# Mathematical Representation of a Mobius Strip in 3D

## Definition:

A Mobius strip is a non-orientable surface with only one side and one continuous boundary. It is formed by taking a rectangular strip of paper, twisting one end by 180 degrees, and then joining it to the other end.

This simple yet fascinating shape has the following unique properties:

- It has only one surface — if you start drawing a line along the surface, you'll eventually return to the starting point having covered both "sides" without lifting your pen.
- It has **only one edge**.
- It's a popular object in **topology**, a branch of mathematics studying shapes and surfaces.

## Parametric Equations of Mobius Strip

Use the parametric equations:

$x(u,v)=(R+v\cdot\cos(u2))\cdot\cos(u)$

$y(u,v)=(R+v\cdot\cos(u2))\cdot\sin(u)$

$z(u,v)=v\cdot\sin(u2)$

Where:

- $u\in[0,2\pi]$
- $v\in[-w/2,w/2]$

| Term | Meaning |
|---|---|
| R | Radius of the central circular path. |
| W | Width of the strip. |
| n | Resolution (number of points used to create the mesh grid). |
| u | Angle going from 0 to 2π |
| v | Distance across the strip, from −w/2-w/2−w/2 to +w/2+w/2+w/2 (the width). |

- As uuu goes from 0 to 2π2\pi2π, the strip loops around a full circle.
- As vvv moves from −w/2-w/2−w/2 to w/2w/2w/2, we cover the thickness of the strip.
- The half-twist is introduced by using u/2u/2u/2 inside sine and cosine.

**Step 1: Initialization**

def __init__(self, R, w, n):

- Stores the radius R, width w, and resolution n.
- Generates u and v values using np.linspace().
- Calls generate_mesh() to compute X, Y, Z points of the surface.

**Step 2: Mesh Generation**

def generate_mesh(self)

u, v = np.meshgrid(self.u_vals, self.v_vals)

- Creates a grid of all combinations of u and v.
- Then uses parametric equations to compute each (x, y, z):

**Step 3: Surface Area Calculation**

def compute_surface_area(self):

**Numerical Method:**

1. Calculate small steps:
   du = 2 * np.pi / (self.n - 1)
   dv = self.w / (self.n - 1)
2. Get partial derivatives (change in x, y, z in u and v directions):
   Xu = np.gradient(self.X, du, axis=1)
   Yu = np.gradient(self.Y, du, axis=1)
   Zu = np.gradient(self.Z, du, axis=1)

   Xv = np.gradient(self.X, dv, axis=0)
   Yv = np.gradient(self.Y, dv, axis=0)
   Zv = np.gradient(self.Z, dv, axis=0)

3. Compute the **cross product** of derivatives:
   This gives a small surface patch area vector.
   cross_x = Yu * Zv - Zu * Yv
   cross_y = Zu * Xv - Xu * Zv
   cross_z = Xu * Yv - Yu * Xv

4. Surface element dA is magnitude of the cross product:
   dA = np.sqrt(cross_x**2 + cross_y**2 + cross_z**2)

5. Sum up all patches:
   return np.sum(dA) * du * dv

**Step 4: Edge Length Calculation**

To **calculate the total edge length** of the Möbius strip, we want to measure the length along the **two sides** of the strip:

- **edge1:** the strip at v=−w/2v = -w/2v=−w/2 (bottom edge in the width)
- **edge2:** the strip at v=+w/2v = +w/2v=+w/2 (top edge)

Even though a Möbius strip has one continuous edge **physically**, when modeled in code with parametric coordinates, we handle it using these two ends for calculation.

### 🔹 Extracting the Edges

```
edge1 = (self.X[0, :], self.Y[0, :], self.Z[0, :])
edge2 = (self.X[-1, :], self.Y[-1, :], self.Z[-1, :])
```

- self.X[0, :] gets all **x-values** along the first row (where v=−w/2v = -w/2v=−w/2) → edge1
- self.X[-1, :] gets all **x-values** along the last row (where v=+w/2v = +w/2v=+w/2) → edge2

You do the same for Y and Z, so:

- edge1 and edge2 are **tuples** of arrays: (x,y,z)(x, y, z)(x,y,z) for each edge.

### 🔹 Function to Compute Arc Length of an Edge

```
def arc_length(edge):
    dx = np.diff(edge[0])
    dy = np.diff(edge[1])
    dz = np.diff(edge[2])
    return np.sum(np.sqrt(dx**2 + dy**2 + dz**2))
```

This function computes the **total distance** between consecutive points on the edge:

1. np.diff(edge[0]) = difference between adjacent x-values.
2. Similarly for y and z.

These give:

- dx[i] = xi+1−xix_{i+1} - x_ixi+1−xi

- dy[i] = yi+1−yiy_{i+1} - y_iyi+1−yi
- dz[i] = zi+1−ziz_{i+1} - z_izi+1−zi

Each tiny segment length is calculated using the **3D distance formula**:
Then we **sum up** all those distances using np.sum(...).

$$\text{length} = \sqrt{dx^2 + dy^2 + dz^2}$$

This gives the **arc length of the edge**.

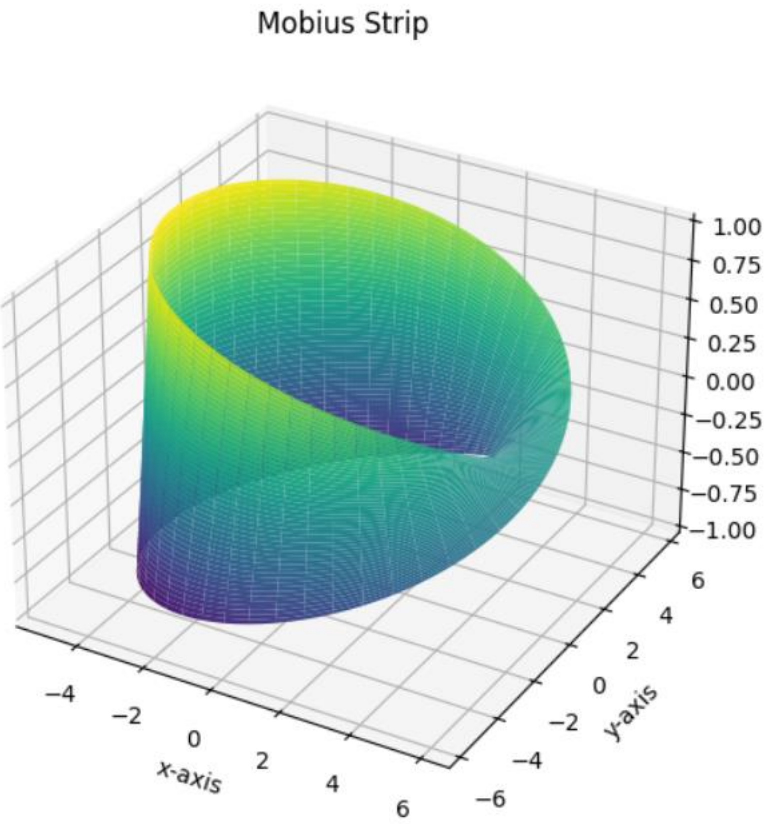
**◈ Return Total Edge Length**

return arc_length(edge1) + arc_length(edge2)

We calculate the arc length of both edge1 and edge2 and add them together. This gives the **total "numerical" edge length** of the Möbius strip mesh (even though it's topologically a single edge).

**Step 5: Plotting**

Surface Area: 64.17458381956531
Edge Length: 63.14092989098662



**Brief Into about functions used to create 3D model of MobiusStrip:**

**1. numpy.linspace(start, stop, num)**

**Purpose:** Creates an evenly spaced array of values between start and stop.

**Syntax:**

np.linspace(start, stop, num)

**2. numpy.meshgrid(x, y)**

**Purpose:** Creates coordinate matrices from coordinate vectors (useful for surface plotting).

**Syntax:**

X, Y = np.meshgrid(x, y)

**3.numpy.gradient(f, dx, axis)**

**Purpose**: Computes numerical derivative (gradient) of array f along a specified axis.

**Syntax:**

np.gradient(f, dx, axis=0 or 1)

**4.numpy.sqrt(x)**

**Purpose**: Computes square root element-wise.

**5.numpy.diff(array)**

**Purpose**: Calculates difference between consecutive elements.

**6.add_subplot() from matplotlib**

**Purpose**: Adds a subplot to a figure, optionally 3D.