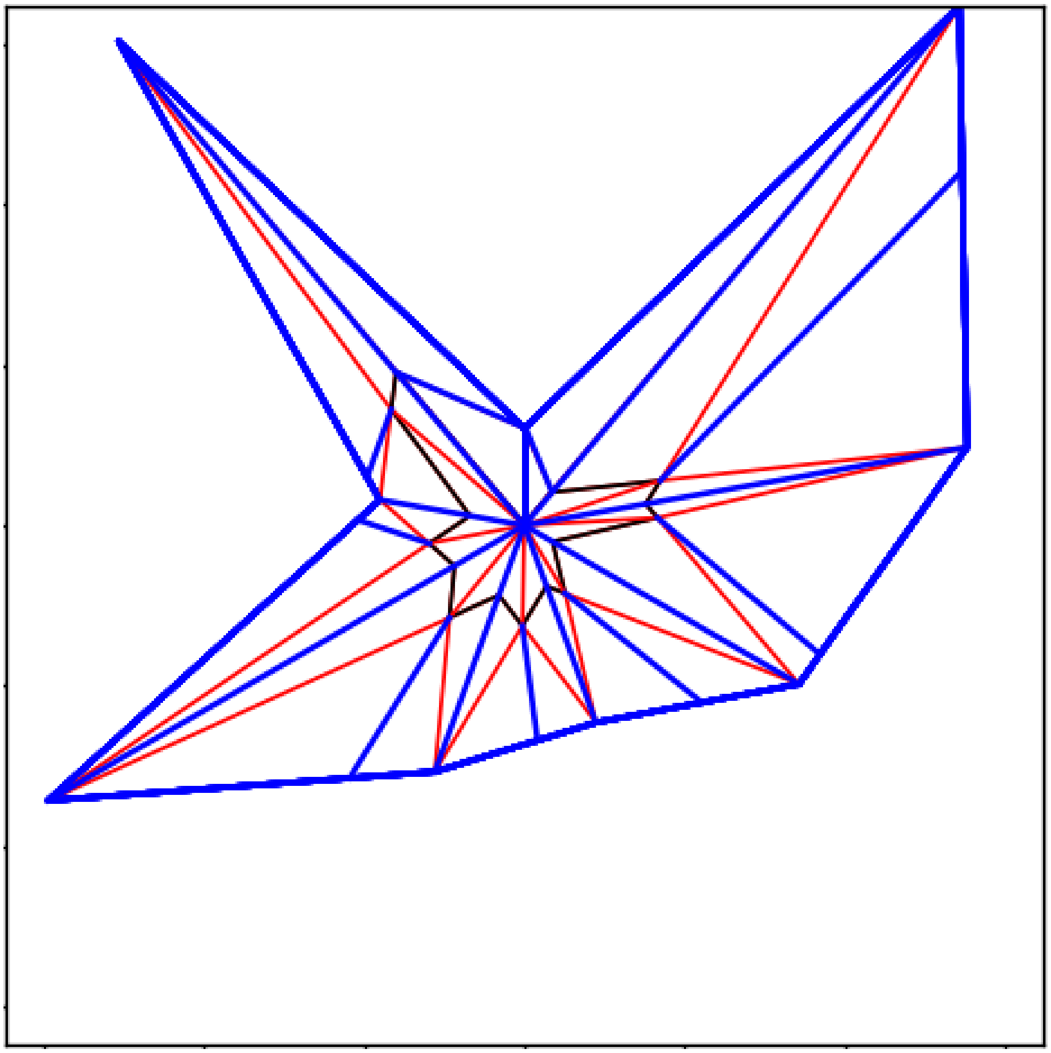
The total lengths of flaps: [2.0115733197759145, 2.9334970226902093, 8.885634531475933]

The length of open-ended flaps: [1.157124160870537, 1.1357906049293427, 3.506406130734023]

Original connectivity between flaps: [0.19726409808079415, 0.24267124446820976, 0.6705161503239744]

Final connectivity value: [0.2833785102974669, 0.18965739994772332, 0.21559858772217017]

Appendix 16.9



Total number of flaps: 9

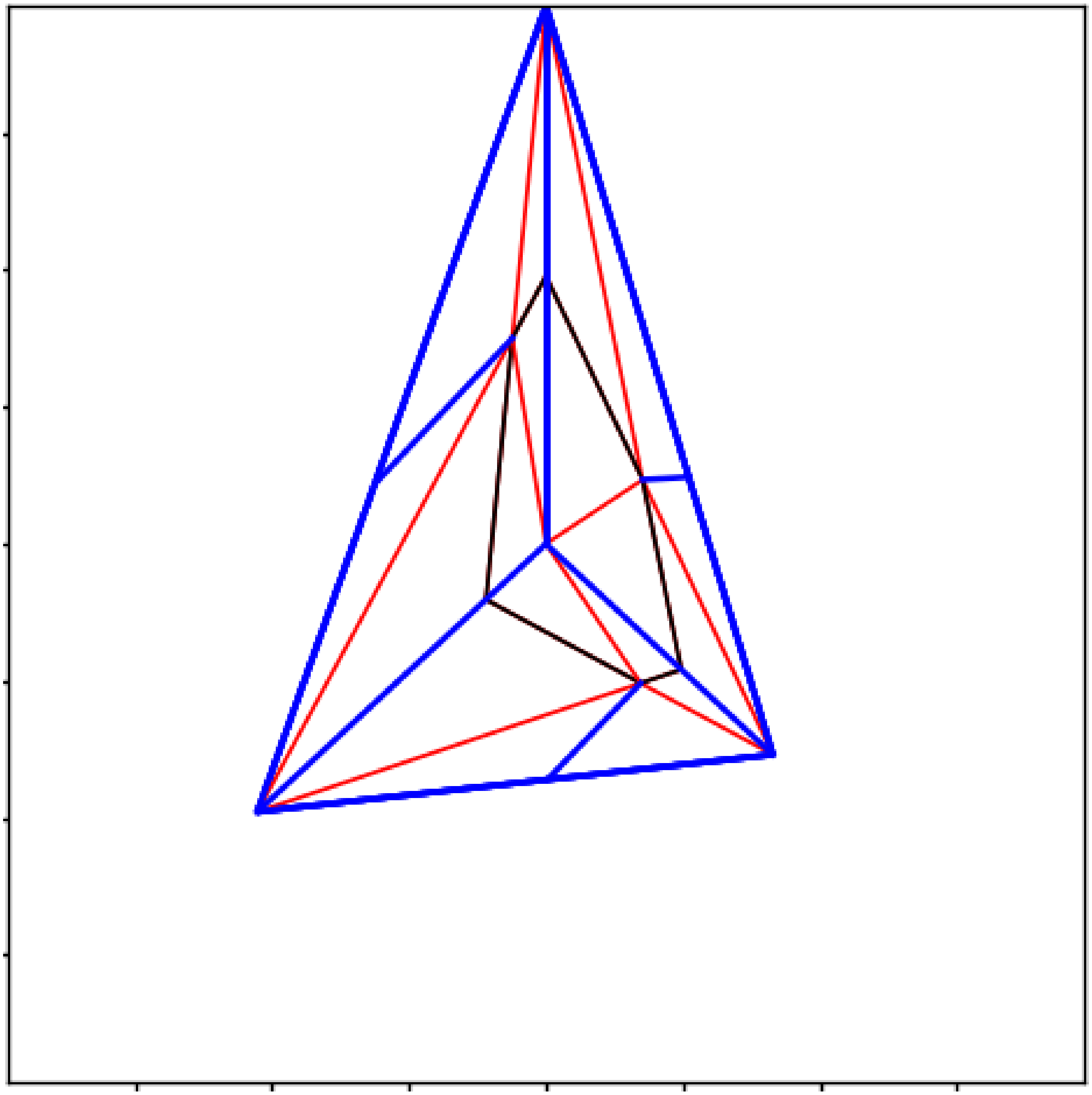
The total lengths of flaps: [1.2110161291068295, 8.454241668979076, 5.613548637679021, 3.960061983611369, 2.6160255165768724, 3.2727483756622218, 6.868013935955064, 1.829731902465634, 7.884522775537047]

The length of open-ended flaps: [1.194117763839045, 7.9168163061062575, 4.075281737727334, 3.5541048273324183, 1.8115528278541422, 2.3456316659684173, 5.859091569962198, 1.1408855496745924, 5.389985412314998]

Original connectivity between flaps: [0.6931094325865687, 0.07552344452186466, 0.556104143078683, 0.28724202657081843, 0.40548694833724475, 0.49023099649409607, 0.21707252765655738, 0.27626011550766094, 0.13194257981630897]

Final connectivity value: [1.4498170028574666, 4.750011850480868, 4.431253545914958, 4.326882255718954, 3.8947084774234595, 2.8715015027773094, 1.5672701104822206, 0.9825258838342412, 0.8846402471426738]

Appendix 16.10



Total number of flaps: 3

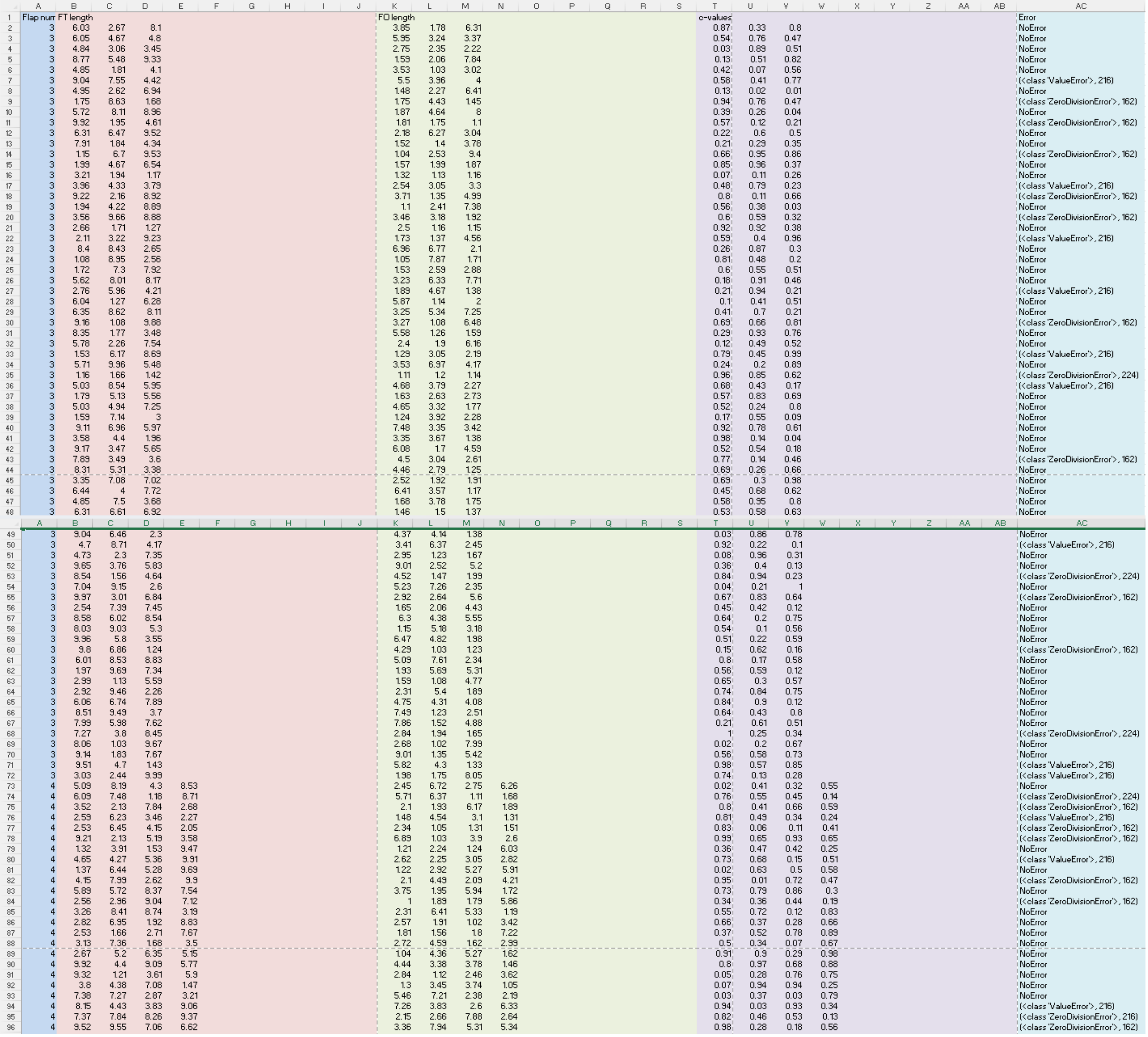
The total lengths of flaps: [9.829701951935462, 5.655647757017817, 7.189501466866154]

The length of open-ended flaps: [4.933024079281786, 2.282107117549569, 5.693972835576422]

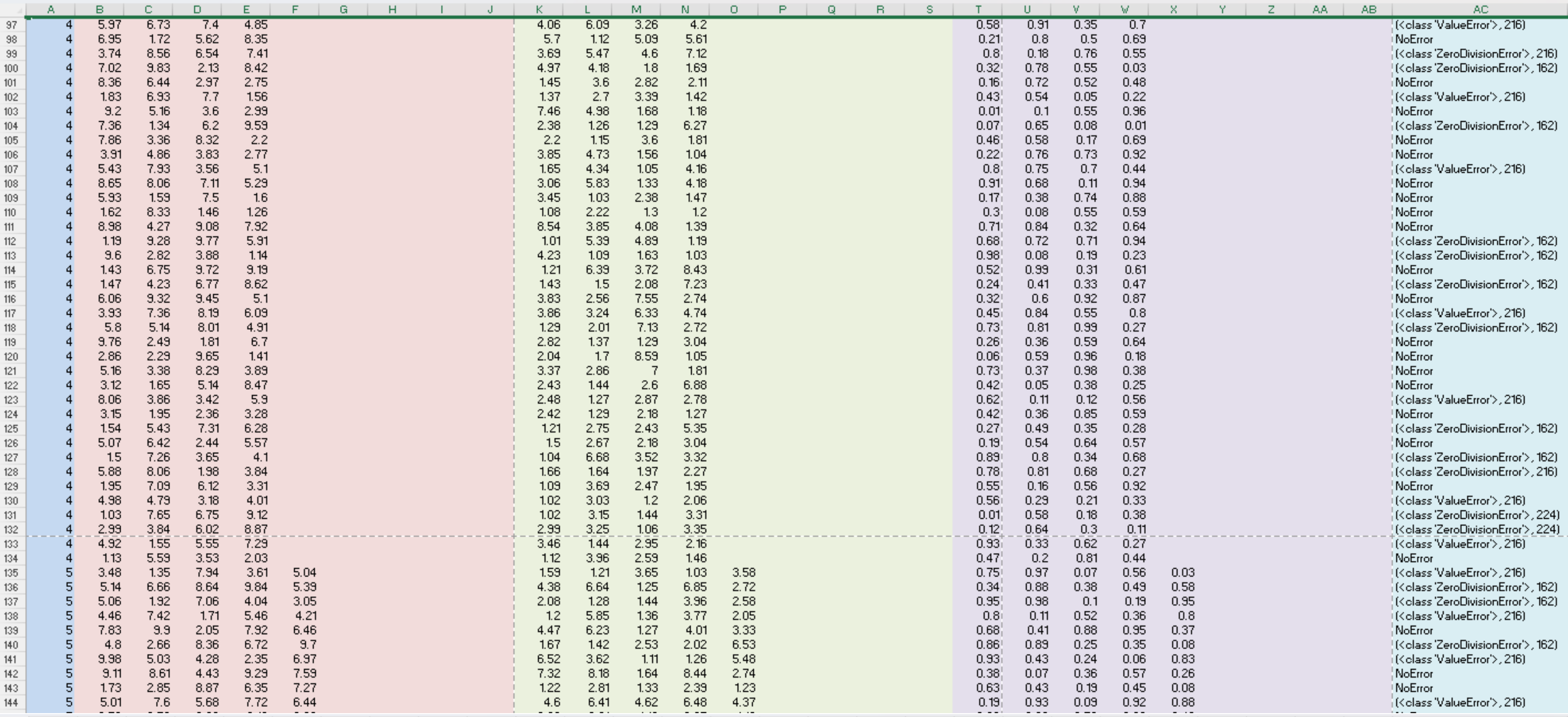
Original connectivity between flaps: [0.22056720804932473, 0.6906013433706388, 0.06410576782529087]

Final connectivity value: [0.25486541534639834, 0.709156180765215, 0.4619390035136482]

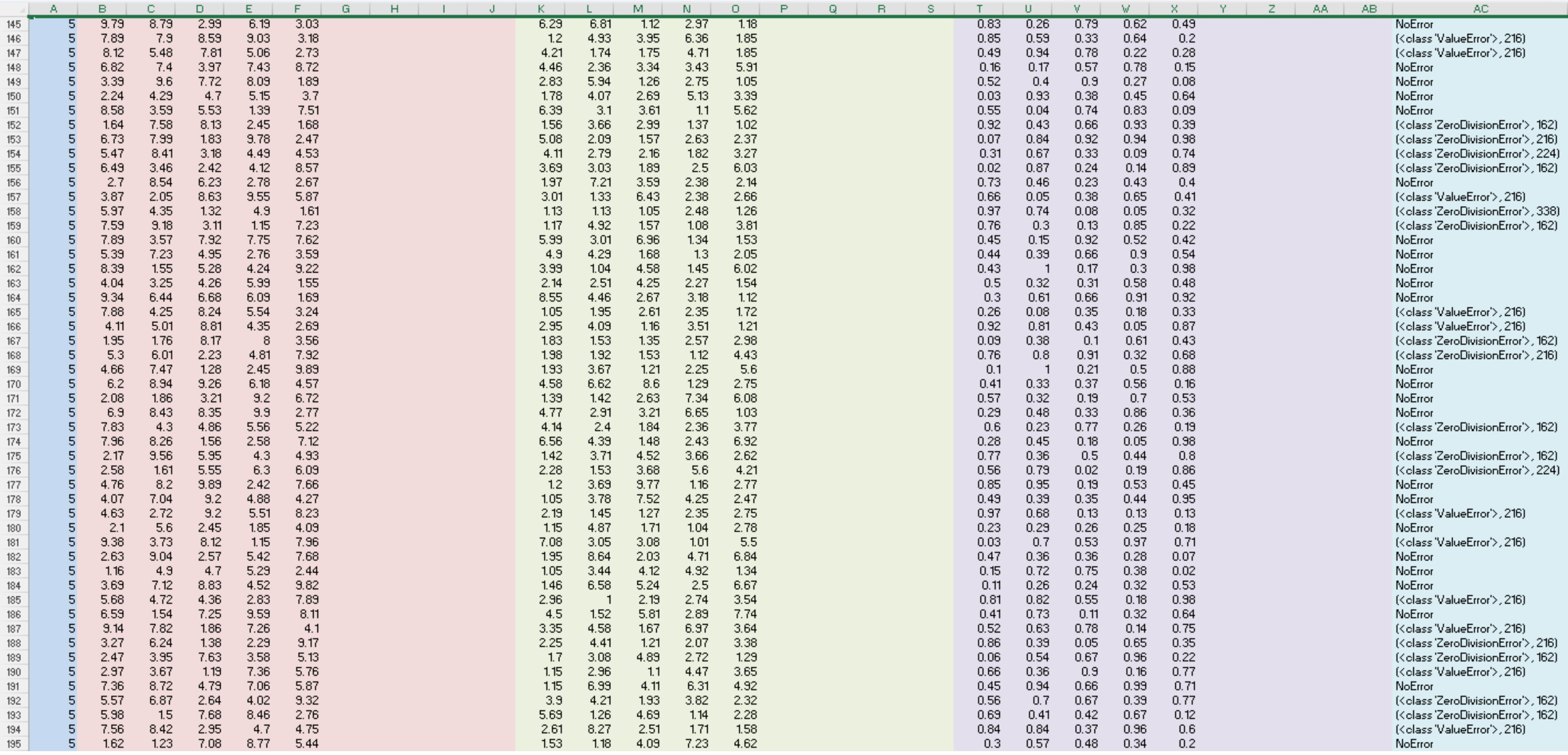
Appendix 17. Third test raw data



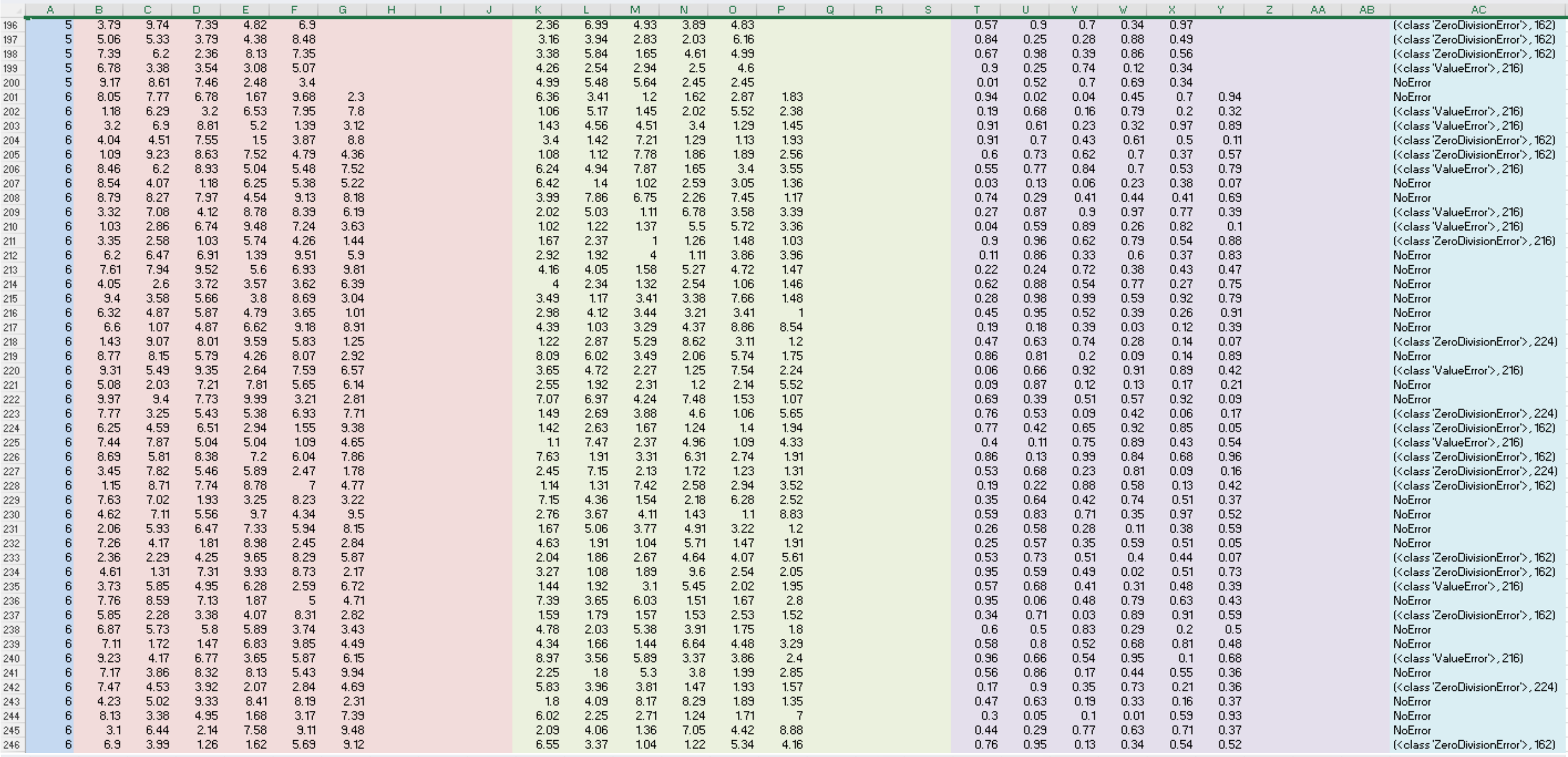
	A	B	C	D	E	K	L	M	N	T	U	V	W	AC
1	Flap num	FT length				FO length				c-values				Error
2	3	6.03	2.67	8.1		3.85	1.78	6.31		0.87	0.33	0.8		NoError
3	3	6.05	4.67	4.8		5.95	3.24	3.37		0.54	0.76	0.47		NoError
4	3	4.84	3.06	3.45		2.75	2.35	2.22		0.03	0.89	0.51		NoError
5	3	8.77	5.48	9.33		1.59	2.06	7.84		0.13	0.51	0.82		NoError
6	3	4.85	1.81	4.1		3.53	1.03	3.02		0.42	0.07	0.56		NoError
7	3	9.04	7.55	4.42		5.5	3.96	4		0.58	0.41	0.77		(<class 'ValueError'>, 216)
8	3	4.95	2.62	6.94		1.48	2.27	6.41		0.13	0.02	0.01		NoError
9	3	1.75	8.63	1.68		1.75	4.43	1.45		0.94	0.76	0.47		(<class 'ZeroDivisionError'>, 162)
10	3	5.72	8.11	8.96		1.87	4.64	8		0.39	0.26	0.04		NoError
11	3	9.92	1.95	4.61		1.81	1.75	1.1		0.57	0.12	0.21		(<class 'ZeroDivisionError'>, 162)
12	3	6.31	6.47	9.52		2.18	6.27	3.04		0.22	0.6	0.5		NoError
13	3	7.91	1.84	4.34		1.52	1.4	3.78		0.21	0.29	0.35		NoError
14	3	1.15	6.7	9.53		1.04	2.53	9.4		0.66	0.95	0.86		(<class 'ZeroDivisionError'>, 162)
15	3	1.99	4.67	6.54		1.57	1.99	1.87		0.85	0.96	0.37		NoError
16	3	3.21	1.94	1.17		1.32	1.13	1.16		0.07	0.11	0.26		NoError
17	3	3.96	4.33	3.79		2.54	3.05	3.3		0.48	0.79	0.23		(<class 'ValueError'>, 216)
18	3	9.22	2.16	8.92		3.71	1.35	4.99		0.8	0.11	0.66		(<class 'ZeroDivisionError'>, 162)
19	3	1.94	4.22	8.89		1.1	2.41	7.38		0.56	0.38	0.03		NoError
20	3	3.56	9.66	8.88		3.46	3.18	1.92		0.6	0.59	0.32		(<class 'ZeroDivisionError'>, 162)
21	3	2.66	1.71	1.27		2.5	1.16	1.15		0.92	0.92	0.38		NoError
22	3	2.11	3.22	9.23		1.73	1.37	4.56		0.59	0.4	0.96		(<class 'ValueError'>, 216)
23	3	8.4	8.43	2.65		6.96	6.77	2.1		0.26	0.87	0.3		NoError
24	3	1.08	8.95	2.56		1.05	7.87	1.71		0.81	0.48	0.2		NoError
25	3	1.72	7.3	7.92		1.53	2.59	2.88		0.6	0.55	0.51		NoError
26	3	5.62	8.01	8.17		3.23	6.33	7.71		0.18	0.91	0.46		NoError
27	3	2.76	5.96	4.21		1.89	4.67	1.38		0.21	0.94	0.21		(<class 'ValueError'>, 216)
28	3	6.04	1.27	6.28		5.87	1.14	2		0.1	0.41	0.51		NoError
29	3	6.35	8.62	8.11		3.25	5.34	7.25		0.41	0.7	0.21		NoError
30	3	9.16	1.08	9.88		3.27	1.08	6.48		0.69	0.66	0.81		(<class 'ZeroDivisionError'>, 162)
31	3	8.35	1.77	3.48		5.58	1.26	1.59		0.29	0.93	0.76		NoError
32	3	5.78	2.26	7.54		2.4	1.9	6.16		0.12	0.49	0.52		NoError
33	3	1.53	6.17	8.69		1.29	3.05	2.19		0.79	0.45	0.99		(<class 'ValueError'>, 216)
34	3	5.71	9.96	5.48		3.53	6.97	4.17		0.24	0.2	0.89		NoError
35	3	1.16	1.66	1.42		1.11	1.2	1.14		0.96	0.85	0.62		(<class 'ZeroDivisionError'>, 224)
36	3	5.03	8.54	5.95		4.68	3.79	2.27		0.68	0.43	0.17		(<class 'ValueError'>, 216)
37	3	1.79	5.13	5.56		1.63	2.63	2.73		0.57	0.83	0.69		NoError
38	3	5.03	4.94	7.25		4.65	3.32	1.77		0.52	0.24	0.8		NoError
39	3	1.59	7.14	3		1.24	3.92	2.28		0.17	0.55	0.09		NoError
40	3	9.11	6.96	5.97		7.48	3.35	3.42		0.92	0.78	0.61		NoError
41	3	3.58	4.4	1.96		3.35	3.67	1.38		0.98	0.14	0.04		NoError
42	3	9.17	3.47	5.65		6.08	1.7	4.59		0.52	0.54	0.18		NoError
43	3	7.89	3.49	3.6		4.5	3.04	2.61		0.77	0.14	0.46		(<class 'ZeroDivisionError'>, 162)
44	3	8.31	5.31	3.38		4.46	2.79	1.25		0.69	0.26	0.66		NoError
45	3	3.35	7.08	7.02		2.52	1.92	1.91		0.69	0.3	0.98		NoError
46	3	6.44	4	7.72		6.41	3.57	1.17		0.45	0.68	0.62		NoError
47	3	4.85	7.5	3.68		1.68	3.78	1.75		0.58	0.95	0.8		NoError
48	3	6.31	6.61	6.92		1.46	1.5	1.37		0.53	0.58	0.63		NoError
49	3	9.04	6.46	2.3		4.37	4.14	1.38		0.03	0.86	0.78		NoError
50	3	4.7	8.71	4.17		3.41	6.37	2.45		0.92	0.22	0.1		(<class 'ValueError'>, 216)
51	3	4.73	2.3	7.35		2.95	1.23	1.67		0.08	0.96	0.31		NoError
52	3	9.65	3.76	5.83		9.01	2.52	5.2		0.36	0.4	0.13		NoError
53	3	8.54	1.56	4.64		4.52	1.47	1.99		0.84	0.94	0.23		(<class 'ZeroDivisionError'>, 224)
54	3	7.04	9.15	2.6		5.23	7.26	2.35		0.04	0.21	1		NoError
55	3	9.97	3.01	6.84		2.92	2.64	5.6		0.67	0.83	0.64		(<class 'ZeroDivisionError'>, 162)
56	3	2.54	7.39	7.45		1.65	2.06	4.43		0.45	0.42	0.12		NoError
57	3	8.58	6.02	8.54		6.3	4.38	5.55		0.64	0.2	0.75		NoError
58	3	8.03	9.03	5.3		1.15	5.18	3.18		0.54	0.1	0.56		NoError
59	3	9.96	5.8	3.55		6.47	4.82	1.98		0.51	0.22	0.59		NoError
60	3	9.8	6.86	1.24		4.29	1.03	1.23		0.15	0.62	0.16		(<class 'ZeroDivisionError'>, 162)
61	3	6.01	8.53	8.83		5.09	7.61	2.34		0.8	0.17	0.58		NoError
62	3	1.97	9.69	7.34		1.93	5.69	5.31		0.56	0.59	0.12		NoError
63	3	2.99	1.13	5.59		1.59	1.08	4.77		0.65	0.3	0.57		NoError
64	3	2.92	9.46	2.26		2.31	5.4	1.89		0.74	0.84	0.75		NoError
65	3	6.06	6.74	7.89		4.75	4.31	4.08		0.84	0.9	0.12		NoError
66	3	8.51	9.49	3.7		7.49	1.23	2.51		0.64	0.43	0.8		NoError
67	3	7.99	5.98	7.62		7.86	1.52	4.88		0.21	0.61	0.51		NoError
68	3	7.27	3.8	8.45		2.84	1.94	1.65		1	0.25	0.34		(<class 'ZeroDivisionError'>, 224)
69	3	8.06	1.03	9.67		2.68	1.02	7.99		0.02	0.2	0.67		NoError
70	3	9.14	1.83	7.67		9.01	1.35	5.42		0.56	0.58	0.73		NoError
71	3	9.51	4.7	1.43		5.82	4.3	1.33		0.98	0.57	0.85		(<class 'ValueError'>, 216)
72	3	3.03	2.44	9.99		1.98	1.75	8.05		0.74	0.13	0.28		(<class 'ValueError'>, 216)
73	4	5.09	8.19	4.3	8.53	2.45	6.72	2.75	6.26	0.02	0.41	0.32	0.55	NoError
74	4	6.09	7.48	1.18	8.71	5.71	6.37	1.11	1.68	0.76	0.55	0.45	0.14	(<class 'ZeroDivisionError'>, 224)
75	4	3.52	2.13	7.84	2.68	2.1	1.93	6.17	1.89	0.8	0.41	0.66	0.59	(<class 'ZeroDivisionError'>, 162)
76	4	2.59	6.23	3.46	2.27	1.48	4.54	3.1	1.31	0.81	0.49	0.34	0.24	(<class 'ValueError'>, 216)
77	4	2.53	6.45	4.15	2.05	2.34	1.05	1.31	1.51	0.83	0.06	0.11	0.41	(<class 'ZeroDivisionError'>, 162)
78	4	9.21	2.13	5.19	3.58	6.89	1.03	3.9	2.6	0.99	0.65	0.93	0.65	(<class 'ZeroDivisionError'>, 162)
79	4	1.32	3.91	1.53	9.47	1.21	2.24	1.24	6.03	0.36	0.47	0.42	0.25	NoError
80	4	4.65	4.27	5.36	9.91	2.62	2.25	3.05	2.82	0.73	0.68	0.15	0.51	(<class 'ValueError'>, 216)
81	4	1.37	6.44	5.28	9.69	1.22	2.92	5.27	5.91	0.02	0.63	0.5	0.58	NoError
82	4	4.15	7.99	2.62	9.9	2.1	4.49	2.09	4.21	0.95	0.01	0.72	0.47	(<class 'ZeroDivisionError'>, 162)
83	4	5.89	5.72	8.37	7.54	3.75	1.95	5.94	1.72	0.73	0.79	0.86	0.3	NoError
84	4	2.56	2.96	9.04	7.12	1	1.89	1.79	5.86	0.34	0.36	0.44	0.19	(<class 'ZeroDivisionError'>, 162)
85	4	3.26	8.41	8.74	3.19	2.31	6.41	5.33	1.19	0.55	0.72	0.12	0.83	NoError
86	4	2.82	6.95	1.92	8.83	2.57	1.91	1.02	3.42	0.66	0.37	0.28	0.66	NoError
87	4	2.53	1.66	2.71	7.67	1.81	1.56	1.8	7.22	0.37	0.52	0.78	0.89	NoError
88	4	3.13	7.36	1.68	3.5	2.72	4.59	1.62	2.99	0.5	0.34	0.07	0.67	NoError
89	4	2.67	5.2	6.35	5.15	1.04	4.36	5.27	1.62	0.91	0.9	0.29	0.98	NoError
90	4	9.92	4.4	9.09	5.77	4.44	3.38	3.78	1.46	0.8	0.97	0.68	0.88	NoError
91	4	9.32	1.21	3.61	5.9	2.84	1.12	2.46	3.62	0.05	0.28	0.76	0.75	NoError
92	4	3.8	4.38	7.08	1.47	1.3	3.45	3.74	1.05	0.07	0.94	0.94	0.25	NoError
93	4	7.38	7.27	2.87	3.21	5.46	7.21	2.38	2.19	0.03	0.37	0.03	0.79	NoError
94	4	8.15	4.43	3.83	9.06	7.26	3.83	2.6	6.33	0.94	0.03	0.93	0.34	(<class 'ValueError'>, 216)
95	4	7.37	7.84	8.26	9.37	2.15	2.66	7.88	2.64	0.82	0.46	0.53	0.13	(<class 'ZeroDivisionError'>, 216)
96	4	9.52	9.55	7.06	6.62	3.36	7.94	5.31	5.34	0.98	0.28	0.18	0.56	(<class 'ZeroDivisionError'>, 162)



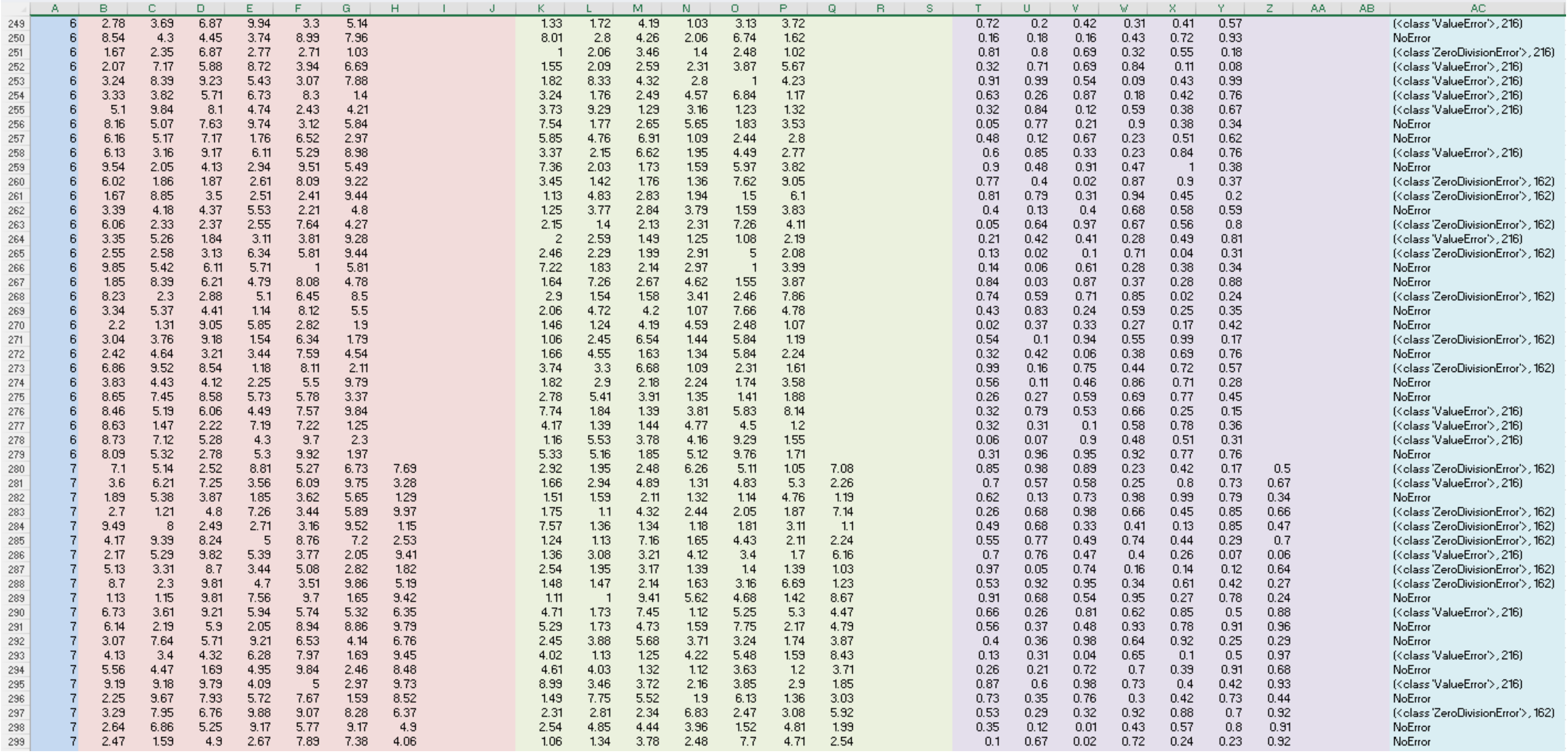
	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	AA	AB	AC
97	4	5.97	6.73	7.4	4.85						4.06	6.09	3.26	4.2						0.58	0.91	0.35	0.7						(<class 'ValueError'>, 216)
98	4	6.95	1.72	5.62	8.35						5.7	1.12	5.09	5.61						0.21	0.8	0.5	0.69						NoError
99	4	3.74	8.56	6.54	7.41						3.69	5.47	4.6	7.12						0.8	0.18	0.76	0.55						(<class 'ZeroDivisionError'>, 216)
100	4	7.02	9.83	2.13	8.42						4.97	4.18	1.8	1.69						0.32	0.78	0.55	0.03						(<class 'ZeroDivisionError'>, 162)
101	4	8.36	6.44	2.97	2.75						1.45	3.6	2.82	2.11						0.16	0.72	0.52	0.48						NoError
102	4	1.83	6.93	7.7	1.56						1.37	2.7	3.39	1.42						0.43	0.54	0.05	0.22						(<class 'ValueError'>, 216)
103	4	9.2	5.16	3.6	2.99						7.46	4.98	1.68	1.18						0.01	0.1	0.55	0.96						NoError
104	4	7.36	1.34	6.2	9.59						2.38	1.26	1.29	6.27						0.07	0.65	0.08	0.01						(<class 'ZeroDivisionError'>, 162)
105	4	7.86	3.36	8.32	2.2						2.2	1.15	3.6	1.81						0.46	0.58	0.17	0.69						NoError
106	4	3.91	4.86	3.83	2.77						3.85	4.73	1.56	1.04						0.22	0.76	0.73	0.92						NoError
107	4	5.43	7.93	3.56	5.1						1.65	4.34	1.05	4.16						0.8	0.75	0.7	0.44						(<class 'ValueError'>, 216)
108	4	8.65	8.06	7.11	5.29						3.06	5.83	1.33	4.18						0.91	0.68	0.11	0.94						NoError
109	4	5.93	1.59	7.5	1.6						3.45	1.03	2.38	1.47						0.17	0.38	0.74	0.88						NoError
110	4	1.62	8.33	1.46	1.26						1.08	2.22	1.3	1.2						0.3	0.08	0.55	0.59						NoError
111	4	8.98	4.27	9.08	7.92						8.54	3.85	4.08	1.39						0.71	0.84	0.32	0.64						NoError
112	4	1.19	9.28	9.77	5.91						1.01	5.39	4.89	1.19						0.68	0.72	0.71	0.94						(<class 'ZeroDivisionError'>, 162)
113	4	9.6	2.82	3.88	1.14						4.23	1.09	1.63	1.03						0.98	0.08	0.19	0.23						(<class 'ZeroDivisionError'>, 162)
114	4	1.43	6.75	9.72	9.19						1.21	6.39	3.72	8.43						0.52	0.99	0.31	0.61						NoError
115	4	1.47	4.23	6.77	8.62						1.43	1.5	2.08	7.23						0.24	0.41	0.33	0.47						(<class 'ZeroDivisionError'>, 162)
116	4	6.06	9.32	9.45	5.1						3.83	2.56	7.55	2.74						0.32	0.6	0.92	0.87						NoError
117	4	3.93	7.36	8.19	6.09						3.86	3.24	6.33	4.74						0.45	0.84	0.55	0.8						(<class 'ValueError'>, 216)
118	4	5.8	5.14	8.01	4.91						1.29	2.01	7.13	2.72						0.73	0.81	0.99	0.27						(<class 'ZeroDivisionError'>, 162)
119	4	9.76	2.49	1.81	6.7						2.82	1.37	1.29	3.04						0.26	0.36	0.59	0.64						NoError
120	4	2.86	2.29	9.65	1.41						2.04	1.7	8.59	1.05						0.06	0.59	0.96	0.18						NoError
121	4	5.16	3.38	8.29	3.89						3.37	2.86	7	1.81						0.73	0.37	0.98	0.38						NoError
122	4	3.12	1.65	5.14	8.47						2.43	1.44	2.6	6.88						0.42	0.05	0.38	0.25						NoError
123	4	8.06	3.86	3.42	5.9						2.48	1.27	2.87	2.78						0.62	0.11	0.12	0.56						(<class 'ValueError'>, 216)
124	4	3.15	1.95	2.36	3.28						2.42	1.29	2.18	1.27						0.42	0.36	0.85	0.59						NoError
125	4	1.54	5.43	7.31	6.28						1.21	2.75	2.43	5.35						0.27	0.49	0.35	0.28						(<class 'ZeroDivisionError'>, 162)
126	4	5.07	6.42	2.44	5.57						1.5	2.67	2.18	3.04						0.19	0.54	0.64	0.57						NoError
127	4	1.5	7.26	3.65	4.1						1.04	6.68	3.52	3.32						0.89	0.8	0.34	0.68						(<class 'ZeroDivisionError'>, 162)
128	4	5.88	8.06	1.98	3.84						1.66	1.64	1.97	2.27						0.78	0.81	0.68	0.27						(<class 'ZeroDivisionError'>, 216)
129	4	1.95	7.09	6.12	3.31						1.09	3.69	2.47	1.95						0.55	0.16	0.56	0.92						NoError
130	4	4.98	4.79	3.18	4.01						1.02	3.03	1.2	2.06						0.56	0.29	0.21	0.33						(<class 'ValueError'>, 216)
131	4	1.03	7.65	6.75	9.12						1.02	3.15	1.44	3.31						0.01	0.58	0.18	0.38						(<class 'ZeroDivisionError'>, 224)
132	4	2.99	3.84	6.02	8.87						2.99	3.25	1.06	3.35						0.12	0.64	0.3	0.11						(<class 'ZeroDivisionError'>, 224)
133	4	4.92	1.55	5.55	7.29						3.46	1.44	2.95	2.16						0.93	0.33	0.62	0.27						(<class 'ValueError'>, 216)
134	4	1.13	5.59	3.53	2.03						1.12	3.96	2.59	1.46						0.47	0.2	0.81	0.44						NoError
135	5	3.48	1.35	7.94	3.61	5.04					1.59	1.21	3.65	1.03	3.58					0.75	0.97	0.07	0.56	0.03					(<class 'ValueError'>, 216)
136	5	5.14	6.66	8.64	9.84	5.39					4.38	6.64	1.25	6.85	2.72					0.34	0.88	0.38	0.49	0.58					(<class 'ZeroDivisionError'>, 162)
137	5	5.06	1.92	7.06	4.04	3.05					2.08	1.28	1.44	3.96	2.58					0.95	0.98	0.1	0.19	0.95					(<class 'ZeroDivisionError'>, 162)
138	5	4.46	7.42	1.71	5.46	4.21					1.2	5.85	1.36	3.77	2.05					0.8	0.11	0.52	0.36	0.8					(<class 'ValueError'>, 216)
139	5	7.83	9.9	2.05	7.92	6.46					4.47	6.23	1.27	4.01	3.33					0.68	0.41	0.88	0.95	0.37					NoError
140	5	4.8	2.66	8.36	6.72	9.7					1.67	1.42	2.53	2.02	6.53					0.86	0.89	0.25	0.35	0.08					(<class 'ZeroDivisionError'>, 162)
141	5	9.98	5.03	4.28	2.35	6.97					6.52	3.62	1.11	1.26	5.48					0.93	0.43	0.24	0.06	0.83					(<class 'ValueError'>, 216)
142	5	9.11	8.61	4.43	9.29	7.59					7.32	8.18	1.64	8.44	2.74					0.38	0.07	0.36	0.57	0.26					NoError
143	5	1.73	2.85	8.87	6.35	7.27					1.22	2.81	1.33	2.39	1.23					0.63	0.43	0.19	0.45	0.08					NoError
144	5	5.01	7.6	5.68	7.72	6.44					4.6	6.41	4.62	6.48	4.37					0.19	0.93	0.09	0.92	0.88					(<class 'ValueError'>, 216)



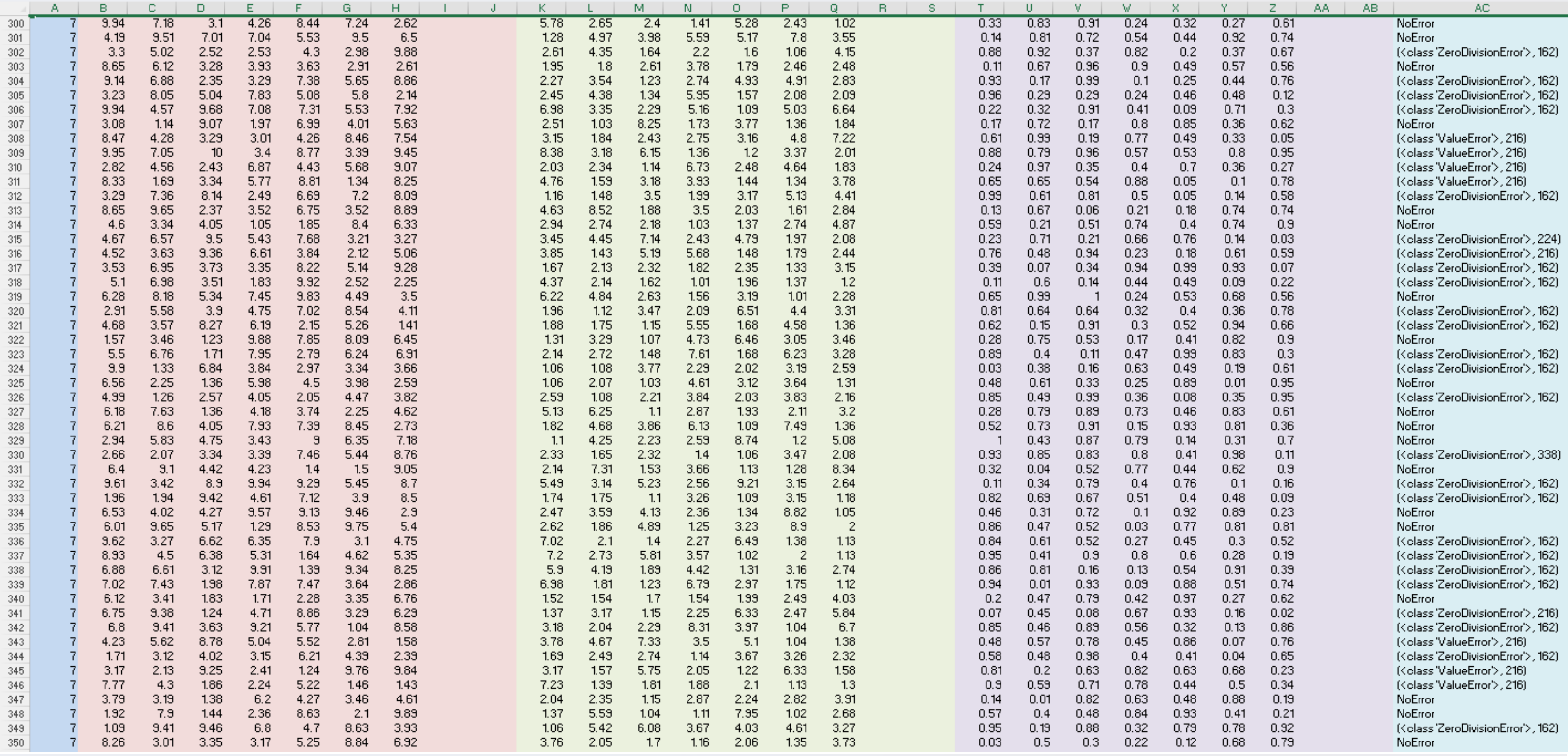
	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	AA	AB	AC
145	5	9.79	8.79	2.99	6.19	3.03					6.29	6.81	1.12	2.97	1.18					0.83	0.26	0.79	0.62	0.49					NoError
146	5	7.89	7.9	8.59	9.03	3.18					1.2	4.93	3.95	6.36	1.85					0.85	0.59	0.33	0.64	0.2					(<class 'ValueError'>, 216)
147	5	8.12	5.48	7.81	5.06	2.73					4.21	1.74	1.75	4.71	1.85					0.49	0.94	0.78	0.22	0.28					(<class 'ValueError'>, 216)
148	5	6.82	7.4	3.97	7.43	8.72					4.46	2.36	3.34	3.43	5.91					0.16	0.17	0.57	0.78	0.15					NoError
149	5	3.39	9.6	7.72	8.09	1.89					2.83	5.94	1.26	2.75	1.05					0.52	0.4	0.9	0.27	0.08					NoError
150	5	2.24	4.29	4.7	5.15	3.7					1.78	4.07	2.69	5.13	3.39					0.03	0.93	0.38	0.45	0.64					NoError
151	5	8.58	3.59	5.53	1.39	7.51					6.39	3.1	3.61	1.1	5.62					0.55	0.04	0.74	0.83	0.09					NoError
152	5	1.64	7.58	8.13	2.45	1.68					1.56	3.66	2.99	1.37	1.02					0.92	0.43	0.66	0.93	0.39					(<class 'ZeroDivisionError'>, 162)
153	5	6.73	7.99	1.83	9.78	2.47					5.08	2.09	1.57	2.63	2.37					0.07	0.84	0.92	0.94	0.98					(<class 'ZeroDivisionError'>, 216)
154	5	5.47	8.41	3.18	4.49	4.53					4.11	2.79	2.16	1.82	3.27					0.31	0.67	0.33	0.09	0.74					(<class 'ZeroDivisionError'>, 224)
155	5	6.49	3.46	2.42	4.12	8.57					3.69	3.03	1.89	2.5	6.03					0.02	0.87	0.24	0.14	0.89					(<class 'ZeroDivisionError'>, 162)
156	5	2.7	8.54	6.23	2.78	2.67					1.97	7.21	3.59	2.38	2.14					0.73	0.46	0.23	0.43	0.4					NoError
157	5	3.87	2.05	8.63	9.55	5.87					3.01	1.33	6.43	2.38	2.66					0.66	0.05	0.38	0.65	0.41					(<class 'ValueError'>, 216)
158	5	5.97	4.35	1.32	4.9	1.61					1.13	1.13	1.05	2.48	1.26					0.97	0.74	0.08	0.05	0.32					(<class 'ZeroDivisionError'>, 338)
159	5	7.59	9.18	3.11	1.15	7.23					1.17	4.92	1.57	1.08	3.81					0.76	0.3	0.13	0.85	0.22					(<class 'ZeroDivisionError'>, 162)
160	5	7.89	3.57	7.92	7.75	7.62					5.99	3.01	6.96	1.34	1.53					0.45	0.15	0.92	0.52	0.42					NoError
161	5	5.39	7.23	4.95	2.76	3.59					4.9	4.29	1.68	1.3	2.05					0.44	0.39	0.66	0.9	0.54					NoError
162	5	8.39	1.55	5.28	4.24	9.22					3.99	1.04	4.58	1.45	6.02					0.43	1	0.17	0.3	0.98					NoError
163	5	4.04	3.25	4.26	5.99	1.55					2.14	2.51	4.25	2.27	1.54					0.5	0.32	0.31	0.58	0.48					NoError
164	5	9.34	6.44	6.68	6.09	1.69					8.55	4.46	2.67	3.18	1.12					0.3	0.61	0.66	0.91	0.92					NoError
165	5	7.88	4.25	8.24	5.54	3.24					1.05	1.95	2.61	2.35	1.72					0.26	0.08	0.35	0.18	0.33					(<class 'ValueError'>, 216)
166	5	4.11	5.01	8.81	4.35	2.69					2.95	4.09	1.16	3.51	1.21					0.92	0.81	0.43	0.05	0.87					(<class 'ValueError'>, 216)
167	5	1.95	1.76	8.17	8	3.56					1.83	1.53	1.35	2.57	2.98					0.09	0.38	0.1	0.61	0.43					(<class 'ZeroDivisionError'>, 162)
168	5	5.3	6.01	2.23	4.81	7.92					1.98	1.92	1.53	1.12	4.43					0.76	0.8	0.91	0.32	0.68					(<class 'ZeroDivisionError'>, 216)
169	5	4.66	7.47	1.28	2.45	9.89					1.93	3.67	1.21	2.25	5.6					0.1	1	0.21	0.5	0.88					NoError
170	5	6.2	8.94	9.26	6.18	4.57					4.58	6.62	8.6	1.29	2.75					0.41	0.33	0.37	0.56	0.16					NoError
171	5	2.08	1.86	3.21	9.2	6.72					1.39	1.42	2.63	7.34	6.08					0.57	0.32	0.19	0.7	0.53					NoError
172	5	6.9	8.43	8.35	9.9	2.77					4.77	2.91	3.21	6.65	1.03					0.29	0.48	0.33	0.86	0.36					NoError
173	5	7.83	4.3	4.86	5.56	5.22					4.14	2.4	1.84	2.36	3.77					0.6	0.23	0.77	0.26	0.19					(<class 'ZeroDivisionError'>, 162)
174	5	7.96	8.26	1.56	2.58	7.12					6.56	4.39	1.48	2.43	6.92					0.28	0.45	0.18	0.05	0.98					NoError
175	5	2.17	9.56	5.95	4.3	4.93					1.42	3.71	4.52	3.66	2.62					0.77	0.36	0.5	0.44	0.8					(<class 'ZeroDivisionError'>, 162)
176	5	2.58	1.61	5.55	6.3	6.09					2.28	1.53	3.68	5.6	4.21					0.56	0.79	0.02	0.19	0.86					(<class 'ZeroDivisionError'>, 224)
177	5	4.76	8.2	9.89	2.42	7.66					1.2	3.69	9.77	1.16	2.77					0.85	0.95	0.19	0.53	0.45					NoError
178	5	4.07	7.04	9.2	4.88	4.27					1.05	3.78	7.52	4.25	2.47					0.49	0.39	0.35	0.44	0.95					NoError
179	5	4.63	2.72	9.2	5.51	8.23					2.19	1.45	1.27	2.35	2.75					0.97	0.68	0.13	0.13	0.13					(<class 'ValueError'>, 216)
180	5	2.1	5.6	2.45	1.85	4.09					1.15	4.87	1.71	1.04	2.78					0.23	0.29	0.26	0.25	0.18					NoError
181	5	9.38	3.73	8.12	1.15	7.96					7.08	3.05	3.08	1.01	5.5					0.03	0.7	0.53	0.97	0.71					(<class 'ValueError'>, 216)
182	5	2.63	9.04	2.57	5.42	7.68					1.95	8.64	2.03	4.71	6.84					0.47	0.36	0.36	0.28	0.07					NoError
183	5	1.16	4.9	4.7	5.29	2.44					1.05	3.44	4.12	4.92	1.34					0.15	0.72	0.75	0.38	0.02					NoError
184	5	3.69	7.12	8.83	4.52	9.82					1.46	6.58	5.24	2.5	6.67					0.11	0.26	0.24	0.32	0.53					NoError
185	5	5.68	4.72	4.36	2.83	7.89					2.96	1	2.19	2.74	3.54					0.81	0.82	0.55	0.18	0.98					(<class 'ValueError'>, 216)
186	5	6.59	1.54	7.25	9.59	8.11					4.5	1.52	5.81	2.89	7.74					0.41	0.73	0.11	0.32	0.64					NoError
187	5	9.14	7.82	1.86	7.26	4.1					3.35	4.58	1.67	6.97	3.64					0.52	0.63	0.78	0.14	0.75					(<class 'ValueError'>, 216)
188	5	3.27	6.24	1.38	2.29	9.17					2.25	4.41	1.21	2.07	3.38					0.86	0.39	0.05	0.65	0.35					(<class 'ZeroDivisionError'>, 216)
189	5	2.47	3.95	7.63	3.58	5.13					1.7	3.08	4.89	2.72	1.29					0.06	0.54	0.67	0.96	0.22					(<class 'ZeroDivisionError'>, 162)
190	5	2.97	3.67	1.19	7.36	5.76					1.15	2.96	1.1	4.47	3.65					0.66	0.36	0.9	0.16	0.77					(<class 'ValueError'>, 216)
191	5	7.36	8.72	4.79	7.06	5.87					1.15	6.99	4.11	6.31	4.92					0.45	0.94	0.66	0.99	0.71					NoError
192	5	5.57	6.87	2.64	4.02	9.32					3.9	4.21	1.93	3.82	2.32					0.56	0.7	0.67	0.39	0.77					(<class 'ZeroDivisionError'>, 162)
193	5	5.98	1.5	7.68	8.46	2.76					5.69	1.26	4.69	1.14	2.28					0.69	0.41	0.42	0.67	0.12					(<class 'ZeroDivisionError'>, 162)
194	5	7.56	8.42	2.95	4.7	4.75					2.61	8.27	2.51	1.71	1.58					0.84	0.84	0.37	0.96	0.6					(<class 'ValueError'>, 216)
195	5	1.62	1.23	7.08	8.77	5.44					1.53	1.18	4.09	7.23	4.62					0.3	0.57	0.48	0.34	0.2					NoError



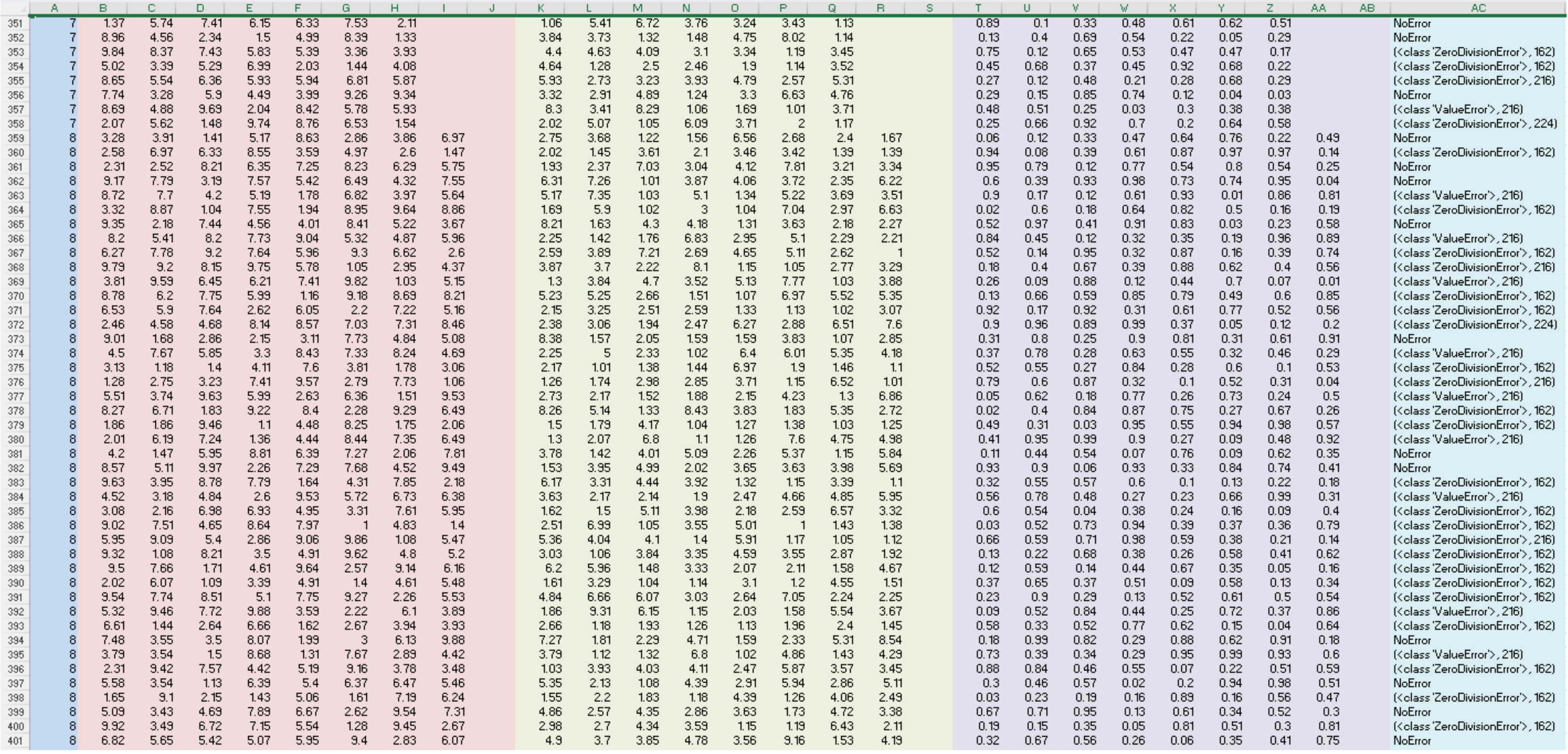
	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	AA	AB	AC
196	5	3.79	9.74	7.39	4.82	6.9					2.36	6.99	4.93	3.89	4.83					0.57	0.9	0.7	0.34	0.97					(<class 'ZeroDivisionError'>, 162)
197	5	5.06	5.33	3.79	4.38	8.48					3.16	3.94	2.83	2.03	6.16					0.84	0.25	0.28	0.88	0.49					(<class 'ZeroDivisionError'>, 162)
198	5	7.39	6.2	2.36	8.13	7.35					3.38	5.84	1.65	4.61	4.99					0.67	0.98	0.39	0.86	0.56					(<class 'ZeroDivisionError'>, 162)
199	5	6.78	3.38	3.54	3.08	5.07					4.26	2.54	2.94	2.5	4.6					0.9	0.25	0.74	0.12	0.34					(<class 'ValueError'>, 216)
200	5	9.17	8.61	7.46	2.48	3.4					4.99	5.48	5.64	2.45	2.45					0.01	0.52	0.7	0.69	0.34					NoError
201	6	8.05	7.77	6.78	1.67	9.68	2.3				6.36	3.41	1.2	1.62	2.87	1.83				0.94	0.02	0.04	0.45	0.7	0.94				NoError
202	6	1.18	6.29	3.2	6.53	7.95	7.8				1.06	5.17	1.45	2.02	5.52	2.38				0.19	0.68	0.16	0.79	0.2	0.32				(<class 'ValueError'>, 216)
203	6	3.2	6.9	8.81	5.2	1.39	3.12				1.43	4.56	4.51	3.4	1.29	1.45				0.91	0.61	0.23	0.32	0.97	0.89				(<class 'ValueError'>, 216)
204	6	4.04	4.51	7.55	1.5	3.87	8.8				3.4	1.42	7.21	1.29	1.13	1.93				0.91	0.7	0.43	0.61	0.5	0.11				(<class 'ZeroDivisionError'>, 162)
205	6	1.09	9.23	8.63	7.52	4.79	4.36				1.08	1.12	7.78	1.86	1.89	2.56				0.6	0.73	0.62	0.7	0.37	0.57				(<class 'ZeroDivisionError'>, 162)
206	6	8.46	6.2	8.93	5.04	5.48	7.52				6.24	4.94	7.87	1.65	3.4	3.55				0.55	0.77	0.84	0.7	0.53	0.79				(<class 'ValueError'>, 216)
207	6	8.54	4.07	1.18	6.25	5.38	5.22				6.42	1.4	1.02	2.59	3.05	1.36				0.03	0.13	0.06	0.23	0.38	0.07				NoError
208	6	8.79	8.27	7.97	4.54	9.13	8.18				3.99	7.86	6.75	2.26	7.45	1.17				0.74	0.29	0.41	0.44	0.41	0.69				NoError
209	6	3.32	7.08	4.12	8.78	8.39	6.19				2.02	5.03	1.11	6.78	3.58	3.39				0.27	0.87	0.9	0.97	0.77	0.39				(<class 'ValueError'>, 216)
210	6	1.03	2.86	6.74	9.48	7.24	3.63				1.02	1.22	1.37	5.5	5.72	3.36				0.04	0.59	0.89	0.26	0.82	0.1				(<class 'ValueError'>, 216)
211	6	3.35	2.58	1.03	5.74	4.26	1.44				1.67	2.37	1	1.26	1.48	1.03				0.9	0.96	0.62	0.79	0.54	0.88				(<class 'ZeroDivisionError'>, 216)
212	6	6.2	6.47	6.91	1.39	9.51	5.9				2.92	1.92	4	1.11	3.86	3.96				0.11	0.86	0.33	0.6	0.37	0.83				NoError
213	6	7.61	7.94	9.52	5.6	6.93	9.81				4.16	4.05	1.58	5.27	4.72	1.47				0.22	0.24	0.72	0.38	0.43	0.47				NoError
214	6	4.05	2.6	3.72	3.57	3.62	6.39				4	2.34	1.32	2.54	1.06	1.46				0.62	0.88	0.54	0.77	0.27	0.75				NoError
215	6	9.4	3.58	5.66	3.8	8.69	3.04				3.49	1.17	3.41	3.38	7.66	1.48				0.28	0.98	0.99	0.59	0.92	0.79				NoError
216	6	6.32	4.87	5.87	4.79	3.65	1.01				2.98	4.12	3.44	3.21	3.41	1				0.45	0.95	0.52	0.39	0.26	0.91				NoError
217	6	6.6	1.07	4.87	6.62	9.18	8.91				4.39	1.03	3.29	4.37	8.86	8.54				0.19	0.18	0.39	0.03	0.12	0.39				NoError
218	6	1.43	9.07	8.01	9.59	5.83	1.25				1.22	2.87	5.29	8.62	3.11	1.2				0.47	0.63	0.74	0.28	0.14	0.07				(<class 'ZeroDivisionError'>, 224)
219	6	8.77	8.15	5.79	4.26	8.07	2.92				8.09	6.02	3.49	2.06	5.74	1.75				0.86	0.81	0.2	0.09	0.14	0.89				NoError
220	6	9.31	5.49	9.35	2.64	7.59	6.57				3.65	4.72	2.27	1.25	7.54	2.24				0.06	0.66	0.92	0.91	0.89	0.42				(<class 'ValueError'>, 216)
221	6	5.08	2.03	7.21	7.81	5.65	6.14				2.55	1.92	2.31	1.2	2.14	5.52				0.09	0.87	0.12	0.13	0.17	0.21				NoError
222	6	9.97	9.4	7.73	9.99	3.21	2.81				7.07	6.97	4.24	7.48	1.53	1.07				0.69	0.39	0.51	0.57	0.92	0.09				NoError
223	6	7.77	3.25	5.43	5.38	6.93	7.71				1.49	2.69	3.88	4.6	1.06	5.65				0.76	0.53	0.09	0.42	0.06	0.17				(<class 'ZeroDivisionError'>, 224)
224	6	6.25	4.59	6.51	2.94	1.55	9.38				1.42	2.63	1.67	1.24	1.4	1.94				0.77	0.42	0.65	0.92	0.85	0.05				(<class 'ZeroDivisionError'>, 162)
225	6	7.44	7.87	5.04	5.04	1.09	4.65				1.1	7.47	2.37	4.96	1.09	4.33				0.4	0.11	0.75	0.89	0.43	0.54				(<class 'ValueError'>, 216)
226	6	8.69	5.81	8.38	7.2	6.04	7.86				7.63	1.91	3.31	6.31	2.74	1.91				0.86	0.13	0.99	0.84	0.68	0.96				(<class 'ZeroDivisionError'>, 162)
227	6	3.45	7.82	5.46	5.89	2.47	1.78				2.45	7.15	2.13	1.72	1.23	1.31				0.53	0.68	0.23	0.81	0.09	0.16				(<class 'ZeroDivisionError'>, 224)
228	6	1.15	8.71	7.74	8.78	7	4.77				1.14	1.31	7.42	2.58	2.94	3.52				0.19	0.22	0.88	0.58	0.13	0.42				(<class 'ZeroDivisionError'>, 162)
229	6	7.63	7.02	1.93	3.25	8.23	3.22				7.15	4.36	1.54	2.18	6.28	2.52				0.35	0.64	0.42	0.74	0.51	0.37				NoError
230	6	4.62	7.11	5.56	9.7	4.34	9.5				2.76	3.67	4.11	1.43	1.1	8.83				0.59	0.83	0.71	0.35	0.97	0.52				NoError
231	6	2.06	5.93	6.47	7.33	5.94	8.15				1.67	5.06	3.77	4.91	3.22	1.2				0.26	0.58	0.28	0.11	0.38	0.59				NoError
232	6	7.26	4.17	1.81	8.98	2.45	2.84				4.63	1.91	1.04	5.71	1.47	1.91				0.25	0.57	0.35	0.59	0.51	0.05				NoError
233	6	2.36	2.29	4.25	9.65	8.29	5.87				2.04	1.86	2.67	4.64	4.07	5.61				0.53	0.73	0.51	0.4	0.44	0.07				(<class 'ZeroDivisionError'>, 162)
234	6	4.61	1.31	7.31	9.93	8.73	2.17				3.27	1.08	1.89	9.6	2.54	2.05				0.95	0.59	0.49	0.02	0.51	0.73				(<class 'ZeroDivisionError'>, 162)
235	6	3.73	5.85	4.95	6.28	2.59	6.72				1.44	1.92	3.1	5.45	2.02	1.95				0.57	0.68	0.41	0.31	0.48	0.39				(<class 'ValueError'>, 216)
236	6	7.76	8.59	7.13	1.87	5	4.71				7.39	3.65	6.03	1.51	1.67	2.8				0.95	0.06	0.48	0.79	0.63	0.43				NoError
237	6	5.85	2.28	3.38	4.07	8.31	2.82				1.59	1.79	1.57	1.53	2.53	1.52				0.34	0.71	0.03	0.89	0.91	0.59				(<class 'ZeroDivisionError'>, 162)
238	6	6.87	5.73	5.8	5.89	3.74	3.43				4.78	2.03	5.38	3.91	1.75	1.8				0.6	0.5	0.83	0.29	0.2	0.5				NoError
239	6	7.11	1.72	1.47	6.83	9.85	4.49				4.34	1.66	1.44	6.64	4.48	3.29				0.58	0.8	0.52	0.68	0.81	0.48				NoError
240	6	9.23	4.17	6.77	3.65	5.87	6.15				8.97	3.56	5.89	3.37	3.86	2.4				0.96	0.66	0.54	0.95	0.1	0.68				(<class 'ValueError'>, 216)
241	6	7.17	3.86	8.32	8.13	5.43	9.94				2.25	1.8	5.3	3.8	1.99	2.85				0.56	0.86	0.17	0.44	0.55	0.36				NoError
242	6	7.47	4.53	3.92	2.07	2.84	4.69				5.83	3.96	3.81	1.47	1.93	1.57				0.17	0.9	0.35	0.73	0.21	0.36				(<class 'ZeroDivisionError'>, 224)
243	6	4.23	5.02	9.33	8.41	8.19	2.31				1.8	4.09	8.17	8.29	1.89	1.35				0.47	0.63	0.19	0.33	0.16	0.37				NoError
244	6	8.13	3.38	4.95	1.68	3.17	7.39				6.02	2.25	2.71	1.24	1.71	7				0.3	0.05	0.1	0.01	0.59	0.93				NoError
245	6	3.1	6.44	2.14	7.58	9.11	9.48				2.09	4.06	1.36	7.05	4.42	8.88				0.44	0.29	0.77	0.63	0.71	0.37				NoError
246	6	6.9	3.99	1.26	1.62	5.69	9.12				6.55	3.37	1.04	1.22	5.34	4.16				0.76	0.95	0.13	0.34	0.54	0.52				(<class 'ZeroDivisionError'>, 162)



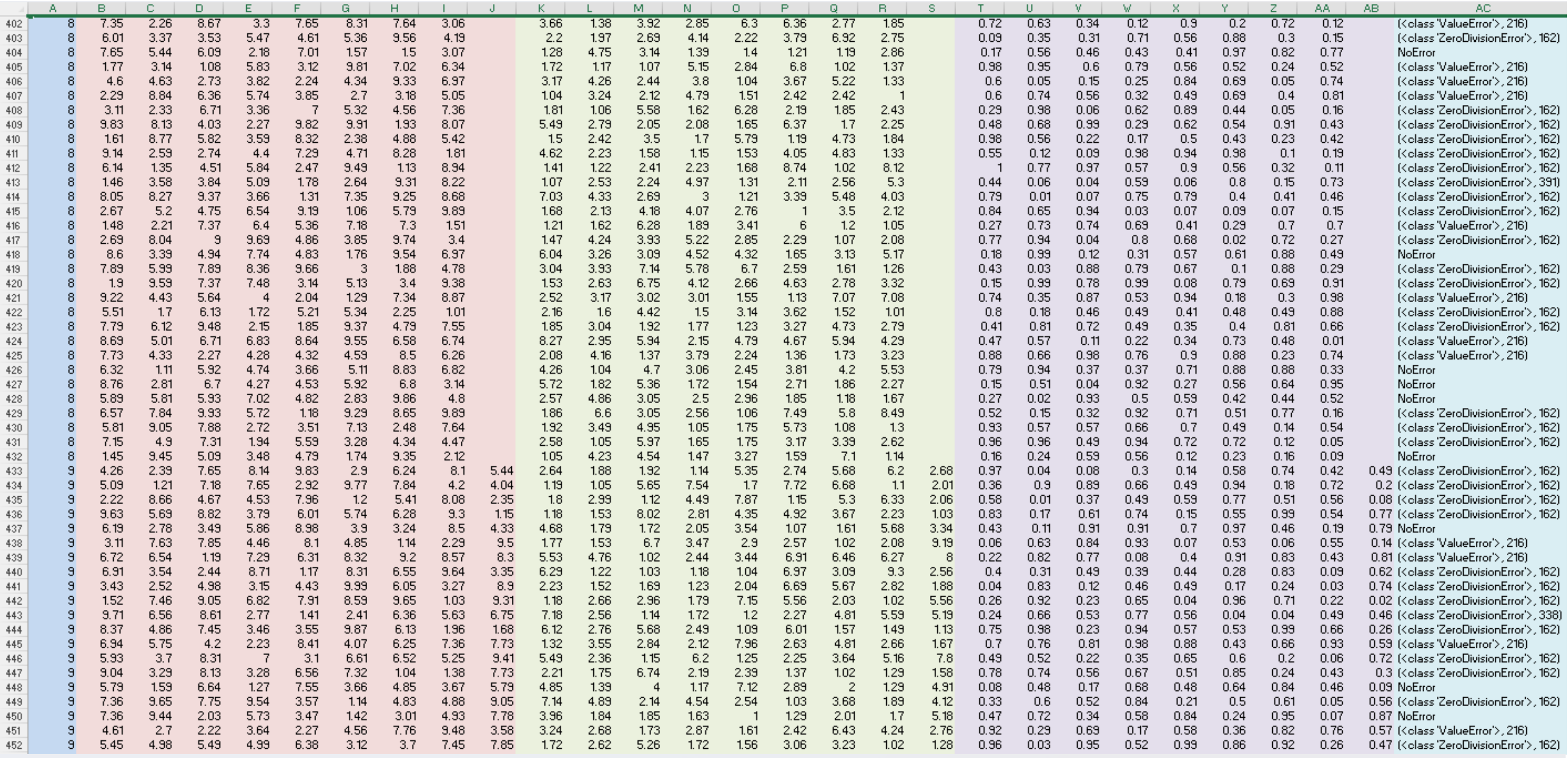
	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	AA	AB	AC
249	6	2.78	3.69	6.87	9.94	3.3	5.14				1.33	1.72	4.19	1.03	3.13	3.72				0.72	0.2	0.42	0.31	0.41	0.57				(<class 'ValueError'>, 216)
250	6	8.54	4.3	4.45	3.74	8.99	7.96				8.01	2.8	4.26	2.06	6.74	1.62				0.16	0.18	0.16	0.43	0.72	0.93				NoError
251	6	1.67	2.35	6.87	2.77	2.71	1.03				1	2.06	3.46	1.4	2.48	1.02				0.81	0.8	0.69	0.32	0.55	0.18				(<class 'ZeroDivisionError'>, 216)
252	6	2.07	7.17	5.88	8.72	3.94	6.69				1.55	2.09	2.59	2.31	3.87	5.67				0.32	0.71	0.69	0.84	0.11	0.08				(<class 'ValueError'>, 216)
253	6	3.24	8.39	9.23	5.43	3.07	7.88				1.82	8.33	4.32	2.8	1	4.23				0.91	0.99	0.54	0.09	0.43	0.99				(<class 'ValueError'>, 216)
254	6	3.33	3.82	5.71	6.73	8.3	1.4				3.24	1.76	2.49	4.57	6.84	1.17				0.63	0.26	0.87	0.18	0.42	0.76				(<class 'ValueError'>, 216)
255	6	5.1	9.84	8.1	4.74	2.43	4.21				3.73	9.29	1.29	3.16	1.23	1.32				0.32	0.84	0.12	0.59	0.38	0.67				(<class 'ValueError'>, 216)
256	6	8.16	5.07	7.63	9.74	3.12	5.84				7.54	1.77	2.65	5.65	1.83	3.53				0.05	0.77	0.21	0.9	0.38	0.34				NoError
257	6	6.16	5.17	7.17	1.76	6.52	2.97				5.85	4.76	6.91	1.09	2.44	2.8				0.48	0.12	0.67	0.23	0.51	0.62				NoError
258	6	6.13	3.16	9.17	6.11	5.29	8.98				3.37	2.15	6.62	1.95	4.49	2.77				0.6	0.85	0.33	0.23	0.84	0.76				(<class 'ValueError'>, 216)
259	6	9.54	2.05	4.13	2.94	9.51	5.49				7.36	2.03	1.73	1.59	5.97	3.82				0.9	0.48	0.91	0.47	1	0.38				NoError
260	6	6.02	1.86	1.87	2.61	8.09	9.22				3.45	1.42	1.76	1.36	7.62	9.05				0.77	0.4	0.02	0.87	0.9	0.37				(<class 'ZeroDivisionError'>, 162)
261	6	1.67	8.85	3.5	2.51	2.41	9.44				1.13	4.83	2.83	1.94	1.5	6.1				0.81	0.79	0.31	0.94	0.45	0.2				(<class 'ZeroDivisionError'>, 162)
262	6	3.39	4.18	4.37	5.53	2.21	4.8				1.25	3.77	2.84	3.79	1.59	3.83				0.4	0.13	0.4	0.68	0.58	0.59				NoError
263	6	6.06	2.33	2.37	2.55	7.64	4.27				2.15	1.4	2.13	2.31	7.26	4.11				0.05	0.64	0.97	0.67	0.56	0.8				(<class 'ZeroDivisionError'>, 162)
264	6	3.35	5.26	1.84	3.11	3.81	9.28				2	2.59	1.49	1.25	1.08	2.19				0.21	0.42	0.41	0.28	0.49	0.81				(<class 'ValueError'>, 216)
265	6	2.55	2.58	3.13	6.34	5.81	9.44				2.46	2.29	1.99	2.91	5	2.08				0.13	0.02	0.1	0.71	0.04	0.31				(<class 'ZeroDivisionError'>, 162)
266	6	9.85	5.42	6.11	5.71	1	5.81				7.22	1.83	2.14	2.97	1	3.99				0.14	0.06	0.61	0.28	0.38	0.34				NoError
267	6	1.85	8.39	6.21	4.79	8.08	4.78				1.64	7.26	2.67	4.62	1.55	3.87				0.84	0.03	0.87	0.37	0.28	0.88				NoError
268	6	8.23	2.3	2.88	5.1	6.45	8.5				2.9	1.54	1.58	3.41	2.46	7.86				0.74	0.59	0.71	0.85	0.02	0.24				(<class 'ZeroDivisionError'>, 162)
269	6	3.34	5.37	4.41	1.14	8.12	5.5				2.06	4.72	4.2	1.07	7.66	4.78				0.43	0.83	0.24	0.59	0.25	0.35				NoError
270	6	2.2	1.31	9.05	5.85	2.82	1.9				1.46	1.24	4.19	4.59	2.48	1.07				0.02	0.37	0.33	0.27	0.17	0.42				NoError
271	6	3.04	3.76	9.18	1.54	6.34	1.79				1.06	2.45	6.54	1.44	5.84	1.19				0.54	0.1	0.94	0.55	0.99	0.17				(<class 'ZeroDivisionError'>, 162)
272	6	2.42	4.64	3.21	3.44	7.59	4.54				1.66	4.55	1.63	1.34	5.84	2.24				0.32	0.42	0.06	0.38	0.69	0.76				NoError
273	6	6.86	9.52	8.54	1.18	8.11	2.11				3.74	3.3	6.68	1.09	2.31	1.61				0.99	0.16	0.75	0.44	0.72	0.57				(<class 'ZeroDivisionError'>, 162)
274	6	3.83	4.43	4.12	2.25	5.5	9.79				1.82	2.9	2.18	2.24	1.74	3.58				0.56	0.11	0.46	0.86	0.71	0.28				NoError
275	6	8.65	7.45	8.58	5.73	5.78	3.37				2.78	5.41	3.91	1.35	1.41	1.88				0.26	0.27	0.59	0.69	0.77	0.45				NoError
276	6	8.46	5.19	6.06	4.49	7.57	9.84				7.74	1.84	1.39	3.81	5.83	8.14				0.32	0.79	0.53	0.66	0.25	0.15				(<class 'ValueError'>, 216)
277	6	8.63	1.47	2.22	7.19	7.22	1.25				4.17	1.39	1.44	4.77	4.5	1.2				0.32	0.31	0.1	0.58	0.78	0.36				(<class 'ValueError'>, 216)
278	6	8.73	7.12	5.28	4.3	9.7	2.3				1.16	5.53	3.78	4.16	9.29	1.55				0.06	0.07	0.9	0.48	0.51	0.31				(<class 'ValueError'>, 216)
279	6	8.09	5.32	2.78	5.3	9.92	1.97				5.33	5.16	1.85	5.12	9.76	1.71				0.31	0.96	0.95	0.92	0.77	0.76				NoError
280	7	7.1	5.14	2.52	8.81	5.27	6.73	7.69			2.92	1.95	2.48	6.26	5.11	1.05	7.08			0.85	0.98	0.89	0.23	0.42	0.17	0.5			(<class 'ZeroDivisionError'>, 162)
281	7	3.6	6.21	7.25	3.56	6.09	9.75	3.28			1.66	2.94	4.89	1.31	4.83	5.3	2.26			0.7	0.57	0.58	0.25	0.8	0.73	0.67			(<class 'ValueError'>, 216)
282	7	1.89	5.38	3.87	1.85	3.62	5.65	1.29			1.51	1.59	2.11	1.32	1.14	4.76	1.19			0.62	0.13	0.73	0.98	0.99	0.79	0.34			NoError
283	7	2.7	1.21	4.8	7.26	3.44	5.89	9.97			1.75	1.1	4.32	2.44	2.05	1.87	7.14			0.26	0.68	0.98	0.66	0.45	0.85	0.66			(<class 'ZeroDivisionError'>, 162)
284	7	9.49	8	2.49	2.71	3.16	9.52	1.15			7.57	1.36	1.34	1.18	1.81	3.11	1.1			0.49	0.68	0.33	0.41	0.13	0.85	0.47			(<class 'ZeroDivisionError'>, 162)
285	7	4.17	9.39	8.24	5	8.76	7.2	2.53			1.24	1.13	7.16	1.65	4.43	2.11	2.24			0.55	0.77	0.49	0.74	0.44	0.29	0.7			(<class 'ZeroDivisionError'>, 162)
286	7	2.17	5.29	9.82	5.39	3.77	2.05	9.41			1.36	3.08	3.21	4.12	3.4	1.7	6.16			0.7	0.76	0.47	0.4	0.26	0.07	0.06			(<class 'ValueError'>, 216)
287	7	5.13	3.31	8.7	3.44	5.08	2.82	1.82			2.54	1.95	3.17	1.39	1.4	1.39	1.03			0.97	0.05	0.74	0.16	0.14	0.12	0.64			(<class 'ZeroDivisionError'>, 162)
288	7	8.7	2.3	9.81	4.7	3.51	9.86	5.19			1.48	1.47	2.14	1.63	3.16	6.69	1.23			0.53	0.92	0.95	0.34	0.61	0.42	0.27			(<class 'ZeroDivisionError'>, 162)
289	7	1.13	1.15	9.81	7.56	9.7	1.65	9.42			1.11	1	9.41	5.62	4.68	1.42	8.67			0.91	0.68	0.54	0.95	0.27	0.78	0.24			NoError
290	7	6.73	3.61	9.21	5.94	5.74	5.32	6.35			4.71	1.73	7.45	1.12	5.25	5.3	4.47			0.66	0.26	0.81	0.62	0.85	0.5	0.88			(<class 'ValueError'>, 216)
291	7	6.14	2.19	5.9	2.05	8.94	8.86	9.79			5.29	1.73	4.73	1.59	7.75	2.17	4.79			0.56	0.37	0.48	0.93	0.78	0.91	0.96			NoError
292	7	3.07	7.64	5.71	9.21	6.53	4.14	6.76			2.45	3.88	5.68	3.71	3.24	1.74	3.87			0.4	0.36	0.98	0.64	0.92	0.25	0.29			NoError
293	7	4.13	3.4	4.32	6.28	7.97	1.69	9.45			4.02	1.13	1.25	4.22	5.48	1.59	8.43			0.13	0.31	0.04	0.65	0.1	0.5	0.97			(<class 'ValueError'>, 216)
294	7	5.56	4.47	1.69	4.95	9.84	2.46	8.48			4.61	4.03	1.32	1.12	3.63	1.2	3.71			0.26	0.21	0.72	0.7	0.39	0.91	0.68			NoError
295	7	9.19	9.18	9.79	4.09	5	2.97	9.73			8.99	3.46	3.72	2.16	3.85	2.9	1.85			0.87	0.6	0.98	0.73	0.4	0.42	0.93			(<class 'ValueError'>, 216)
296	7	2.25	9.67	7.93	5.72	7.67	1.59	8.52			1.49	7.75	5.52	1.9	6.13	1.36	3.03			0.73	0.35	0.76	0.3	0.42	0.73	0.44			NoError
297	7	3.29	7.95	6.76	9.88	9.07	8.28	6.37			2.31	2.81	2.34	6.83	2.47	3.08	5.92			0.53	0.29	0.32	0.92	0.88	0.7	0.92			(<class 'ZeroDivisionError'>, 162)
298	7	2.64	6.86	5.25	9.17	5.77	9.17	4.9			2.54	4.85	4.44	3.96	1.52	4.81	1.99			0.35	0.12	0.01	0.43	0.57	0.8	0.91			NoError
299	7	2.47	1.59	4.9	2.67	7.89	7.38	4.06			1.06	1.34	3.78	2.48	7.7	4.71	2.54			0.1	0.67	0.02	0.72	0.24	0.23	0.92			NoError



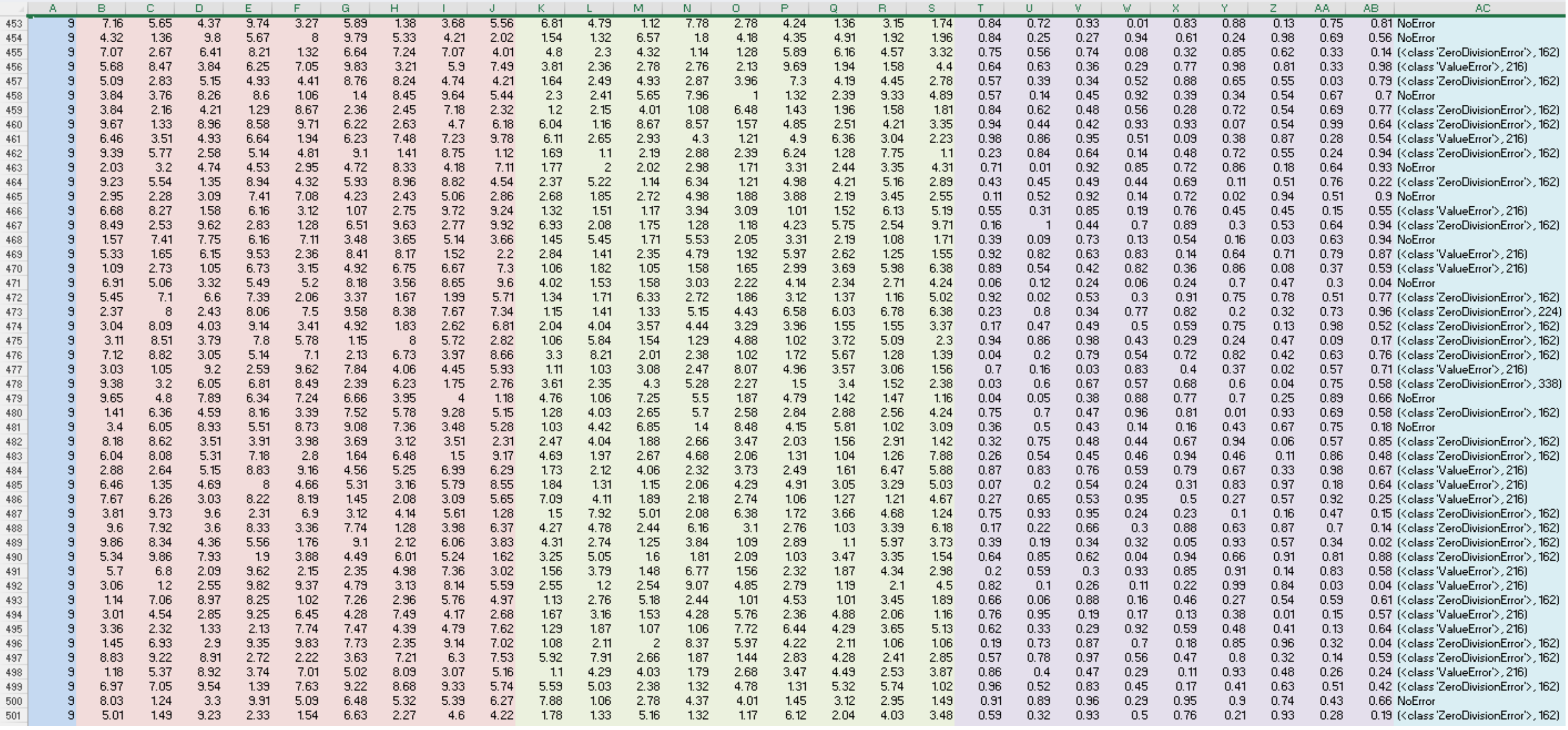
	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	AA	AB	AC
300	7	9.94	7.18	3.1	4.26	8.44	7.24	2.62			5.78	2.65	2.4	1.41	5.28	2.43	1.02			0.33	0.83	0.91	0.24	0.32	0.27	0.61			NoError
301	7	4.19	9.51	7.01	7.04	5.53	9.5	6.5			1.28	4.97	3.98	5.59	5.17	7.8	3.55			0.14	0.81	0.72	0.54	0.44	0.92	0.74			NoError
302	7	3.3	5.02	2.52	2.53	4.3	2.98	9.88			2.61	4.35	1.64	2.2	1.6	1.06	4.15			0.88	0.92	0.37	0.82	0.2	0.37	0.67			(<class 'ZeroDivisionError'>, 162)
303	7	8.65	6.12	3.28	3.93	3.63	2.91	2.61			1.95	1.8	2.61	3.78	1.79	2.46	2.48			0.11	0.67	0.96	0.9	0.49	0.57	0.56			NoError
304	7	9.14	6.88	2.35	3.29	7.38	5.65	8.86			2.27	3.54	1.23	2.74	4.93	4.91	2.83			0.93	0.17	0.99	0.1	0.25	0.44	0.76			(<class 'ZeroDivisionError'>, 162)
305	7	3.23	8.05	5.04	7.83	5.08	5.8	2.14			2.45	4.38	1.34	5.95	1.57	2.08	2.09			0.96	0.29	0.29	0.24	0.46	0.48	0.12			(<class 'ZeroDivisionError'>, 162)
306	7	9.94	4.57	9.68	7.08	7.31	5.53	7.92			6.98	3.35	2.29	5.16	1.09	5.03	6.64			0.22	0.32	0.91	0.41	0.09	0.71	0.3			(<class 'ZeroDivisionError'>, 162)
307	7	3.08	1.14	9.07	1.97	6.99	4.01	5.63			2.51	1.03	8.25	1.73	3.77	1.36	1.84			0.17	0.72	0.17	0.8	0.85	0.36	0.62			NoError
308	7	8.47	4.28	3.29	3.01	4.26	8.46	7.54			3.15	1.84	2.43	2.75	3.16	4.8	7.22			0.61	0.99	0.19	0.77	0.49	0.33	0.05			(<class 'ValueError'>, 216)
309	7	9.95	7.05	10	3.4	8.77	3.39	9.45			8.38	3.18	6.15	1.36	1.2	3.37	2.01			0.88	0.79	0.96	0.57	0.53	0.8	0.95			(<class 'ValueError'>, 216)
310	7	2.82	4.56	2.43	6.87	4.43	5.68	9.07			2.03	2.34	1.14	6.73	2.48	4.64	1.83			0.24	0.97	0.35	0.4	0.7	0.36	0.27			(<class 'ValueError'>, 216)
311	7	8.33	1.69	3.34	5.77	8.81	1.34	8.25			4.76	1.59	3.18	3.93	1.44	1.34	3.78			0.65	0.65	0.54	0.88	0.05	0.1	0.78			(<class 'ValueError'>, 216)
312	7	3.29	7.36	8.14	2.49	6.69	7.2	8.09			1.16	1.48	3.5	1.99	3.17	5.13	4.41			0.99	0.61	0.81	0.5	0.05	0.14	0.58			(<class 'ZeroDivisionError'>, 162)
313	7	8.65	9.65	2.37	3.52	6.75	3.52	8.89			4.63	8.52	1.88	3.5	2.03	1.61	2.84			0.13	0.67	0.06	0.21	0.18	0.74	0.74			NoError
314	7	4.6	3.34	4.05	1.05	1.85	8.4	6.33			2.94	2.74	2.18	1.03	1.37	2.74	4.87			0.59	0.21	0.51	0.74	0.4	0.74	0.9			NoError
315	7	4.67	6.57	9.5	5.43	7.68	3.21	3.27			3.45	4.45	7.14	2.43	4.79	1.97	2.08			0.23	0.71	0.21	0.66	0.76	0.14	0.03			(<class 'ZeroDivisionError'>, 224)
316	7	4.52	3.63	9.36	6.61	3.84	2.12	5.06			3.85	1.43	5.19	5.68	1.48	1.79	2.44			0.76	0.48	0.94	0.23	0.18	0.61	0.59			(<class 'ZeroDivisionError'>, 216)
317	7	3.53	6.95	3.73	3.35	8.22	5.14	9.28			1.67	2.13	2.32	1.82	2.35	1.33	3.15			0.39	0.07	0.34	0.94	0.99	0.93	0.07			(<class 'ZeroDivisionError'>, 162)
318	7	5.1	6.98	3.51	1.83	9.92	2.52	2.25			4.37	2.14	1.62	1.01	1.96	1.37	1.2			0.11	0.6	0.14	0.44	0.49	0.09	0.22			(<class 'ZeroDivisionError'>, 162)
319	7	6.28	8.18	5.34	7.45	9.83	4.49	3.5			6.22	4.84	2.63	1.56	3.19	1.01	2.28			0.65	0.99	1	0.24	0.53	0.68	0.56			NoError
320	7	2.91	5.58	3.9	4.75	7.02	8.54	4.11			1.96	1.12	3.47	2.09	6.51	4.4	3.31			0.81	0.64	0.64	0.32	0.4	0.36	0.78			(<class 'ZeroDivisionError'>, 162)
321	7	4.68	3.57	8.27	6.19	2.15	5.26	1.41			1.88	1.75	1.15	5.55	1.68	4.58	1.36			0.62	0.15	0.91	0.3	0.52	0.94	0.66			(<class 'ZeroDivisionError'>, 162)
322	7	1.57	3.46	1.23	9.88	7.85	8.09	6.45			1.31	3.29	1.07	4.73	6.46	3.05	3.46			0.28	0.75	0.53	0.17	0.41	0.82	0.9			NoError
323	7	5.5	6.76	1.71	7.95	2.79	6.24	6.91			2.14	2.72	1.48	7.61	1.68	6.23	3.28			0.89	0.4	0.11	0.47	0.99	0.83	0.3			(<class 'ZeroDivisionError'>, 162)
324	7	9.9	1.33	6.84	3.84	2.97	3.34	3.66			1.06	1.08	3.77	2.29	2.02	3.19	2.59			0.03	0.38	0.16	0.63	0.49	0.19	0.61			(<class 'ZeroDivisionError'>, 162)
325	7	6.56	2.25	1.36	5.98	4.5	3.98	2.59			1.06	2.07	1.03	4.61	3.12	3.64	1.31			0.48	0.61	0.33	0.25	0.89	0.01	0.95			NoError
326	7	4.99	1.26	2.57	4.05	2.05	4.47	3.82			2.59	1.08	2.21	3.84	2.03	3.83	2.16			0.85	0.49	0.99	0.36	0.08	0.35	0.95			(<class 'ZeroDivisionError'>, 162)
327	7	6.18	7.63	1.36	4.18	3.74	2.25	4.62			5.13	6.25	1.1	2.87	1.93	2.11	3.2			0.28	0.79	0.89	0.73	0.46	0.83	0.61			NoError
328	7	6.21	8.6	4.05	7.93	7.39	8.45	2.73			1.82	4.68	3.86	6.13	1.09	7.49	1.36			0.52	0.73	0.91	0.15	0.93	0.81	0.36			NoError
329	7	2.94	5.83	4.75	3.43	9	6.35	7.18			1.1	4.25	2.23	2.59	8.74	1.2	5.08			1	0.43	0.87	0.79	0.14	0.31	0.7			NoError
330	7	2.66	2.07	3.34	3.39	7.46	5.44	8.76			2.33	1.65	2.32	1.4	1.06	3.47	2.08			0.93	0.85	0.83	0.8	0.41	0.98	0.11			(<class 'ZeroDivisionError'>, 338)
331	7	6.4	9.1	4.42	4.23	1.4	1.5	9.05			2.14	7.31	1.53	3.66	1.13	1.28	8.34			0.32	0.04	0.52	0.77	0.44	0.62	0.9			NoError
332	7	9.61	3.42	8.9	9.94	9.29	5.45	8.7			5.49	3.14	5.23	2.56	9.21	3.15	2.64			0.11	0.34	0.79	0.4	0.76	0.1	0.16			(<class 'ZeroDivisionError'>, 162)
333	7	1.96	1.94	9.42	4.61	7.12	3.9	8.5			1.74	1.75	1.1	3.26	1.09	3.15	1.18			0.82	0.69	0.67	0.51	0.4	0.48	0.09			(<class 'ZeroDivisionError'>, 162)
334	7	6.53	4.02	4.27	9.57	9.13	9.46	2.9			2.47	3.59	4.13	2.36	1.34	8.82	1.05			0.46	0.31	0.72	0.1	0.92	0.89	0.23			NoError
335	7	6.01	9.65	5.17	1.29	8.53	9.75	5.4			2.62	1.86	4.89	1.25	3.23	8.9	2			0.86	0.47	0.52	0.03	0.77	0.81	0.81			NoError
336	7	9.62	3.27	6.62	6.35	7.9	3.1	4.75			7.02	2.1	1.4	2.27	6.49	1.38	1.13			0.84	0.61	0.52	0.27	0.45	0.3	0.52			(<class 'ZeroDivisionError'>, 162)
337	7	8.93	4.5	6.38	5.31	1.64	4.62	5.35			7.2	2.73	5.81	3.57	1.02	2	1.13			0.95	0.41	0.9	0.8	0.6	0.28	0.19			(<class 'ZeroDivisionError'>, 162)
338	7	6.88	6.61	3.12	9.91	1.39	9.34	8.25			5.9	4.19	1.89	4.42	1.31	3.16	2.74			0.86	0.81	0.16	0.13	0.54	0.91	0.39			(<class 'ZeroDivisionError'>, 162)
339	7	7.02	7.43	1.98	7.87	7.47	3.64	2.86			6.98	1.81	1.23	6.79	2.97	1.75	1.12			0.94	0.01	0.93	0.09	0.88	0.51	0.74			(<class 'ZeroDivisionError'>, 162)
340	7	6.12	3.41	1.83	1.71	2.28	3.35	6.76			1.52	1.54	1.7	1.54	1.99	2.49	4.03			0.2	0.47	0.79	0.42	0.97	0.27	0.62			NoError
341	7	6.75	9.38	1.24	4.71	8.86	3.29	6.29			1.37	3.17	1.15	2.25	6.33	2.47	5.84			0.07	0.45	0.08	0.67	0.93	0.16	0.02			(<class 'ZeroDivisionError'>, 216)
342	7	6.8	9.41	3.63	9.21	5.77	1.04	8.58			3.18	2.04	2.29	8.31	3.97	1.04	6.7			0.85	0.46	0.89	0.56	0.32	0.13	0.86			(<class 'ZeroDivisionError'>, 162)
343	7	4.23	5.62	8.78	5.04	5.52	2.81	1.58			3.78	4.67	7.33	3.5	5.1	1.04	1.38			0.48	0.57	0.78	0.45	0.86	0.07	0.76			(<class 'ValueError'>, 216)
344	7	1.71	3.12	4.02	3.15	6.21	4.39	2.39			1.69	2.49	2.74	1.14	3.67	3.26	2.32			0.58	0.48	0.98	0.4	0.41	0.04	0.65			(<class 'ZeroDivisionError'>, 162)
345	7	3.17	2.13	9.25	2.41	1.24	9.76	9.84			3.17	1.57	5.75	2.05	1.22	6.33	1.58			0.81	0.2	0.63	0.82	0.63	0.68	0.23			(<class 'ValueError'>, 216)
346	7	7.77	4.3	1.86	2.24	5.22	1.46	1.43			7.23	1.39	1.81	1.88	2.1	1.13	1.3			0.9	0.59	0.71	0.78	0.44	0.5	0.34			(<class 'ValueError'>, 216)
347	7	3.79	3.19	1.38	6.2	4.27	3.46	4.61			2.04	2.35	1.15	2.87	2.24	2.82	3.91			0.14	0.01	0.82	0.63	0.48	0.88	0.19			NoError
348	7	1.92	7.9	1.44	2.36	8.63	2.1	9.89			1.37	5.59	1.04	1.11	7.95	1.02	2.68			0.57	0.4	0.48	0.84	0.93	0.41	0.21			NoError
349	7	1.09	9.41	9.46	6.8	4.7	8.63	3.93			1.06	5.42	6.08	3.67	4.03	4.61	3.27			0.95	0.19	0.88	0.32	0.79	0.78	0.92			(<class 'ZeroDivisionError'>, 162)
350	7	8.26	3.01	3.35	3.17	5.25	8.84	6.92			3.76	2.05	1.7	1.16	2.06	1.35	3.73			0.03	0.5	0.3	0.22	0.12	0.68	0.79			NoError



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	AA	AB	AC
351	7	1.37	5.74	7.41	6.15	6.33	7.53	2.11			1.06	5.41	6.72	3.76	3.24	3.43	1.13			0.89	0.1	0.33	0.48	0.61	0.62	0.51			NoError
352	7	8.96	4.56	2.34	1.5	4.99	8.39	1.33			3.84	3.73	1.32	1.48	4.75	8.02	1.14			0.13	0.4	0.69	0.54	0.22	0.05	0.29			NoError
353	7	9.84	8.37	7.43	5.83	5.39	3.36	3.93			4.4	4.63	4.09	3.1	3.34	1.19	3.45			0.75	0.12	0.65	0.53	0.47	0.47	0.17			(<class 'ZeroDivisionError'>, 162)
354	7	5.02	3.39	5.29	6.99	2.03	1.44	4.08			4.64	1.28	2.5	2.46	1.9	1.14	3.52			0.45	0.68	0.37	0.45	0.92	0.68	0.22			(<class 'ZeroDivisionError'>, 162)
355	7	8.65	5.54	6.36	5.93	5.94	6.81	5.87			5.93	2.73	3.23	3.93	4.79	2.57	5.31			0.27	0.12	0.48	0.21	0.28	0.68	0.29			(<class 'ZeroDivisionError'>, 216)
356	7	7.74	3.28	5.9	4.49	3.99	9.26	9.34			3.32	2.91	4.89	1.24	3.3	6.63	4.76			0.29	0.15	0.85	0.74	0.12	0.04	0.03			NoError
357	7	8.69	4.88	9.69	2.04	8.42	5.78	5.93			8.3	3.41	8.29	1.06	1.69	1.01	3.71			0.48	0.51	0.25	0.03	0.3	0.38	0.38			(<class 'ValueError'>, 216)
358	7	2.07	5.62	1.48	9.74	8.76	6.53	1.54			2.02	5.07	1.05	6.09	3.71	2	1.17			0.25	0.66	0.92	0.7	0.2	0.64	0.58			(<class 'ZeroDivisionError'>, 224)
359	8	3.28	3.91	1.41	5.17	8.63	2.86	3.86	6.97		2.75	3.68	1.22	1.56	6.56	2.68	2.4	1.67		0.06	0.12	0.33	0.47	0.64	0.76	0.22	0.49		NoError
360	8	2.58	6.97	6.33	8.55	3.59	4.97	2.6	1.47		2.02	1.45	3.61	2.1	3.46	3.42	1.39	1.39		0.94	0.08	0.39	0.61	0.87	0.97	0.97	0.14		(<class 'ZeroDivisionError'>, 162)
361	8	2.31	2.52	8.21	6.35	7.25	8.23	6.29	5.75		1.93	2.37	7.03	3.04	4.12	7.81	3.21	3.34		0.95	0.79	0.12	0.77	0.54	0.8	0.54	0.25		NoError
362	8	9.17	7.79	3.19	7.57	5.42	6.49	4.32	7.55		6.31	7.26	1.01	3.87	4.06	3.72	2.35	6.22		0.6	0.39	0.93	0.98	0.73	0.74	0.95	0.04		NoError
363	8	8.72	7.7	4.2	5.19	1.78	6.82	3.97	5.64		5.17	7.35	1.03	5.1	1.34	5.22	3.69	3.51		0.9	0.17	0.12	0.61	0.93	0.01	0.86	0.81		(<class 'ValueError'>, 216)
364	8	3.32	8.87	1.04	7.55	1.94	8.95	9.64	8.86		1.69	5.9	1.02	3	1.04	7.04	2.97	6.63		0.02	0.6	0.18	0.64	0.82	0.5	0.16	0.19		(<class 'ZeroDivisionError'>, 162)
365	8	9.35	2.18	7.44	4.56	4.01	8.41	5.22	3.67		8.21	1.63	4.3	4.18	1.31	3.63	2.18	2.27		0.52	0.97	0.41	0.91	0.83	0.03	0.23	0.58		NoError
366	8	8.2	5.41	8.2	7.73	9.04	5.32	4.87	5.96		2.25	1.42	1.76	6.83	2.95	5.1	2.29	2.21		0.84	0.45	0.12	0.32	0.35	0.19	0.96	0.89		(<class 'ValueError'>, 216)
367	8	6.27	7.78	9.2	7.64	5.96	9.3	6.62	2.6		2.59	3.89	7.21	2.69	4.65	5.11	2.62	1		0.52	0.14	0.95	0.32	0.87	0.16	0.39	0.74		(<class 'ZeroDivisionError'>, 162)
368	8	9.79	9.2	8.15	9.75	5.78	1.05	2.95	4.37		3.87	3.7	2.22	8.1	1.15	1.05	2.77	3.29		0.18	0.4	0.67	0.39	0.88	0.62	0.4	0.56		(<class 'ZeroDivisionError'>, 216)
369	8	3.81	9.59	6.45	6.21	7.41	9.82	1.03	5.15		1.3	3.84	4.7	3.52	5.13	7.77	1.03	3.88		0.26	0.09	0.88	0.12	0.44	0.7	0.07	0.01		(<class 'ValueError'>, 216)
370	8	8.78	6.2	7.75	5.99	1.16	9.18	8.69	8.21		5.23	5.25	2.66	1.51	1.07	6.97	5.52	5.35		0.13	0.66	0.59	0.85	0.79	0.49	0.6	0.85		(<class 'ZeroDivisionError'>, 162)
371	8	6.53	5.9	7.64	2.62	6.05	2.2	7.22	5.16		2.15	3.25	2.51	2.59	1.33	1.13	1.02	3.07		0.92	0.17	0.92	0.31	0.61	0.77	0.52	0.56		(<class 'ZeroDivisionError'>, 162)
372	8	2.46	4.58	4.68	8.14	8.57	7.03	7.31	8.46		2.38	3.06	1.94	2.47	6.27	2.88	6.51	7.6		0.9	0.96	0.89	0.99	0.37	0.05	0.12	0.2		(<class 'ZeroDivisionError'>, 224)
373	8	9.01	1.68	2.86	2.15	3.11	7.73	4.84	5.08		8.38	1.57	2.05	1.59	1.59	3.83	1.07	2.85		0.31	0.8	0.25	0.9	0.81	0.31	0.61	0.91		NoError
374	8	4.5	7.67	5.85	3.3	8.43	7.33	8.24	4.69		2.25	5	2.33	1.02	6.4	6.01	5.35	4.18		0.37	0.78	0.28	0.63	0.55	0.32	0.46	0.29		(<class 'ValueError'>, 216)
375	8	3.13	1.18	1.4	4.11	7.6	3.81	1.78	3.06		2.17	1.01	1.38	1.44	6.97	1.9	1.46	1.1		0.52	0.55	0.27	0.84	0.28	0.6	0.1	0.53		(<class 'ZeroDivisionError'>, 162)
376	8	1.28	2.75	3.23	7.41	9.57	2.79	7.73	1.06		1.26	1.74	2.98	2.85	3.71	1.15	6.52	1.01		0.79	0.6	0.87	0.32	0.1	0.52	0.31	0.04		(<class 'ZeroDivisionError'>, 216)
377	8	5.51	3.74	9.63	5.99	2.63	6.36	1.51	9.53		2.73	2.17	1.52	1.88	2.15	4.23	1.3	6.86		0.05	0.62	0.18	0.77	0.26	0.73	0.24	0.5		(<class 'ValueError'>, 216)
378	8	8.27	6.71	1.83	9.22	8.4	2.28	9.29	6.49		8.26	5.14	1.33	8.43	3.83	1.83	5.35	2.72		0.02	0.4	0.84	0.87	0.75	0.27	0.67	0.26		(<class 'ZeroDivisionError'>, 162)
379	8	1.86	1.86	9.46	1.1	4.48	8.25	1.75	2.06		1.5	1.79	4.17	1.04	1.27	1.38	1.03	1.25		0.49	0.31	0.03	0.95	0.55	0.94	0.98	0.57		(<class 'ZeroDivisionError'>, 162)
380	8	2.01	6.19	7.24	1.36	4.44	8.44	7.35	6.49		1.3	2.07	6.8	1.1	1.26	7.6	4.75	4.98		0.41	0.95	0.99	0.9	0.27	0.09	0.48	0.92		(<class 'ValueError'>, 216)
381	8	4.2	1.47	5.95	8.81	6.39	7.27	2.06	7.81		3.78	1.42	4.01	5.09	2.26	5.37	1.15	5.84		0.11	0.44	0.54	0.07	0.76	0.09	0.62	0.35		NoError
382	8	8.57	5.11	9.97	2.26	7.29	7.68	4.52	9.49		1.53	3.95	4.99	2.02	3.65	3.63	3.98	5.69		0.93	0.9	0.06	0.93	0.33	0.84	0.74	0.41		NoError
383	8	9.63	3.95	8.78	7.79	1.64	4.31	7.85	2.18		6.17	3.31	4.44	3.92	1.32	1.15	3.39	1.1		0.32	0.55	0.57	0.6	0.1	0.13	0.22	0.18		(<class 'ZeroDivisionError'>, 162)
384	8	4.52	3.18	4.84	2.6	9.53	5.72	6.73	6.38		3.63	2.17	2.14	1.9	2.47	4.66	4.85	5.95		0.56	0.78	0.48	0.27	0.23	0.66	0.99	0.31		(<class 'ValueError'>, 216)
385	8	3.08	2.16	6.98	6.93	4.95	3.31	7.61	5.95		1.62	1.5	5.11	3.98	2.18	2.59	6.57	3.32		0.6	0.54	0.04	0.38	0.24	0.16	0.09	0.4		(<class 'ZeroDivisionError'>, 162)
386	8	9.02	7.51	4.65	8.64	7.97	1	4.83	1.4		2.51	6.99	1.05	3.55	5.01	1	1.43	1.38		0.03	0.52	0.73	0.94	0.39	0.37	0.36	0.79		(<class 'ZeroDivisionError'>, 162)
387	8	5.95	9.09	5.4	2.86	9.06	9.86	1.08	5.47		5.36	4.04	4.1	1.4	5.91	1.17	1.05	1.12		0.66	0.59	0.71	0.98	0.59	0.38	0.21	0.14		(<class 'ZeroDivisionError'>, 216)
388	8	9.32	1.08	8.21	3.5	4.91	9.62	4.8	5.2		3.03	1.06	3.84	3.35	4.59	3.55	2.87	1.92		0.13	0.22	0.68	0.38	0.26	0.58	0.41	0.62		(<class 'ZeroDivisionError'>, 162)
389	8	9.5	7.66	1.71	4.61	9.64	2.57	9.14	6.16		6.2	5.96	1.48	3.33	2.07	2.11	1.58	4.67		0.12	0.59	0.14	0.44	0.67	0.35	0.05	0.16		(<class 'ZeroDivisionError'>, 162)
390	8	2.02	6.07	1.09	3.39	4.91	1.4	4.61	5.48		1.61	3.29	1.04	1.14	3.1	1.2	4.55	1.51		0.37	0.65	0.37	0.51	0.09	0.58	0.13	0.34		(<class 'ZeroDivisionError'>, 162)
391	8	9.54	7.74	8.51	5.1	7.75	9.27	2.26	5.53		4.84	6.66	6.07	3.03	2.64	7.05	2.24	2.25		0.23	0.9	0.29	0.13	0.52	0.61	0.5	0.54		(<class 'ZeroDivisionError'>, 162)
392	8	5.32	9.46	7.72	9.88	3.59	2.22	6.1	3.89		1.86	9.31	6.15	1.15	2.03	1.58	5.54	3.67		0.09	0.52	0.84	0.44	0.25	0.72	0.37	0.86		(<class 'ValueError'>, 216)
393	8	6.61	1.44	2.64	6.66	1.62	2.67	3.94	3.93		2.66	1.18	1.93	1.26	1.13	1.96	2.4	1.45		0.58	0.33	0.52	0.77	0.62	0.15	0.04	0.64		(<class 'ZeroDivisionError'>, 162)
394	8	7.48	3.55	3.5	8.07	1.99	3	6.13	9.88		7.27	1.81	2.29	4.71	1.59	2.33	5.31	8.54		0.18	0.99	0.82	0.29	0.88	0.62	0.91	0.18		NoError
395	8	3.79	3.54	1.5	8.68	1.31	7.67	2.89	4.42		3.79	1.12	1.32	6.8	1.02	4.86	1.43	4.29		0.73	0.39	0.34	0.29	0.95	0.99	0.93	0.6		(<class 'ValueError'>, 216)
396	8	2.31	9.42	7.57	4.42	5.19	9.16	3.78	3.48		1.03	3.93	4.03	4.11	2.47	5.87	3.57	3.45		0.88	0.84	0.46	0.55	0.07	0.22	0.51	0.59		(<class 'ZeroDivisionError'>, 162)
397	8	5.58	3.54	1.13	6.39	5.4	6.37	6.47	5.46		5.35	2.13	1.08	4.39	2.91	5.94	2.86	5.11		0.3	0.46	0.57	0.02	0.2	0.94	0.98	0.51		NoError
398	8	1.65	9.1	2.15	1.43	5.06	1.61	7.19	6.24		1.55	2.2	1.83	1.18	4.39	1.26	4.06	2.49		0.03	0.23	0.19	0.16	0.89	0.16	0.56	0.47		(<class 'ZeroDivisionError'>, 162)
399	8	5.09	3.43	4.69	7.89	6.67	2.62	9.54	7.31		4.86	2.57	4.35	2.86	3.63	1.73	4.72	3.38		0.67	0.71	0.95	0.13	0.61	0.34	0.52	0.3		NoError
400	8	9.92	3.49	6.72	7.15	5.54	1.28	9.45	2.67		2.98	2.7	4.34	3.59	1.15	1.19	6.43	2.11		0.19	0.15	0.35	0.05	0.81	0.51	0.3	0.81		(<class 'ZeroDivisionError'>, 162)
401	8	6.82	5.65	5.42	5.07	5.95	9.4	2.83	6.07		4.9	3.7	3.85	4.78	3.56	9.16	1.53	4.19		0.32	0.67	0.56	0.26	0.06	0.35	0.41	0.75		NoError



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	AA	AB	AC
402	8	7.35	2.26	8.67	3.3	7.65	8.31	7.64	3.06		3.66	1.38	3.92	2.85	6.3	6.36	2.77	1.85		0.72	0.63	0.34	0.12	0.9	0.2	0.72	0.12		(<class 'ValueError'>, 216)
403	8	6.01	3.37	3.53	5.47	4.61	5.36	9.56	4.19		2.2	1.97	2.69	4.14	2.22	3.79	6.92	2.75		0.09	0.35	0.31	0.71	0.56	0.88	0.3	0.15		(<class 'ZeroDivisionError'>, 162)
404	8	7.65	5.44	6.09	2.18	7.01	1.57	1.5	3.07		1.28	4.75	3.14	1.39	1.4	1.21	1.19	2.86		0.17	0.56	0.46	0.43	0.41	0.97	0.82	0.77		NoError
405	8	1.77	3.14	1.08	5.83	3.12	9.81	7.02	6.34		1.72	1.17	1.07	5.15	2.84	6.8	1.02	1.37		0.98	0.95	0.6	0.79	0.56	0.52	0.24	0.52		(<class 'ValueError'>, 216)
406	8	4.6	4.63	2.73	3.82	2.24	4.34	9.33	6.97		3.17	4.26	2.44	3.8	1.04	3.67	5.22	1.33		0.6	0.05	0.15	0.25	0.84	0.69	0.05	0.74		(<class 'ValueError'>, 216)
407	8	2.29	8.84	6.36	5.74	3.85	2.7	3.18	5.05		1.04	3.24	2.12	4.79	1.51	2.42	2.42	1		0.6	0.74	0.56	0.32	0.49	0.69	0.4	0.81		(<class 'ValueError'>, 216)
408	8	3.11	2.33	6.71	3.36	7	5.32	4.56	7.36		1.81	1.06	5.58	1.62	6.28	2.19	1.85	2.43		0.29	0.98	0.06	0.62	0.89	0.44	0.05	0.16		(<class 'ZeroDivisionError'>, 162)
409	8	9.83	8.13	4.03	2.27	9.82	9.91	1.93	8.07		5.49	2.79	2.05	2.08	1.65	6.37	1.7	2.25		0.48	0.68	0.99	0.29	0.62	0.54	0.91	0.43		(<class 'ZeroDivisionError'>, 162)
410	8	1.61	8.77	5.82	3.59	8.32	2.38	4.88	5.42		1.5	2.42	3.5	1.7	5.79	1.19	4.73	1.84		0.98	0.56	0.22	0.17	0.5	0.43	0.23	0.42		(<class 'ZeroDivisionError'>, 162)
411	8	9.14	2.59	2.74	4.4	7.29	4.71	8.28	1.81		4.62	2.23	1.58	1.15	1.53	4.05	4.83	1.33		0.55	0.12	0.09	0.98	0.94	0.98	0.1	0.19		(<class 'ZeroDivisionError'>, 162)
412	8	6.14	1.35	4.51	5.84	2.47	9.49	1.13	8.94		1.41	1.22	2.41	2.23	1.68	8.74	1.02	8.12		1	0.77	0.97	0.57	0.9	0.56	0.32	0.11		(<class 'ZeroDivisionError'>, 162)
413	8	1.46	3.58	3.84	5.09	1.78	2.64	9.31	8.22		1.07	2.53	2.24	4.97	1.31	2.11	2.56	5.3		0.44	0.06	0.04	0.59	0.06	0.8	0.15	0.73		(<class 'ZeroDivisionError'>, 391)
414	8	8.05	8.27	9.37	3.66	1.31	7.35	9.25	8.68		7.03	4.33	2.69	3	1.21	3.39	5.48	4.03		0.79	0.01	0.07	0.75	0.79	0.4	0.41	0.46		(<class 'ZeroDivisionError'>, 162)
415	8	2.67	5.2	4.75	6.54	9.19	1.06	5.79	9.89		1.68	2.13	4.18	4.07	2.76	1	3.5	2.12		0.84	0.65	0.94	0.03	0.07	0.09	0.07	0.15		(<class 'ZeroDivisionError'>, 162)
416	8	1.48	2.21	7.37	6.4	5.36	7.18	7.3	1.51		1.21	1.62	6.28	1.89	3.41	6	1.2	1.05		0.27	0.73	0.74	0.69	0.41	0.29	0.7	0.7		(<class 'ValueError'>, 216)
417	8	2.69	8.04	9	9.69	4.86	3.85	9.74	3.4		1.47	4.24	3.93	5.22	2.85	2.29	1.07	2.08		0.77	0.94	0.04	0.8	0.68	0.02	0.72	0.27		(<class 'ZeroDivisionError'>, 162)
418	8	8.6	3.39	4.94	7.74	4.83	1.76	9.54	6.97		6.04	3.26	3.09	4.52	4.32	1.65	3.13	5.17		0.18	0.99	0.12	0.31	0.57	0.61	0.88	0.49		NoError
419	8	7.89	5.99	7.89	8.36	9.66	3	1.88	4.78		3.04	3.93	7.14	5.78	6.7	2.59	1.61	1.26		0.43	0.03	0.88	0.79	0.67	0.1	0.88	0.29		(<class 'ZeroDivisionError'>, 162)
420	8	1.9	9.59	7.37	7.48	3.14	5.13	3.4	9.38		1.53	2.63	6.75	4.12	2.66	4.63	2.78	3.32		0.15	0.99	0.78	0.99	0.08	0.79	0.69	0.91		(<class 'ZeroDivisionError'>, 162)
421	8	9.22	4.43	5.64	4	2.04	1.29	7.34	8.87		2.52	3.17	3.02	3.01	1.55	1.13	7.07	7.08		0.74	0.35	0.87	0.53	0.94	0.18	0.3	0.98		(<class 'ValueError'>, 216)
422	8	5.51	1.7	6.13	1.72	5.21	5.34	2.25	1.01		2.16	1.6	4.42	1.5	3.14	3.62	1.52	1.01		0.8	0.18	0.46	0.49	0.41	0.48	0.49	0.88		(<class 'ZeroDivisionError'>, 162)
423	8	7.79	6.12	9.48	2.15	1.85	9.37	4.79	7.55		1.85	3.04	1.92	1.77	1.23	3.27	4.73	2.79		0.41	0.81	0.72	0.49	0.35	0.4	0.81	0.66		(<class 'ZeroDivisionError'>, 162)
424	8	8.69	5.01	6.71	6.83	8.64	9.55	6.58	6.74		8.27	2.95	5.94	2.15	4.79	4.67	5.94	4.29		0.47	0.57	0.11	0.22	0.34	0.73	0.48	0.01		(<class 'ValueError'>, 216)
425	8	7.73	4.33	2.27	4.28	4.32	4.59	8.5	6.26		2.08	4.16	1.37	3.79	2.24	1.36	1.73	3.23		0.88	0.66	0.98	0.76	0.9	0.88	0.23	0.74		(<class 'ValueError'>, 216)
426	8	6.32	1.11	5.92	4.74	3.66	5.11	8.83	6.82		4.26	1.04	4.7	3.06	2.45	3.81	4.2	5.53		0.79	0.94	0.37	0.37	0.71	0.88	0.88	0.33		NoError
427	8	8.76	2.81	6.7	4.27	4.53	5.92	6.8	3.14		5.72	1.82	5.36	1.72	1.54	2.71	1.86	2.27		0.15	0.51	0.04	0.92	0.27	0.56	0.64	0.95		NoError
428	8	5.89	5.81	5.93	7.02	4.82	2.83	9.86	4.8		2.57	4.86	3.05	2.5	2.96	1.85	1.18	1.67		0.27	0.02	0.93	0.5	0.59	0.42	0.44	0.52		NoError
429	8	6.57	7.84	9.93	5.72	1.18	9.29	8.65	9.89		1.86	6.6	3.05	2.56	1.06	7.49	5.8	8.49		0.52	0.15	0.32	0.92	0.71	0.51	0.77	0.16		(<class 'ZeroDivisionError'>, 162)
430	8	5.81	9.05	7.88	2.72	3.51	7.13	2.48	7.64		1.92	3.49	4.95	1.05	1.75	5.73	1.08	1.3		0.93	0.57	0.57	0.66	0.7	0.49	0.14	0.54		(<class 'ZeroDivisionError'>, 162)
431	8	7.15	4.9	7.31	1.94	5.59	3.28	4.34	4.47		2.58	1.05	5.97	1.65	1.75	3.17	3.39	2.62		0.96	0.96	0.49	0.94	0.72	0.72	0.12	0.05		(<class 'ZeroDivisionError'>, 162)
432	8	1.45	9.45	5.09	3.48	4.79	1.74	9.35	2.12		1.05	4.23	4.54	1.47	3.27	1.59	7.1	1.14		0.16	0.24	0.59	0.56	0.12	0.23	0.16	0.09		NoError
433	9	4.26	2.39	7.65	8.14	9.83	2.9	6.24	8.1	5.44	2.64	1.88	1.92	1.14	5.35	2.74	5.68	6.2	2.68	0.97	0.04	0.08	0.3	0.14	0.58	0.74	0.42	0.49	(<class 'ZeroDivisionError'>, 162)
434	9	5.09	1.21	7.18	7.65	2.92	9.77	7.84	4.2	4.04	1.19	1.05	5.65	7.54	1.7	7.72	6.68	1.1	2.01	0.36	0.9	0.89	0.66	0.49	0.94	0.18	0.72	0.2	(<class 'ZeroDivisionError'>, 162)
435	9	2.22	8.66	4.67	4.53	7.96	1.2	5.41	8.08	2.35	1.8	2.99	1.12	4.49	7.87	1.15	5.3	6.33	2.06	0.58	0.01	0.37	0.49	0.59	0.77	0.51	0.56	0.08	(<class 'ZeroDivisionError'>, 162)
436	9	9.63	5.69	8.82	3.79	6.01	5.74	6.28	9.3	1.15	1.18	1.53	8.02	2.81	4.35	4.92	3.67	2.23	1.03	0.83	0.17	0.61	0.74	0.15	0.55	0.99	0.54	0.77	(<class 'ZeroDivisionError'>, 162)
437	9	6.19	2.78	3.49	5.86	8.98	3.9	3.24	8.5	4.33	4.68	1.79	1.72	2.05	3.54	1.07	1.61	5.68	3.34	0.43	0.11	0.91	0.91	0.7	0.97	0.46	0.19	0.79	NoError
438	9	3.11	7.63	7.85	4.46	8.1	4.85	1.14	2.29	9.5	1.77	1.53	6.7	3.47	2.9	2.57	1.02	2.08	9.19	0.06	0.63	0.84	0.93	0.07	0.53	0.06	0.55	0.14	(<class 'ValueError'>, 216)
439	9	6.72	6.54	1.19	7.29	6.31	8.32	9.2	8.57	8.3	5.53	4.76	1.02	2.44	3.44	6.91	6.46	6.27	8	0.22	0.82	0.77	0.08	0.4	0.91	0.83	0.43	0.81	(<class 'ValueError'>, 216)
440	9	6.91	3.54	2.44	8.71	1.17	8.31	6.55	9.64	3.35	6.29	1.22	1.03	1.18	1.04	6.97	3.09	9.3	2.56	0.4	0.31	0.49	0.39	0.44	0.28	0.83	0.09	0.62	(<class 'ZeroDivisionError'>, 162)
441	9	3.43	2.52	4.98	3.15	4.43	9.99	6.05	3.27	8.9	2.23	1.52	1.69	1.23	2.04	6.69	5.67	2.82	1.88	0.04	0.83	0.12	0.46	0.49	0.17	0.24	0.03	0.74	(<class 'ZeroDivisionError'>, 162)
442	9	1.52	7.46	9.05	6.82	7.91	8.59	9.65	1.03	9.31	1.18	2.66	2.96	1.79	7.15	5.56	2.03	1.02	5.56	0.26	0.92	0.23	0.65	0.04	0.96	0.71	0.22	0.02	(<class 'ZeroDivisionError'>, 162)
443	9	9.71	6.56	8.61	2.77	1.41	2.41	6.36	5.63	6.75	7.18	2.56	1.14	1.72	1.2	2.27	4.81	5.59	5.19	0.24	0.66	0.53	0.77	0.56	0.04	0.04	0.49	0.46	(<class 'ZeroDivisionError'>, 338)
444	9	8.37	4.86	7.45	3.46	3.55	9.87	6.13	1.96	1.68	6.12	2.76	5.68	2.49	1.09	6.01	1.57	1.49	1.13	0.75	0.98	0.23	0.94	0.57	0.53	0.99	0.66	0.26	(<class 'ZeroDivisionError'>, 162)
445	9	6.94	5.75	4.2	2.23	8.41	4.07	6.25	7.36	7.73	1.32	3.55	2.84	2.12	7.96	2.63	4.81	2.66	1.67	0.7	0.76	0.81	0.98	0.88	0.43	0.66	0.93	0.59	(<class 'ValueError'>, 216)
446	9	5.93	3.7	8.31	7	3.1	6.61	6.52	5.25	9.41	5.49	2.36	1.15	6.2	1.25	2.25	3.64	5.16	7.8	0.49	0.52	0.22	0.35	0.65	0.6	0.2	0.06	0.72	(<class 'ZeroDivisionError'>, 162)
447	9	9.04	3.29	8.13	3.28	6.56	7.32	1.04	1.38	7.73	2.21	1.75	6.74	2.19	2.39	1.37	1.02	1.29	1.58	0.78	0.74	0.56	0.67	0.51	0.85	0.24	0.43	0.3	(<class 'ZeroDivisionError'>, 162)
448	9	5.79	1.59	6.64	1.27	7.55	3.66	4.85	3.67	5.79	4.85	1.39	4	1.17	7.12	2.89	2	1.29	4.91	0.08	0.48	0.17	0.68	0.48	0.64	0.84	0.46	0.09	NoError
449	9	7.36	9.65	7.75	9.54	3.57	1.14	4.83	4.88	9.05	7.14	4.89	2.14	4.54	2.54	1.03	3.68	1.89	4.12	0.33	0.6	0.52	0.84	0.21	0.5	0.61	0.05	0.56	(<class 'ZeroDivisionError'>, 162)
450	9	7.36	9.44	2.03	5.73	3.47	1.42	3.01	4.93	7.78	3.96	1.84	1.85	1.63	1	1.29	2.01	1.7	5.18	0.47	0.72	0.34	0.58	0.84	0.24	0.95	0.07	0.87	NoError
451	9	4.61	2.7	2.22	3.64	2.27	4.56	7.76	9.48	3.58	3.24	2.68	1.73	2.87	1.61	2.42	6.43	4.24	2.76	0.92	0.29	0.69	0.17	0.58	0.36	0.82	0.76	0.57	(<class 'ValueError'>, 216)
452	9	5.45	4.98	5.49	4.99	6.38	3.12	3.7	7.45	7.85	1.72	2.62	5.26	1.72	1.56	3.06	3.23	1.02	1.28	0.96	0.03	0.95	0.52	0.99	0.86	0.92	0.26	0.47	(<class 'ZeroDivisionError'>, 162)



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	AA	AB	AC
453	9	7.16	5.65	4.37	9.74	3.27	5.89	1.38	3.68	5.56	6.81	4.79	1.12	7.78	2.78	4.24	1.36	3.15	1.74	0.84	0.72	0.93	0.01	0.83	0.88	0.13	0.75	0.81	NoError
454	9	4.32	1.36	9.8	5.67	8	9.79	5.33	4.21	2.02	1.54	1.32	6.57	1.8	4.18	4.35	4.91	1.92	1.96	0.84	0.25	0.27	0.94	0.61	0.24	0.98	0.69	0.56	NoError
455	9	7.07	2.67	6.41	8.21	1.32	6.64	7.24	7.07	4.01	4.8	2.3	4.32	1.14	1.28	5.89	6.16	4.57	3.32	0.75	0.56	0.74	0.08	0.32	0.85	0.62	0.33	0.14	(<class 'ZeroDivisionError'>, 162)
456	9	5.68	8.47	3.84	6.25	7.05	9.83	3.21	5.9	7.49	3.81	2.36	2.78	2.76	2.13	9.69	1.94	1.58	4.4	0.64	0.63	0.36	0.29	0.77	0.98	0.81	0.33	0.98	(<class 'ValueError'>, 216)
457	9	5.09	2.83	5.15	4.93	4.41	8.76	8.24	4.74	4.21	1.64	2.49	4.93	2.87	3.96	7.3	4.19	4.45	2.78	0.57	0.39	0.34	0.52	0.88	0.65	0.55	0.03	0.79	(<class 'ZeroDivisionError'>, 162)
458	9	3.84	3.76	8.26	8.6	1.06	1.4	8.45	9.64	5.44	2.3	2.41	5.65	7.96	1	1.32	2.39	9.33	4.89	0.57	0.14	0.45	0.92	0.39	0.34	0.54	0.67	0.7	NoError
459	9	3.84	2.16	4.21	1.29	8.67	2.36	2.45	7.18	2.32	1.2	2.15	4.01	1.08	6.48	1.43	1.96	1.58	1.81	0.84	0.62	0.48	0.56	0.28	0.72	0.54	0.69	0.77	(<class 'ZeroDivisionError'>, 162)
460	9	9.67	1.33	8.96	8.58	9.71	6.22	2.63	4.7	6.18	6.04	1.16	8.67	8.57	1.57	4.85	2.51	4.21	3.35	0.94	0.44	0.42	0.93	0.93	0.07	0.54	0.99	0.64	(<class 'ZeroDivisionError'>, 162)
461	9	6.46	3.51	4.93	6.64	1.94	6.23	7.48	7.23	9.78	6.11	2.65	2.93	4.3	1.21	4.9	6.36	3.04	2.23	0.98	0.86	0.95	0.51	0.09	0.38	0.87	0.28	0.54	(<class 'ValueError'>, 216)
462	9	9.39	5.77	2.58	5.14	4.81	9.1	1.41	8.75	1.12	1.69	1.1	2.19	2.88	2.39	6.24	1.28	7.75	1.1	0.23	0.84	0.64	0.14	0.48	0.72	0.55	0.24	0.94	(<class 'ZeroDivisionError'>, 162)
463	9	2.03	3.2	4.74	4.53	2.95	4.72	8.33	4.18	7.11	1.77	2	2.02	2.98	1.71	3.31	2.44	3.35	4.31	0.71	0.01	0.92	0.85	0.72	0.86	0.18	0.64	0.93	NoError
464	9	9.23	5.54	1.35	8.94	4.32	5.93	8.96	8.82	4.54	2.37	5.22	1.14	6.34	1.21	4.98	4.21	5.16	2.89	0.43	0.45	0.49	0.44	0.69	0.11	0.51	0.76	0.22	(<class 'ZeroDivisionError'>, 162)
465	9	2.95	2.28	3.09	7.41	7.08	4.23	2.43	5.06	2.86	2.68	1.85	2.72	4.98	1.88	3.88	2.19	3.45	2.55	0.11	0.52	0.92	0.14	0.72	0.02	0.94	0.51	0.9	NoError
466	9	6.68	8.27	1.58	6.16	3.12	1.07	2.75	9.72	9.24	1.32	1.51	1.17	3.94	3.09	1.01	1.52	6.13	5.19	0.55	0.31	0.85	0.19	0.76	0.45	0.45	0.15	0.55	(<class 'ValueError'>, 216)
467	9	8.49	2.53	9.62	2.83	1.28	6.51	9.63	2.77	9.92	6.93	2.08	1.75	1.28	1.18	4.23	5.75	2.54	9.71	0.16	1	0.44	0.7	0.89	0.3	0.53	0.64	0.94	(<class 'ZeroDivisionError'>, 162)
468	9	1.57	7.41	7.75	6.16	7.11	3.48	3.65	5.14	3.66	1.45	5.45	1.71	5.53	2.05	3.31	2.19	1.08	1.71	0.39	0.09	0.73	0.13	0.54	0.16	0.03	0.63	0.94	NoError
469	9	5.33	1.65	6.15	9.53	2.36	8.41	8.17	1.52	2.2	2.84	1.41	2.35	4.79	1.92	5.97	2.62	1.25	1.55	0.92	0.82	0.63	0.83	0.14	0.64	0.71	0.79	0.87	(<class 'ValueError'>, 216)
470	9	1.09	2.73	1.05	6.73	3.15	4.92	6.75	6.67	7.3	1.06	1.82	1.05	1.58	1.65	2.99	3.69	5.98	6.38	0.89	0.54	0.42	0.82	0.36	0.86	0.08	0.37	0.59	(<class 'ValueError'>, 216)
471	9	6.91	5.06	3.32	5.49	5.2	8.18	3.56	8.65	9.6	4.02	1.53	1.58	3.03	2.22	4.14	2.34	2.71	4.24	0.06	0.12	0.24	0.06	0.24	0.7	0.47	0.3	0.04	NoError
472	9	5.45	7.1	6.6	7.39	2.06	3.37	1.67	1.99	5.71	1.34	1.71	6.33	2.72	1.86	3.12	1.37	1.16	5.02	0.92	0.02	0.53	0.3	0.91	0.75	0.78	0.51	0.77	(<class 'ZeroDivisionError'>, 162)
473	9	2.37	8	2.43	8.06	7.5	9.58	8.38	7.67	7.34	1.15	1.41	1.33	5.15	4.43	6.58	6.03	6.78	6.38	0.23	0.8	0.34	0.77	0.82	0.2	0.32	0.73	0.96	(<class 'ZeroDivisionError'>, 224)
474	9	3.04	8.09	4.03	9.14	3.41	4.92	1.83	2.62	6.81	2.04	4.04	3.57	4.44	3.29	3.96	1.55	1.55	3.37	0.17	0.47	0.49	0.5	0.59	0.75	0.13	0.98	0.52	(<class 'ZeroDivisionError'>, 162)
475	9	3.11	8.51	3.79	7.8	5.78	1.15	8	5.72	2.82	1.06	5.84	1.54	1.29	4.88	1.02	3.72	5.09	2.3	0.94	0.86	0.98	0.43	0.29	0.24	0.47	0.09	0.17	(<class 'ZeroDivisionError'>, 162)
476	9	7.12	8.82	3.05	5.14	7.1	2.13	6.73	3.97	8.66	3.3	8.21	2.01	2.38	1.02	1.72	5.67	1.28	1.39	0.04	0.2	0.79	0.54	0.72	0.82	0.42	0.63	0.76	(<class 'ZeroDivisionError'>, 162)
477	9	3.03	1.05	9.2	2.59	9.62	7.84	4.06	4.45	5.93	1.11	1.03	3.08	2.47	8.07	4.96	3.57	3.06	1.56	0.7	0.16	0.03	0.83	0.4	0.37	0.02	0.57	0.71	(<class 'ValueError'>, 216)
478	9	9.38	3.2	6.05	6.81	8.49	2.39	6.23	1.75	2.76	3.61	2.35	4.3	5.28	2.27	1.5	3.4	1.52	2.38	0.03	0.6	0.67	0.57	0.68	0.6	0.04	0.75	0.58	(<class 'ZeroDivisionError'>, 338)
479	9	9.65	4.8	7.89	6.34	7.24	6.66	3.95	4	1.18	4.76	1.06	7.25	5.5	1.87	4.79	1.42	1.47	1.16	0.04	0.05	0.38	0.88	0.77	0.7	0.25	0.89	0.66	NoError
480	9	1.41	6.36	4.59	8.16	3.39	7.52	5.78	9.28	5.15	1.28	4.03	2.65	5.7	2.58	2.84	2.88	2.56	4.24	0.75	0.7	0.47	0.96	0.81	0.01	0.93	0.69	0.58	(<class 'ZeroDivisionError'>, 162)
481	9	3.4	6.05	8.93	5.51	8.73	9.08	7.36	3.48	5.28	1.03	4.42	6.85	1.4	8.48	4.15	5.81	1.02	3.09	0.36	0.5	0.43	0.14	0.16	0.43	0.67	0.75	0.18	NoError
482	9	8.18	8.62	3.51	3.91	3.98	3.69	3.12	3.51	2.31	2.47	4.04	1.88	2.66	3.47	2.03	1.56	2.91	1.42	0.32	0.75	0.48	0.44	0.67	0.94	0.06	0.57	0.85	(<class 'ZeroDivisionError'>, 162)
483	9	6.04	8.08	5.31	7.18	2.8	1.64	6.48	1.5	9.17	4.69	1.97	2.67	4.68	2.06	1.31	1.04	1.26	7.88	0.26	0.54	0.45	0.46	0.94	0.46	0.11	0.86	0.48	(<class 'ZeroDivisionError'>, 162)
484	9	2.88	2.64	5.15	8.83	9.16	4.56	5.25	6.99	6.29	1.73	2.12	4.06	2.32	3.73	2.49	1.61	6.47	5.88	0.87	0.83	0.76	0.59	0.79	0.67	0.33	0.98	0.67	(<class 'ValueError'>, 216)
485	9	6.46	1.35	4.69	8	4.66	5.31	3.16	5.79	8.55	1.84	1.31	1.15	2.06	4.29	4.91	3.05	3.29	5.03	0.07	0.2	0.54	0.24	0.31	0.83	0.97	0.18	0.64	(<class 'ValueError'>, 216)
486	9	7.67	6.26	3.03	8.22	8.19	1.45	2.08	3.09	5.65	7.09	4.11	1.89	2.18	2.74	1.06	1.27	1.21	4.67	0.27	0.65	0.53	0.95	0.5	0.27	0.57	0.92	0.25	(<class 'ValueError'>, 216)
487	9	3.81	9.73	9.6	2.31	6.9	3.12	4.14	5.61	1.28	1.5	7.92	5.01	2.08	6.38	1.72	3.66	4.68	1.24	0.75	0.93	0.95	0.24	0.23	0.1	0.16	0.47	0.15	(<class 'ZeroDivisionError'>, 162)
488	9	9.6	7.92	3.6	8.33	3.36	7.74	1.28	3.98	6.37	4.27	4.78	2.44	6.16	3.1	2.76	1.03	3.39	6.18	0.17	0.22	0.66	0.3	0.88	0.63	0.87	0.7	0.14	(<class 'ZeroDivisionError'>, 162)
489	9	9.86	8.34	4.36	5.56	1.76	9.1	2.12	6.06	3.83	4.31	2.74	1.25	3.84	1.09	2.89	1.1	5.97	3.73	0.39	0.19	0.34	0.32	0.05	0.93	0.57	0.34	0.02	(<class 'ZeroDivisionError'>, 162)
490	9	5.34	9.86	7.93	1.9	3.88	4.49	6.01	5.24	1.62	3.25	5.05	1.6	1.81	2.09	1.03	3.47	3.35	1.54	0.64	0.85	0.62	0.04	0.94	0.66	0.91	0.81	0.88	(<class 'ZeroDivisionError'>, 162)
491	9	5.7	6.8	2.09	9.62	2.15	2.35	4.98	7.36	3.02	1.56	3.79	1.48	6.77	1.56	2.32	1.87	4.34	2.98	0.2	0.59	0.3	0.93	0.85	0.91	0.14	0.83	0.58	(<class 'ValueError'>, 216)
492	9	3.06	1.2	2.55	9.82	9.37	4.79	3.13	8.14	5.59	2.55	1.2	2.54	9.07	4.85	2.79	1.19	2.1	4.5	0.82	0.1	0.26	0.11	0.22	0.99	0.84	0.03	0.04	(<class 'ValueError'>, 216)
493	9	1.14	7.06	8.97	8.25	1.02	7.26	2.96	5.76	4.97	1.13	2.76	5.18	2.44	1.01	4.53	1.01	3.45	1.89	0.66	0.06	0.88	0.16	0.46	0.27	0.54	0.59	0.61	(<class 'ZeroDivisionError'>, 162)
494	9	3.01	4.54	2.85	9.25	6.45	4.28	7.49	4.17	2.68	1.67	3.16	1.53	4.28	5.76	2.36	4.88	2.06	1.16	0.76	0.95	0.19	0.17	0.13	0.38	0.01	0.15	0.57	(<class 'ValueError'>, 216)
495	9	3.36	2.32	1.33	2.13	7.74	7.47	4.39	4.79	7.62	1.29	1.87	1.07	1.06	7.72	6.44	4.29	3.65	5.13	0.62	0.33	0.29	0.92	0.59	0.48	0.41	0.13	0.64	(<class 'ValueError'>, 216)
496	9	1.45	6.93	2.9	9.35	9.83	7.73	2.35	9.14	7.02	1.08	2.11	2	8.37	5.97	4.22	2.11	1.06	1.06	0.19	0.73	0.87	0.7	0.18	0.85	0.96	0.32	0.04	(<class 'ZeroDivisionError'>, 162)
497	9	8.83	9.22	8.91	2.72	2.22	3.63	7.21	6.3	7.53	5.92	7.91	2.66	1.87	1.44	2.83	4.28	2.41	2.85	0.57	0.78	0.97	0.56	0.47	0.8	0.32	0.14	0.59	(<class 'ZeroDivisionError'>, 162)
498	9	1.18	5.37	8.92	3.74	7.01	5.02	8.09	3.07	5.16	1.1	4.29	4.03	1.79	2.68	3.47	4.49	2.53	3.87	0.86	0.4	0.47	0.29	0.11	0.93	0.48	0.26	0.24	(<class 'ValueError'>, 216)
499	9	6.97	7.05	9.54	1.39	7.63	9.22	8.68	9.33	5.74	5.59	5.03	2.38	1.32	4.78	1.31	5.32	5.74	1.02	0.96	0.52	0.83	0.45	0.17	0.41	0.63	0.51	0.42	(<class 'ZeroDivisionError'>, 162)
500	9	8.03	1.24	3.3	9.91	5.09	6.48	5.32	5.39	6.27	7.88	1.06	2.78	4.37	4.01	1.45	3.12	2.95	1.49	0.91	0.89	0.96	0.29	0.95	0.9	0.74	0.43	0.66	NoError
501	9	5.01	1.49	9.23	2.33	1.54	6.63	2.27	4.6	4.22	1.78	1.33	5.16	1.32	1.17	6.12	2.04	4.03	3.48	0.59	0.32	0.93	0.5	0.76	0.21	0.93	0.28	0.19	(<class 'ZeroDivisionError'>, 162)