AIM :

Creating a Mobius strip using parametric equations and computes key geometric properties.

Requirements

Define a MobiusStrip class that:

Accepts:

- Radius R (distance from the center to the strip)
- Width w (strip width)
- Resolution n (number of points in the mesh)

Computes:

- A 3D mesh/grid of (x, y, z) points on the surface
- Surface area (numerically, using integration or approximation)
- Edge length (numerically along the boundary)

Software Requirements

Operating System : Windows 10/11, macOS, or any Linux distribution

Python Version : Python 3.7 or higher

Libraries : NumPy (for matrix computations), Matplotlib (for 3D plotting)

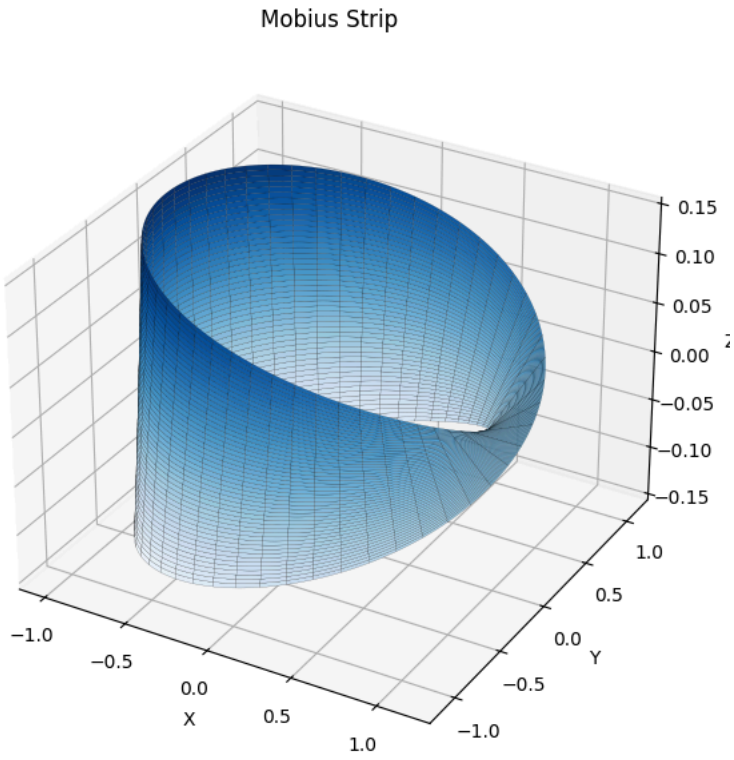
IDE/Editor : VS Code, PyCharm, Jupyter Notebook, or Google Colab

Code implementation

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

class MobiusStrip:
    def __init__(self, R=1.0, w=0.2, n=4):
        self.R = R
        self.w = w
        self.n = n
        self.u, self.v = np.meshgrid(
            np.linspace(0, 2 * np.pi, n),
            np.linspace(-w / 2, w / 2, n)
        )
        self.x, self.y, self.z = self.compute_mesh()

    def compute_mesh(self):
        u = self.u
        v = self.v
        x = (self.R + v * np.cos(u / 2)) * np.cos(u)
        y = (self.R + v * np.cos(u / 2)) * np.sin(u)
        z = v * np.sin(u / 2)
        return x, y, z

    def surface_area(self):
        du = 2 * np.pi / (self.n - 1)
        dv = self.w / (self.n - 1)

        xu = - (self.R + self.v * np.cos(self.u / 2)) * np.sin(self.u) - \
             self.v * np.sin(self.u / 2) * 0.5 * np.cos(self.u)
        yu = (self.R + self.v * np.cos(self.u / 2)) * np.cos(self.u) - \
             self.v * np.sin(self.u / 2) * 0.5 * np.sin(self.u)
        zu = self.v * 0.5 * np.cos(self.u / 2)

        xv = np.cos(self.u / 2) * np.cos(self.u)
        yv = np.cos(self.u / 2) * np.sin(self.u)
        zv = np.sin(self.u / 2)
```

```
            cross_x = yu * zv - zu * yv
            cross_y = zu * xv - xu * zv
            cross_z = xu * yv - yu * xv

            dA = np.sqrt(cross_x**2 + cross_y**2 + cross_z**2)
            area = np.sum(dA) * du * dv
            return area

        def edge_length(self):
            u = np.linspace(0, 2 * np.pi, self.n)
            v = self.w / 2

            def param_curve(v_sign):
                x = (self.R + v_sign * np.cos(u / 2)) * np.cos(u)
                y = (self.R + v_sign * np.cos(u / 2)) * np.sin(u)
                z = v_sign * np.sin(u / 2)
                return x, y, z

            edge_lengths = []
            for sign in [-1, 1]:
                x, y, z = param_curve(sign * v)
                dx = np.diff(x)
                dy = np.diff(y)
                dz = np.diff(z)
                edge_len = np.sum(np.sqrt(dx**2 + dy**2 + dz**2))
                edge_lengths.append(edge_len)

            return sum(edge_lengths)

        def plot(self):
            fig = plt.figure(figsize=(8, 6))
            ax = fig.add_subplot(111, projection='3d')
            ax.plot_surface(self.x, self.y, self.z, cmap='Blues', edgecolor='k', linewidth=0.1)
            ax.set_title("Mobius Strip")
            ax.set_xlabel("X")
            ax.set_ylabel("Y")
            ax.set_zlabel("Z")
            plt.tight_layout()
            plt.show()

    if __name__ == "__main__":
        mobius = MobiusStrip(R=1, w=0.3, n=500)
        print(f"Surface Area ≈ {mobius.surface_area():.4f}")
        print(f"Edge Length ≈ {mobius.edge_length():.4f}")
        mobius.plot()
```

```
Surface Area ≈ 1.8943
Edge Length ≈ 12.6020
```



Code Explanation

Importing Modules

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
```

A class to model and analyze a Möbius strip using parametric equations.

init(self, radius=1.0, width=0.2, resolution=100)

Purpose: Initializes the Möbius strip parameters and prepares the grid for computation.

Parameters:

- radius: The distance from the center of the Möbius strip to its centerline (R).
- width: The width of the strip (w).
- resolution: Number of points used to discretize the parametric domain, controls smoothness.

Process:

Uses np.meshgrid to create a grid over parameters u and v:

u ranges from 0 to 2π v ranges from −w/2 to w/2(across the width)

Calls compute_mesh() to calculate the 3D coordinates of all points.

```
class MobiusStrip:
    def __init__(self, R=1.0, w=0.2, n=100):
        self.R = R
        self.w = w
        self.n = n
        self.u, self.v = np.meshgrid(
            np.linspace(0, 2 * np.pi, n),
            np.linspace(-w / 2, w / 2, n)
        )
        self.x, self.y, self.z = self.compute_mesh()
```

Compute_mesh:

Calculates the 3D coordinates (x, y, z) of the Möbius strip surface using

parametric equations:

$$x = \left(R + v\cos\frac{u}{2}\right)\cos u$$

$$y = \left(R + v\cos\frac{u}{2}\right)\sin u$$

$$z = v\sin\frac{u}{2}$$

```
  def compute_mesh(self):
        u = self.u
        v = self.v
        x = (self.R + v * np.cos(u / 2)) * np.cos(u)
        y = (self.R + v * np.cos(u / 2)) * np.sin(u)
        z = v * np.sin(u / 2)
        return x, y, z
```

Returns Three 2D arrays: x, y, and z, representing coordinates on the surface.

Surface_area

Purpose: Approximates the surface area of the Möbius strip using numerical integration of the differential surface element.

How it works:

- Calculates partial derivatives of the position vector with respect to parameters u and v.
- Computes the cross product of these derivatives at each point to find the local surface area element.
- Sums over all elements multiplied by du and dv (discrete parameter steps) to get total area.

```
def surface_area(self):
        du = 2 * np.pi / (self.n - 1)
        dv = self.w / (self.n - 1)

        # Partial derivatives
        xu = - (self.R + self.v * np.cos(self.u / 2)) * np.sin(self.u) - \
            self.v * np.sin(self.u / 2) * 0.5 * np.cos(self.u)
        yu = (self.R + self.v * np.cos(self.u / 2)) * np.cos(self.u) - \
            self.v * np.sin(self.u / 2) * 0.5 * np.sin(self.u)
        zu = self.v * 0.5 * np.cos(self.u / 2)

        xv = np.cos(self.u / 2) * np.cos(self.u)
        yv = np.cos(self.u / 2) * np.sin(self.u)
        zv = np.sin(self.u / 2)

        # Cross product of partial derivatives
        cross_x = yu * zv - zu * yv
        cross_y = zu * xv - xu * zv
        cross_z = xu * yv - yu * xv

        dA = np.sqrt(cross_x**2 + cross_y**2 + cross_z**2)
        area = np.sum(dA) * du * dv
        return area
```

Returns: A float representing the estimated surface area.

Edge_Length

Purpose: Estimates the total length of the Möbius strip edge.

How it works:

- Möbius strip has a single edge traced twice mathematically; here, computes two boundaries at v=+w/2 and v=-w/2
- For each boundary, computes the 3D coordinates along u and calculates the length by summing distances between consecutive points.
- Adds lengths of both edges to get total edge length.

```
def edge_length(self):
        # Evaluate along v = ±w/2
        u = np.linspace(0, 2 * np.pi, self.n)
        v = self.w / 2

        def param_curve(v_sign):
            x = (self.R + v_sign * np.cos(u / 2)) * np.cos(u)
            y = (self.R + v_sign * np.cos(u / 2)) * np.sin(u)
            z = v_sign * np.sin(u / 2)
            return x, y, z

        edge_lengths = []
        for sign in [-1, 1]:
            x, y, z = param_curve(sign * v)
            dx = np.diff(x)
            dy = np.diff(y)
            dz = np.diff(z)
            edge_len = np.sum(np.sqrt(dx**2 + dy**2 + dz**2))
            edge_lengths.append(edge_len)

        return sum(edge_lengths)
```

Returns: A float representing the approximate total edge length.

Plotting (Visualization)

Purpose: Visualizes the Möbius strip as a 3D surface plot.

How it works:

- Creates a matplotlib figure with a 3D axis.
- Uses ax.plot_surface with the precomputed x, y, z arrays.
- Applies a colormap (plasma), sets axis labels and title, and displays the plot.

```
def plot(self):
    # Create a new figure window with a specific size (8 inches wide by 6 inches tall)
    fig = plt.figure(figsize=(8, 6))

    # Add a 3D subplot to the figure. '111' means 1 row, 1 column, 1st subplot.
    ax = fig.add_subplot(111, projection='3d')

    # Plot the surface of the Möbius strip using the computed x, y, z mesh.
    # 'viridis' is the colormap; 'edgecolor' makes edges black; 'linewidth' sets edge thickness.
    ax.plot_surface(self.x, self.y, self.z, cmap='viridis', edgecolor='k', linewidth=0.1)

    # Set the title of the plot.
    ax.set_title("Mobius Strip")

    # Label the X-axis.
    ax.set_xlabel("X")

    # Label the Y-axis.
    ax.set_ylabel("Y")

    # Label the Z-axis.
    ax.set_zlabel("Z")

    # Automatically adjust the layout to prevent overlapping elements.
    plt.tight_layout()

    # Display the plot window.
    plt.show()
```

Colour codes

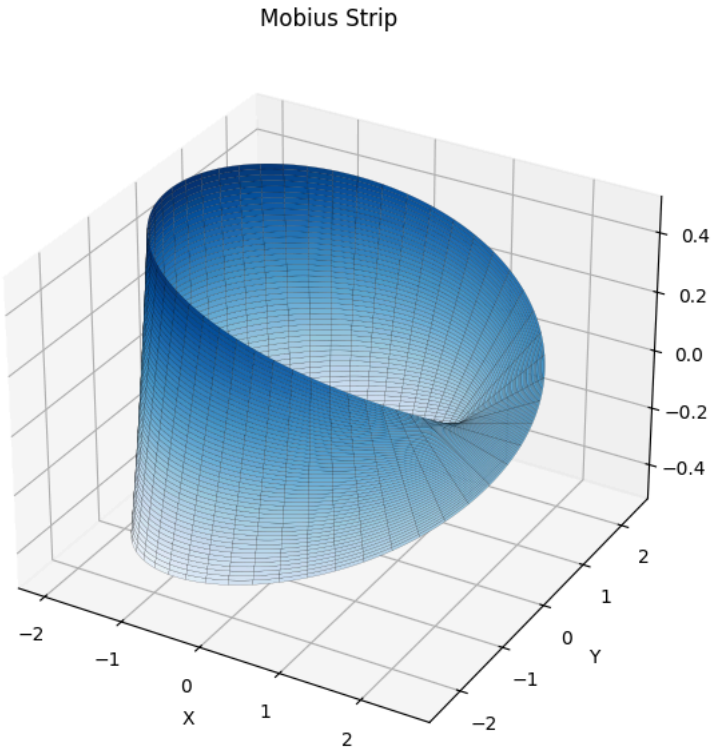
The cmap argument accepts many built-in colormaps from Matplotlib like:

- viridis (used in my code)
- plasma
- inferno
- magma
- cividis
- coolwarm
- winter
- autumn
- spring
- summer

Function Calling

```
if __name__ == "__main__":
    mobius = MobiusStrip(R=1, w=0.3, n=200)
    print(f"Surface Area ≈ {mobius.surface_area():.4f}")
    print(f"Edge Length ≈ {mobius.edge_length():.4f}")
    mobius.plot()
```

Output image



Challenges Faced

- I faced a few challenges while working on this. At first, it was a bit hard to understand the shape of the Möbius strip and how to write it using 3D coordinates. Figuring out how to make the strip look correct with the right formulas took some time.
- Also, making the 3D plot look good was not easy. I had to adjust things like the colors, edges, and axis settings to get a clear and neat visualization. It took a few tries to get everything looking right.

How we calculate the surface Area

- The Möbius strip is defined using two parameters u and v. We calculate the surface coordinates (x, y, z) based on these parameters.
- We compute the partial derivatives of the surface with respect to u and v. These represent small changes along the surface in each direction.
- We take the cross product of these two partial derivatives. The magnitude of this cross product gives the area of a small surface patch at each point.
- We use np.sum to add up the area of all these small patches and multiply by the small step sizes du and dv to approximate the total surface area.

References and Formulas Used

1. Parametric Equations of Möbius Strip

The Möbius strip is defined parametrically by two parameters u and v

$$\begin{cases} x(u,v) = \left(R + v\cos\frac{u}{2}\right)\cos u \\ y(u,v) = \left(R + v\cos\frac{u}{2}\right)\sin u \\ z(u,v) = v\sin\frac{u}{2} \end{cases} \quad u \in [0, 2\pi], \quad v \in \left[-\frac{w}{2}, \frac{w}{2}\right]$$

R is the radius from the center of the strip to the centerline.

w is the width of the strip.

u parameterizes the circular direction.

v parameterizes the width across the strip.

2. Surface Area Approximation

The surface area A is computed using the formula:

$$A = \iint_D \left| \frac{\partial \mathbf{r}}{\partial u} \times \frac{\partial \mathbf{r}}{\partial v} \right| du\, dv$$

Where:

r(u,v)=(x(u,v),y(u,v),z(u,v)) is the position vector.

The magnitude of this vector gives the infinitesimal surface area element.

Numerical integration is done by discretizing u and v into a grid of size n×n.

3. Partial Derivatives

Partial derivatives with respect to u and v are computed as:

$$\frac{\partial \mathbf{r}}{\partial u} = \left(\frac{\partial x}{\partial u}, \frac{\partial y}{\partial u}, \frac{\partial z}{\partial u} \right)$$

$$\frac{\partial \mathbf{r}}{\partial v} = \left(\frac{\partial x}{\partial v}, \frac{\partial y}{\partial v}, \frac{\partial z}{\partial v} \right)$$

These are derived using standard differentiation rules on the parametric equations above.

4. Edge Length Approximation

The Möbius strip has a single continuous edge, but computationally we approximate the edge length by summing lengths of the curves at the boundaries v=+w/2 or -w/2

$$L = \int_0^{2\pi} \left| \frac{d\mathbf{r}}{du} \right| du \quad \text{at} \quad v = \pm \frac{w}{2}$$

$$\left| \frac{d\mathbf{r}}{du} \right| = \sqrt{\left(\frac{dx}{du} \right)^2 + \left(\frac{dy}{du} \right)^2 + \left(\frac{dz}{du} \right)^2}$$

Numerically, this is approximated by summing distances between discrete points on the curves.

Start coding or generate with AI.