Write-Up

How I Structured the Code

The code is organized around a modular MobiusStrip class. Upon initialization, the class takes in the radius R, strip width w, and resolution n, then generates a 3D mesh using the standard parametric equations of a Möbius strip. The core functionalities are split into methods:

- _compute_mesh() creates the (X, Y, Z) grid.
- plot() visualizes the surface in 3D using Matplotlib.
- surface_area() estimates the total surface area numerically.
- edge_length() calculates the total length of both edges.
  This structure ensures clean separation between geometry generation, visualization, and numerical analysis.

How I Approximated Surface Area

To estimate the surface area, I used a numerical technique based on the surface integral:

A=∬||∂r∂u×∂r∂v|| du dvA = \iint ||\frac{\partial \mathbf{r}}{\partial u} \times \frac{\partial \mathbf{r}}{\partial v}|| \, du\,dvA=∬||∂u∂r×∂v∂r||dudv

Instead of computing symbolic derivatives, I used finite differences to approximate the partial derivatives at each point. I then computed the cross product of these tangent vectors and integrated the magnitude over the full domain (u∈[0,2π],v∈[−w/2,w/2])(u \in [0, 2\pi], v \in [-w/2, w/2])(u∈[0,2π],v∈[−w/2,w/2]) using scipy.integrate.dblquad.

Any Challenges I Faced

One challenge was maintaining numerical accuracy in the surface area calculation while keeping performance reasonable. Finite difference methods are sensitive to step size, so I had to carefully balance between precision and speed. Another subtle challenge was ensuring that the Möbius strip rendered correctly with the expected half-twist and did not suffer from mesh overlap or orientation errors.

Visualization: