

Creating a tube along a trefoil knot

- using Matplotlib, NumPy and scikit-vectors

Copyright (c) 2017, 2019 Tor Olav Kristensen, <http://subcube.com> (<http://subcube.com>).

<https://github.com/t-o-k/scikit-vectors> (<https://github.com/t-o-k/scikit-vectors>).

Use of this source code is governed by a BSD-license that can be found in the LICENSE file.

```
In [1]: 1 # This example has been tested with NumPy v1.15.3, Matplotlib v2.1.1. and Jupyter v4.4.0
```

```
In [2]: 1 # Uncomment one of these to get a Matplotlib backend with interactive plots
2
3 # %matplotlib auto
4 # %matplotlib notebook
```

```
In [3]: 1 # Get the necessary libraries
2
3 import matplotlib.colors as colors
4 import matplotlib.pyplot as plt
5 import matplotlib.tri as mtri
6 from mpl_toolkits.mplot3d import Axes3D
7 from mpl_toolkits.mplot3d.art3d import Poly3DCollection
8 import numpy as np
9
10 from skvectors import create_class_Cartesian_3D_Vector
```

```
In [4]: 1 # Size and resolution for Matplotlib figures
2
3 figure_size = (8, 6)
4 figure_dpi = 100
```

```
In [5]: 1 # The functions for the trefoil knot curve
        2
        3 def f_x(t):
        4     return np.cos(t) + 2 * np.cos(2 * t)
        5
        6
        7
        8 def f_y(t):
        9     return np.sin(t) - 2 * np.sin(2 * t)
        10
        11
        12
        13 def f_z(t):
        14     return 2 * np.sin(3 * t)
        15
```

```
In [6]: 1 # The first derivatives of the functions for the curve
        2
        3 def d1_f_x(t):
        4     return -np.sin(t) - 4 * np.sin(2 * t)
        5
        6
        7
        8 def d1_f_y(t):
        9     return np.cos(t) - 4 * np.cos(2 * t)
        10
        11
        12
        13 def d1_f_z(t):
        14     return 6 * np.cos(3 * t)
        15
```

```
In [7]: 1 # The second derivatives of the functions for the curve
        2
        3 def d2_f_x(t):
        4
        5     return -np.cos(t) - 8 * np.cos(2 * t)
        6
        7
        8 def d2_f_y(t):
        9
        10    return -np.sin(t) + 8 * np.sin(2 * t)
        11
        12
        13 def d2_f_z(t):
        14
        15    return -18 * np.sin(3 * t)
```

```
In [8]: 1 # Resolutions for plot
        2
        3 nr_of_points_along_curve = 3 * 2**5 + 1
        4 nr_of_points_across_curve = 3 * 2**2 + 1
```

In [9]:

```
1  # Necessary NumPy functions
2
3  np_functions = \
4  {
5      'not': np.logical_not,
6      'and': np.logical_and,
7      'or': np.logical_or,
8      'all': np.all,
9      'any': np.any,
10     'min': np.minimum,
11     'max': np.maximum,
12     'abs': np.absolute,
13     'trunc': np.trunc,
14     'ceil': np.ceil,
15     'copysign': np.copysign,
16     'log10': np.log10,
17     'cos': np.cos,
18     'sin': np.sin,
19     'atan2': np.arctan2,
20     'pi': np.pi
21 }
```

In [10]:

```
1  # Make a vector class that can hold all the points along the curve
2
3  NP_3D_A1 = \
4      create_class_Cartesian_3D_Vector(
5          name = 'NP_3D_A1',
6          component_names = [ 'x', 'y', 'z' ],
7          brackets = '<>',
8          sep = ', ',
9          cnull = np.zeros(nr_of_points_along_curve),
10         cunit = np.ones(nr_of_points_along_curve),
11         functions = np_functions
12     )
```

```
In [11]: 1 # Calculate the points along the curve
2
3 angles_along_curve = np.linspace(-np.pi, +np.pi, nr_of_points_along_curve, endpoint=True)
4
5 p_o = \
6     NP_3D_A1(
7         x = f_x(angles_along_curve),
8         y = f_y(angles_along_curve),
9         z = f_z(angles_along_curve)
10    )
```

```
In [12]: 1 # Vectors from the first derivatives at the points along the curve
2
3 v_d1 = \
4     NP_3D_A1(
5         x = d1_f_x(angles_along_curve),
6         y = d1_f_y(angles_along_curve),
7         z = d1_f_z(angles_along_curve)
8    )
```

```
In [13]: 1 # Vectors from the second derivatives at the points along the curve
2
3 v_d2 = \
4     NP_3D_A1(
5         x = d2_f_x(angles_along_curve),
6         y = d2_f_y(angles_along_curve),
7         z = d2_f_z(angles_along_curve)
8    )
```

```
In [14]: 1 # Calculate the vectors for all the Frenet frames along the curve
2
3 # Tangent vectors at the points along the curve
4 v_t = v_d1.normalize()
5
6 # Binormal vectors at the points along the curve
7 v_b = v_d1.cross(v_d2).normalize()
8
9 # Normal vectors at the points along the curve
10 v_n = v_t.cross(v_b)
```

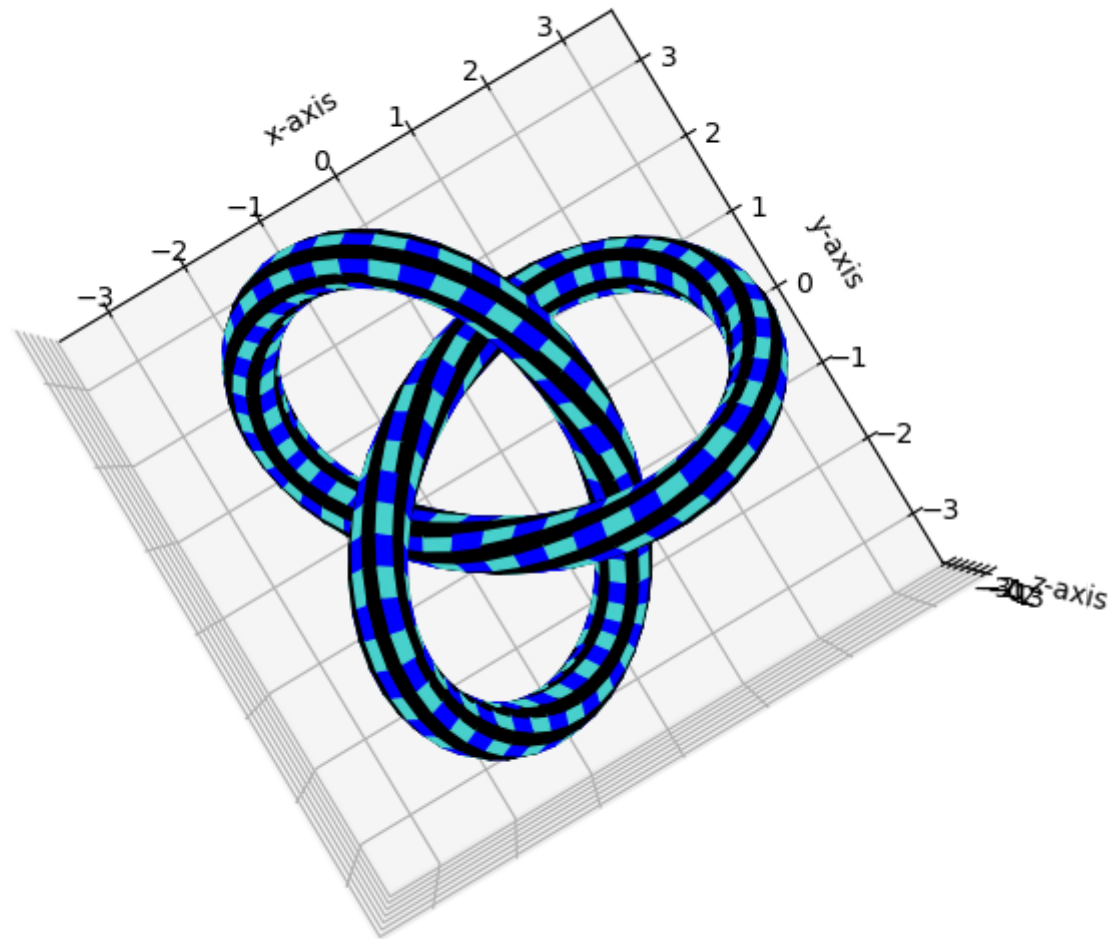
In [15]:

```
1  # For all the points along the curve, calculate points in a circle across the curve
2
3  angles_across_curve = np.linspace(-np.pi, +np.pi, nr_of_points_across_curve, endpoint=True)
4
5  tube_radius = 0.3
6
7  surface_points = \
8      [
9          p_o + v_n.axis_rotate(v_t, angle) * tube_radius
10     #      p_o + v_n.axis_rotate(v_t, angle) * tube_radius * (3 + np.sin(2 * angle)) / 2
11     for angle in angles_across_curve
12     ]
```

In [16]:

```
1  # Show the trefoil knot tube
2
3  fig = plt.figure(figsize=figure_size, dpi=figure_dpi)
4  ax = Axes3D(fig)
5  ax.set_aspect(1)
6  ax.set_title('Trefoil Knot Tube')
7  for j in range(nr_of_points_along_curve-1):
8      for i in range(nr_of_points_across_curve-1):
9          if i % 2 == 0:
10             color = 'black'
11          else:
12             if j % 2 == 0:
13                 color = 'blue'
14             else:
15                 color = 'mediumturquoise'
16             x0, y0, z0 = surface_points[i ]
17             x1, y1, z1 = surface_points[i+1]
18             p00 = (x0[j ], y0[j ], z0[j ])
19             p01 = (x0[j+1], y0[j+1], z0[j+1])
20             p10 = (x1[j ], y1[j ], z1[j ])
21             p11 = (x1[j+1], y1[j+1], z1[j+1])
22             triangle_a = Poly3DCollection([ [ p00, p10, p11 ] ])
23             triangle_a.set_color(color)
24             triangle_a.set_edgecolor(color)
25             ax.add_collection3d(triangle_a)
26             triangle_b = Poly3DCollection([ [ p11, p01, p00 ] ])
27             triangle_b.set_color(color)
28             triangle_b.set_edgecolor(color)
29             ax.add_collection3d(triangle_b)
30 ax.set_xlim(-3.5, +3.5)
31 ax.set_ylim(-3.5, +3.5)
32 ax.set_zlim(-3.5, +3.5)
33 ax.set_xlabel('x-axis')
34 ax.set_ylabel('y-axis')
35 ax.set_zlabel('z-axis')
36 ax.view_init(elev=90, azimuth=-120)
37
38 plt.show()
```

Trefoil Knot Tube



In [17]: 1 # Now do it in another way


```
In [18]: 1 # Make a vector class that can hold all the points on the surface of the tube
2
3 surface_shape = (nr_of_points_across_curve, nr_of_points_along_curve)
4 zeros = np.zeros(surface_shape)
5 ones = np.ones(surface_shape)
6
7 NP_3D_A2 = \
8     create_class_Cartesian_3D_Vector(
9         name = 'NP_3D_A2',
10         component_names = [ 'xx', 'yy', 'zz' ],
11         brackets = [ '<<', '>>' ],
12         sep = ', ',
13         cnull = zeros,
14         cunit = ones,
15         functions = np_functions
16     )
```

```
In [19]: 1 # Verify that NumPy's array broadcasting works as needed
2
3 A1_cunit = NP_3D_A1.component_unit()
4 A2_cunit = NP_3D_A2.component_unit()
5
6 assert (A2_cunit * A1_cunit).shape == surface_shape
```

```
In [20]: 1 # Initialize position vectors for the points
2 # (The 1D arrays are being broadcasted to 2D arrays)
3
4 pp_o = \
5     NP_3D_A2(
6         xx = p_o.x,
7         yy = p_o.y,
8         zz = p_o.z
9     )
```

```
In [21]: 1 # Alternative ways to do the same
2
3 # pp_o = \
4 #     NP_3D_A2(
5 #         xx = A2_cunit * p_o.x,
6 #         yy = A2_cunit * p_o.y,
7 #         zz = A2_cunit * p_o.z
8 #     )
9
10 # tile_size = (nr_of_points_across_curve, 1)
11 # pp_o = \
12 #     NP_3D_A2(
13 #         xx = np.tile(p_o.x, tile_size),
14 #         yy = np.tile(p_o.y, tile_size),
15 #         zz = np.tile(p_o.z, tile_size)
16 #     )
```

```
In [22]: 1 # Initialize tangent, binormal and normal vectors
2
3 vv_t = NP_3D_A2(xx=v_t.x, yy=v_t.y, zz=v_t.z)
4 vv_b = NP_3D_A2(xx=v_b.x, yy=v_b.y, zz=v_b.z)
5 vv_n = NP_3D_A2(xx=v_n.x, yy=v_n.y, zz=v_n.z)
```

```
In [23]: 1 # Set up 2D arrays for angles along and across the curve
2
3 angles_along, angles_across = np.meshgrid(angles_along_curve, angles_across_curve)
```

```
In [24]: 1 # Calculate all the vectors along and across the curve towards the surface of the tube
2
3 vv_s = vv_n.axis_rotate(vv_t, angles_across)
```

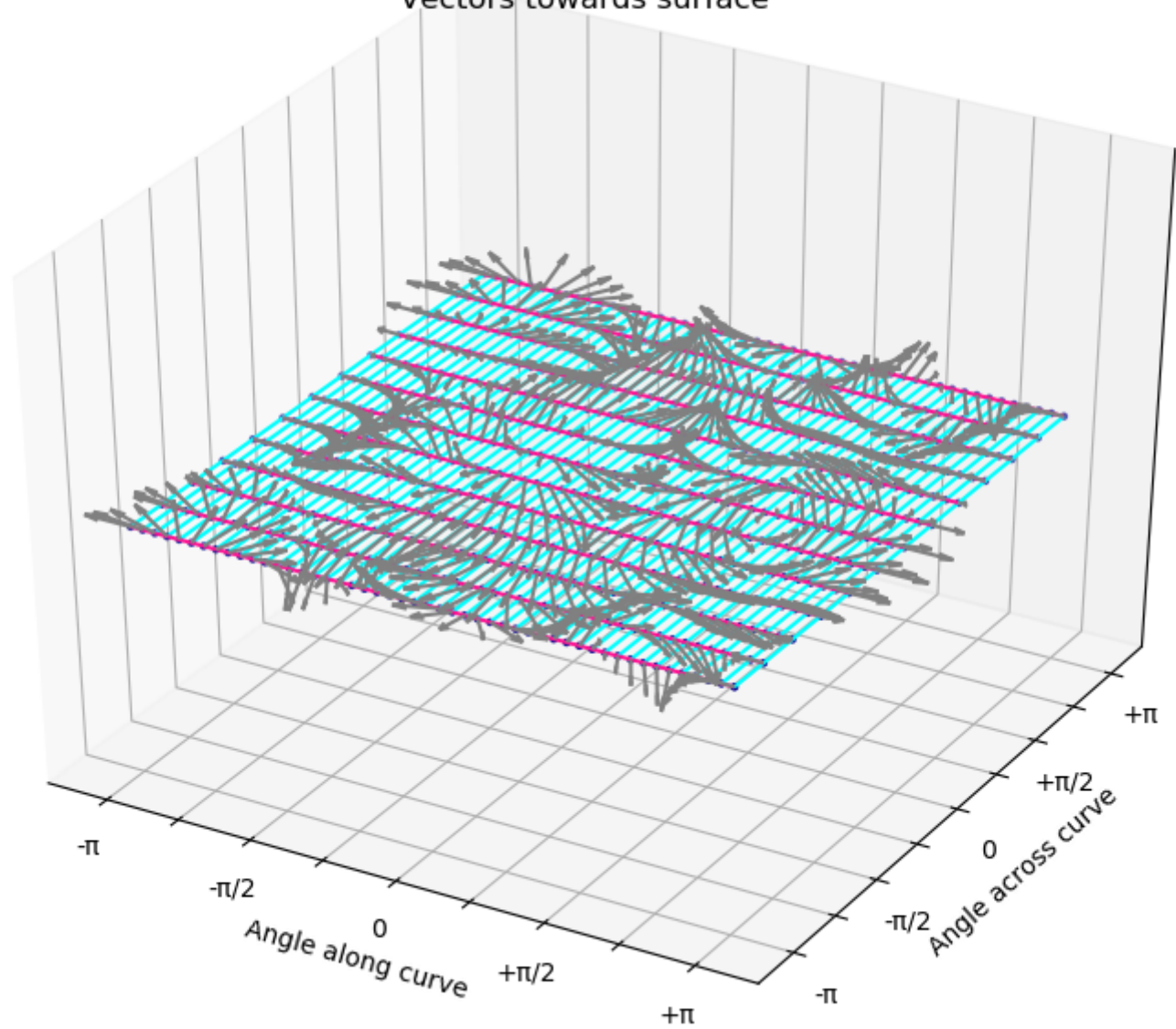
In [25]:

```
1  # Prepare some variables for plotting
2
3  no_labels = [ ]
4  no_ticks = [ ]
5
6  pi_labels = [ '-π', '', '-π/2', '', '0', '', '+π/2', '', '+π' ]
7  pi_ticks = \
8      [
9          n / 4 * np.pi
10         for n in [ -4, -3, -2, -1, 0, +1, +2, +3, +4 ]
11     ]
12
13  vector_length = 0.5
14
15  stride_along = 2
16  stride_across = 1
17
18  sl_along = slice(None, None, stride_along)
19  sl_across = slice(None, None, stride_across)
20  sl = (sl_across, sl_along)
```

In [26]:

```
1  # Show some of the vectors calculated above
2
3  fig = plt.figure(figsize=figure_size, dpi=figure_dpi)
4  ax = Axes3D(fig)
5  ax.set_title('Vectors towards surface')
6  ax.scatter(
7      angles_along[sl], angles_across[sl], zeros[sl],
8      color = 'darkblue',
9      marker = '.',
10     edgecolors = 'face'
11 )
12 ax.plot_wireframe(
13     angles_along, angles_across, zeros,
14     rstride = 0,
15     cstride = stride_along,
16     color = 'cyan'
17 )
18 ax.plot_wireframe(
19     angles_along, angles_across, zeros,
20     rstride = stride_across,
21     cstride = 0,
22     color = 'deeppink'
23 )
24 ax.quiver(
25     angles_along[sl], angles_across[sl], zeros[sl],
26     vv_s.xx[sl], vv_s.yy[sl], vv_s.zz[sl],
27     length = vector_length,
28     pivot = 'tail',
29     color = 'gray'
30 )
31 ax.set_xlim(-np.pi-0.5, +np.pi+0.5)
32 ax.set_ylim(-np.pi-0.5, +np.pi+0.5)
33 ax.set_zlim(-np.pi-0.5, +np.pi+0.5)
34 ax.set_xlabel('Angle along curve')
35 ax.set_ylabel('Angle across curve')
36 ax.set_xticklabels(pi_labels)
37 ax.set_yticklabels(pi_labels)
38 ax.set_zticklabels(no_labels)
39 ax.set_xticks(pi_ticks)
40 ax.set_yticks(pi_ticks)
41 ax.set_zticks(no_ticks)
42 ax.view_init(elev=36, azim=-60)
43 plt.show()
```

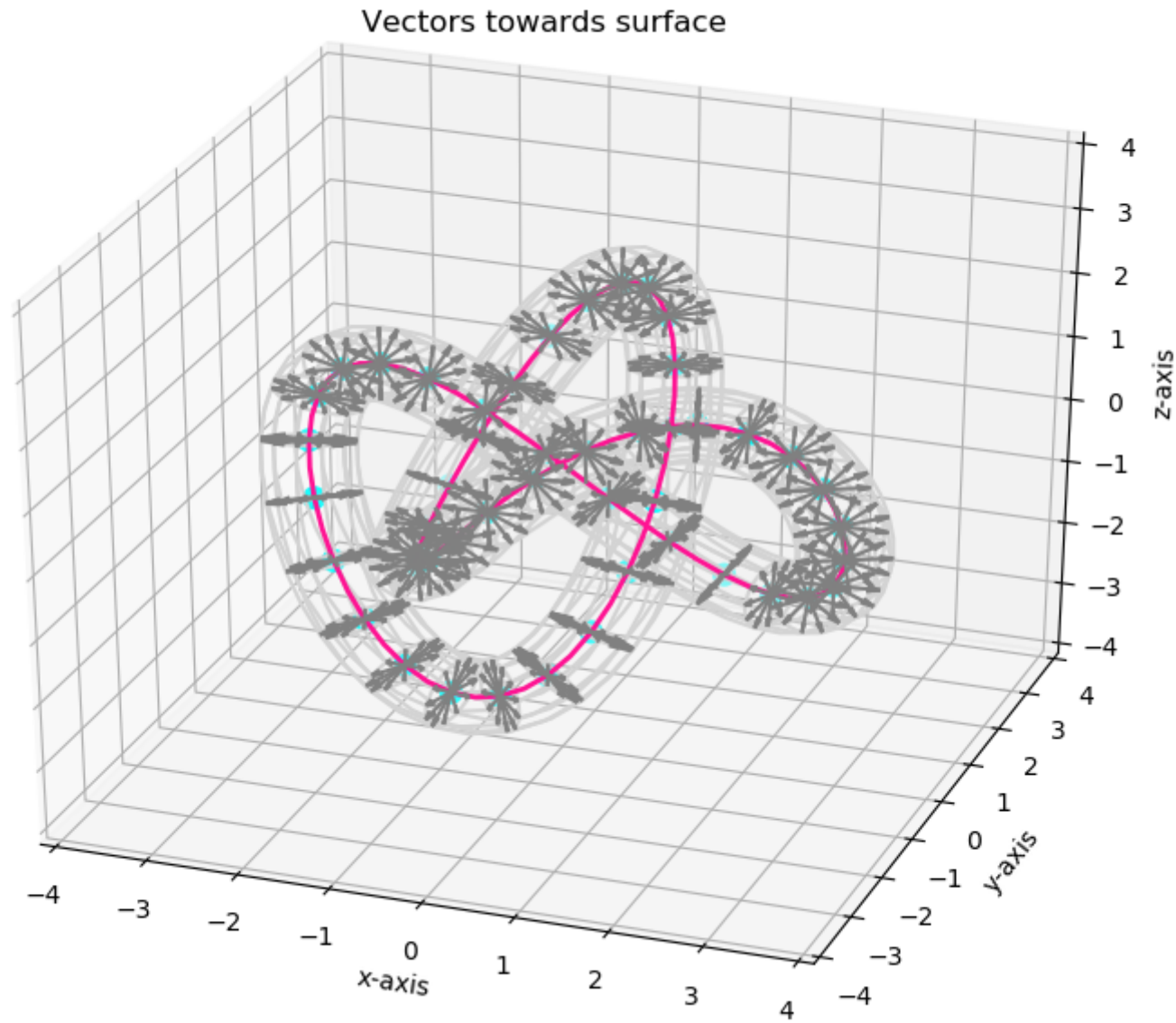
Vectors towards surface



In [27]:

```
1  # Show some of the vectors calculated above
2
3  pp_w = pp_o + vv_s *vector_length
4
5  fig = plt.figure(figsize=figure_size, dpi=figure_dpi)
6  ax = Axes3D(fig)
7  ax.set_title('Vectors towards surface')
8  ax.plot_wireframe(
9      pp_w.xx, pp_w.yy, pp_w.zz,
10     rstride = 0,
11     cstride = stride_along,
12     color = 'lightgray'
13 )
14 ax.plot(
15     p_o.x, p_o.y, p_o.z,
16     color = 'deeppink',
17     linewidth = 2
18 )
19 ax.plot_wireframe(
20     pp_w.xx, pp_w.yy, pp_w.zz,
21     rstride = stride_across,
22     cstride = 0,
23     color = 'lightgray'
24 )
25 ax.quiver(
26     pp_o.xx[sl], pp_o.yy[sl], pp_o.zz[sl],
27     vv_s.xx[sl], vv_s.yy[sl], vv_s.zz[sl],
28     length = vector_length,
29     pivot = 'tail',
30     color = 'gray'
31 )
32 ax.scatter(
33     p_o.x[sl_along], p_o.y[sl_along], p_o.z[sl_along],
34     color = 'cyan',
35     marker = 'o',
36     linewidth = 5
37 )
38 ax.set_xlabel('x-axis')
39 ax.set_ylabel('y-axis')
40 ax.set_zlabel('z-axis')
41 ax.set_xlim(-4, +4)
42 ax.set_ylim(-4, +4)
43 ax.set_zlim(-4, +4)
```

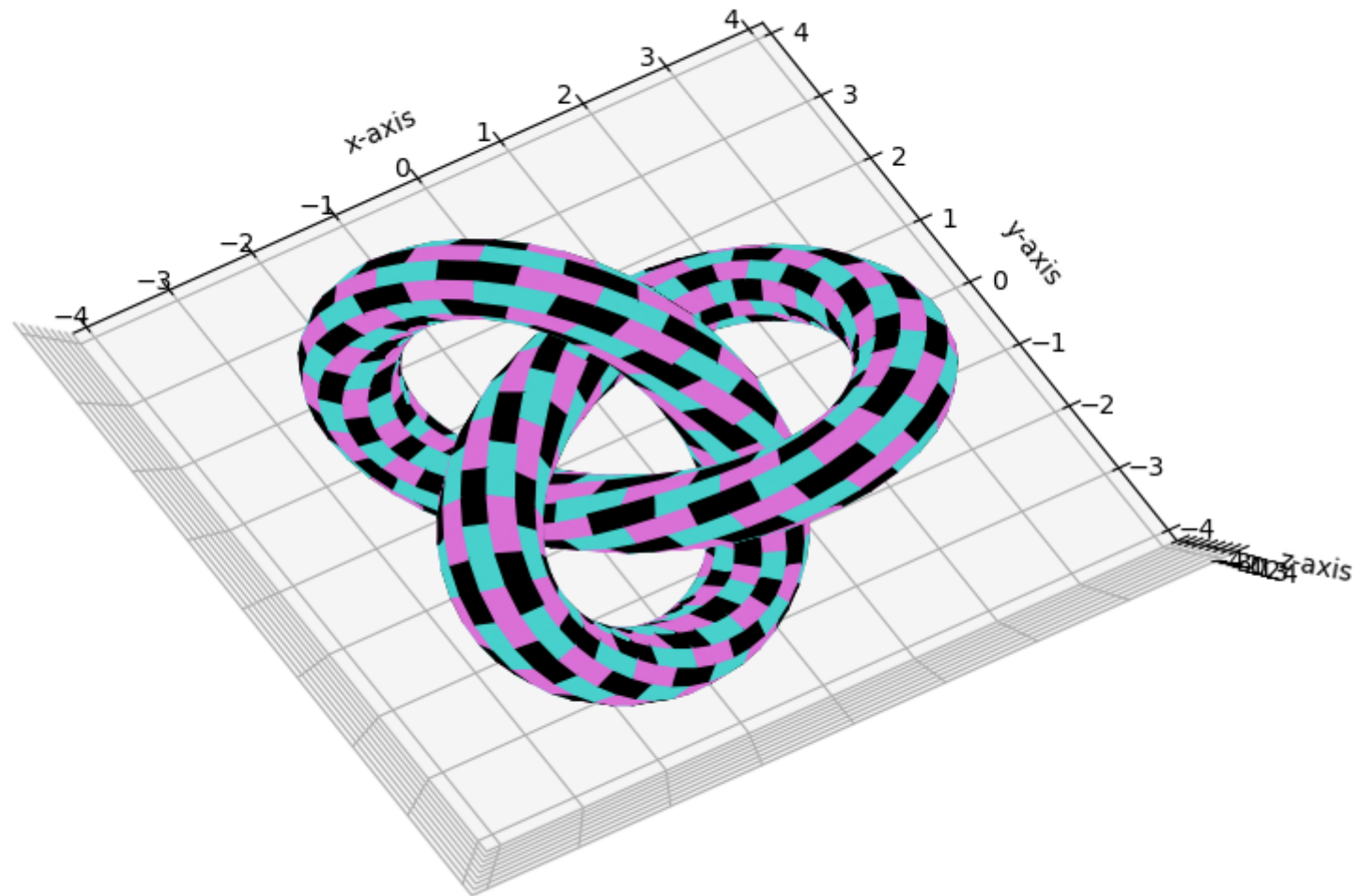
```
44 ax.view_init(elev=30, azim=-70)
45 plt.show()
```



In [28]:

```
1  # Show the trefoil knot tube
2
3  fig = plt.figure(figsize=figure_size, dpi=figure_dpi)
4  ax = Axes3D(fig)
5  ax.set_title('Trefoil Knot Tube with constant radius')
6  for j in range(nr_of_points_along_curve-1):
7      for i in range(nr_of_points_across_curve-1):
8          k = (3 * i + j) % 6
9          # k = (2 * i + 3 * j) % 6
10         if k < 2:
11             color = 'mediumturquoise'
12         elif k < 4:
13             color = 'black'
14         else:
15             color = 'orchid'
16         c00 = (i , j )
17         c01 = (i , j+1)
18         c10 = (i+1, j )
19         c11 = (i+1, j+1)
20         p00 = (pp_w.xx[c00], pp_w.yy[c00], pp_w.zz[c00])
21         p01 = (pp_w.xx[c01], pp_w.yy[c01], pp_w.zz[c01])
22         p10 = (pp_w.xx[c10], pp_w.yy[c10], pp_w.zz[c10])
23         p11 = (pp_w.xx[c11], pp_w.yy[c11], pp_w.zz[c11])
24         triangle_a = Poly3DCollection([ [ p00, p10, p11 ] ])
25         triangle_a.set_color(color)
26         triangle_a.set_edgecolor(color)
27         ax.add_collection3d(triangle_a)
28         triangle_b = Poly3DCollection([ [ p11, p01, p00 ] ])
29         triangle_b.set_color(color)
30         triangle_b.set_edgecolor(color)
31         ax.add_collection3d(triangle_b)
32     ax.set_xlabel('x-axis')
33     ax.set_ylabel('y-axis')
34     ax.set_zlabel('z-axis')
35     ax.set_xlim(-4, +4)
36     ax.set_ylim(-4, +4)
37     ax.set_zlim(-4, +4)
38     ax.view_init(elev=90, azim=-120)
39
40 plt.show()
```

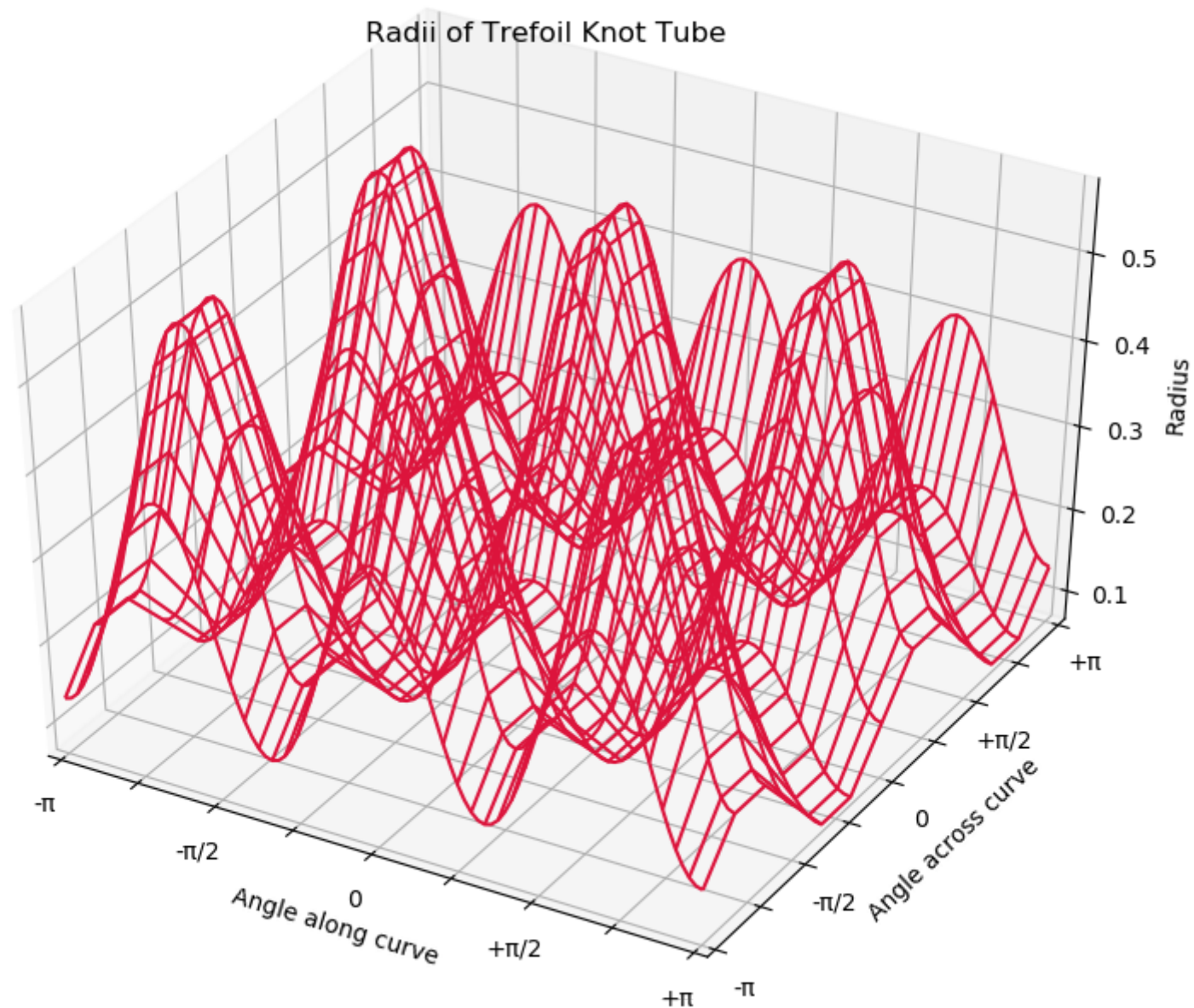

Trefoil Knot Tube with constant radius



```
In [29]: 1 # Calculate all the radii for the tube along and across the curve
          2
          3 rr = (4 + 2 * np.sin(2 * angles_across)) * (6 + 3 * np.cos(3 * angles_along)) / 90
          4
          5 assert rr.shape == surface_shape
```

In [30]:

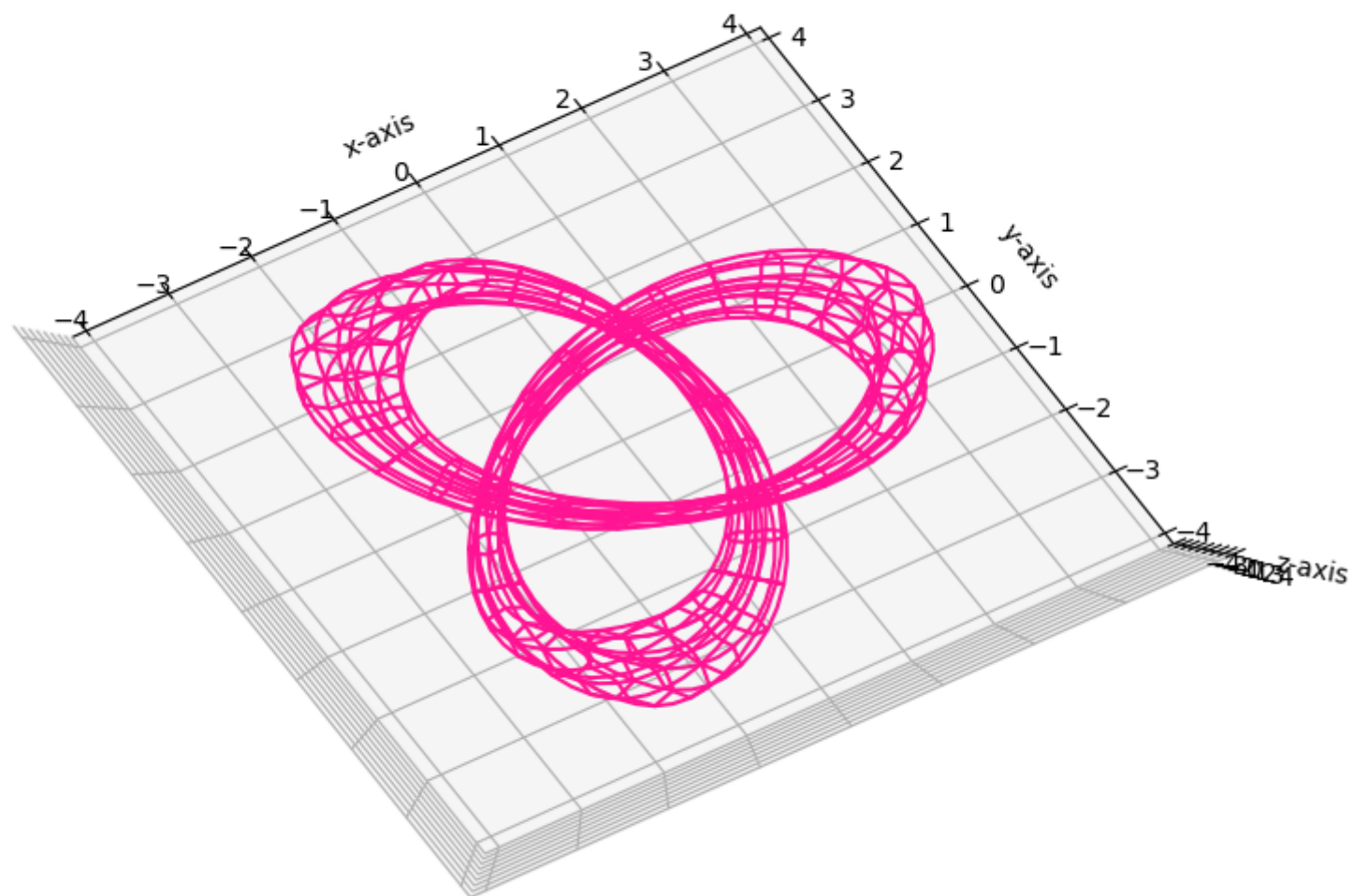
```
1  # Show the varying radii calculated above
2
3  fig = plt.figure(figsize=figure_size, dpi=figure_dpi)
4  ax = Axes3D(fig)
5  ax.set_title('Radii of Trefoil Knot Tube')
6  ax.plot_wireframe(angles_along, angles_across, rr, color='crimson')
7  ax.set_xlabel('Angle along curve')
8  ax.set_ylabel('Angle across curve')
9  ax.set_zlabel('Radius')
10 ax.set_xlim(-np.pi, +np.pi)
11 ax.set_ylim(-np.pi, +np.pi)
12 ax.set_xticklabels(pi_labels)
13 ax.set_yticklabels(pi_labels)
14 ax.set_xticks(pi_ticks)
15 ax.set_yticks(pi_ticks)
16 ax.view_init(elev=40, azimuth=-60)
17 plt.show()
```



```
In [31]: 1 # Calculate all the position vectors for the points on the surface of the tube
          2
          3 pp_s = pp_o + vv_s * rr
```

```
In [32]: 1 # Show the trefoil knot tube
          2
          3 fig = plt.figure(figsize=figure_size, dpi=figure_dpi)
          4 ax = Axes3D(fig)
          5 ax.set_title('Trefoil Knot Tube with varying radius')
          6 ax.plot_wireframe(*pp_s, color='deeppink')
          7 ax.set_xlabel('x-axis')
          8 ax.set_ylabel('y-axis')
          9 ax.set_zlabel('z-axis')
         10 ax.set_xlim(-4, +4)
         11 ax.set_ylim(-4, +4)
         12 ax.set_zlim(-4, +4)
         13 ax.view_init(elev=90, azimuth=-120)
         14 plt.show()
```

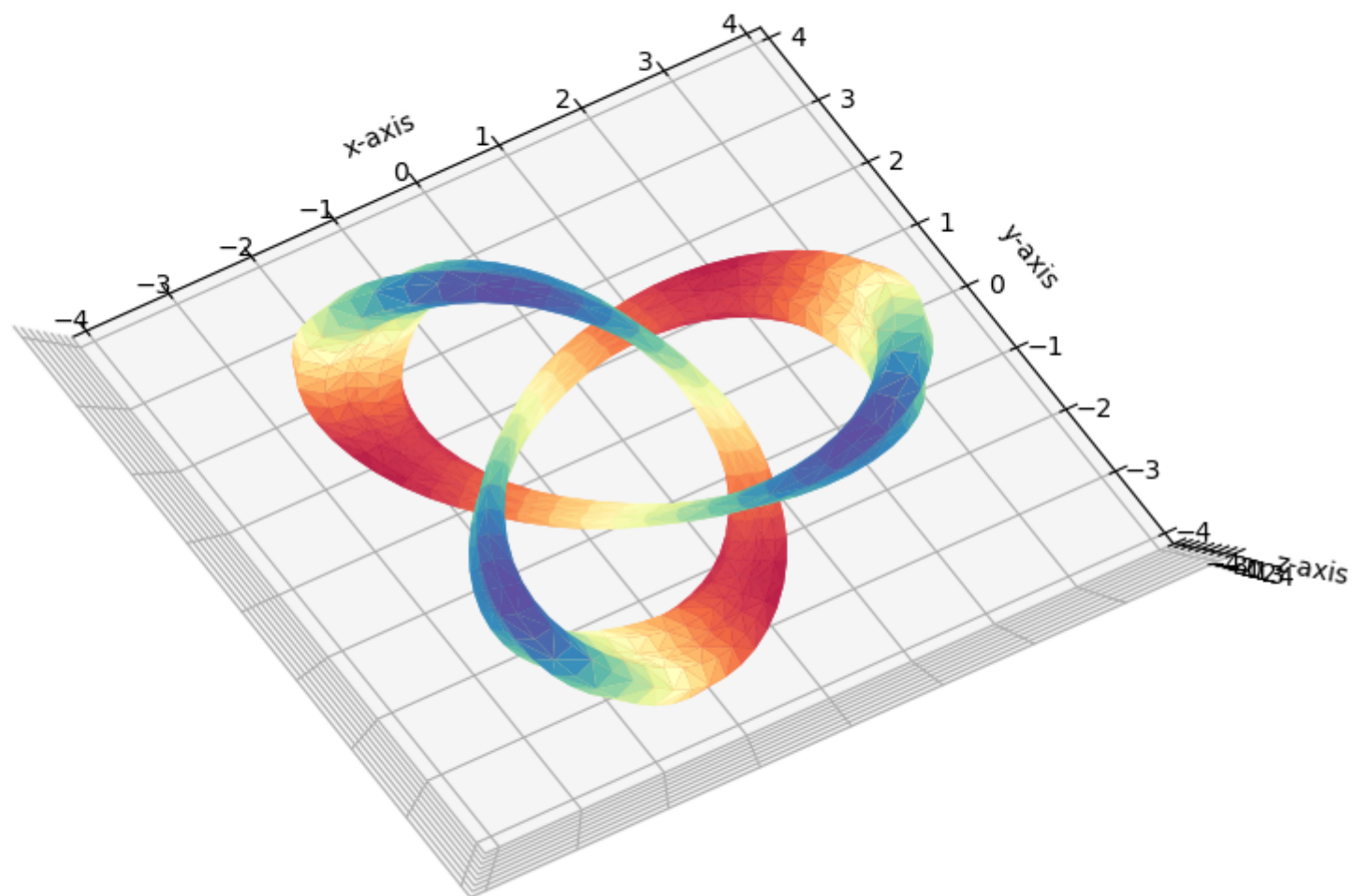
Trefoil Knot Tube with varying radius



In [33]:

```
1  # Show the trefoil knot tube
2
3  tri = \
4      mtri.Triangulation(
5          angles_along.flatten(),
6          angles_across.flatten()
7      )
8
9  fig = plt.figure(figsize=figure_size, dpi=figure_dpi)
10 ax = Axes3D(fig)
11 ax.set_title('Trefoil Knot Tube with varying radius')
12 ax.plot_trisurf(
13     pp_s.xx.flatten(),
14     pp_s.yy.flatten(),
15     pp_s.zz.flatten(),
16     triangles = tri.triangles,
17     cmap = plt.cm.Spectral
18 )
19 ax.set_xlabel('x-axis')
20 ax.set_ylabel('y-axis')
21 ax.set_zlabel('z-axis')
22 ax.set_xlim(-4, +4)
23 ax.set_ylim(-4, +4)
24 ax.set_zlim(-4, +4)
25 ax.view_init(elev=90, azimuth=-120)
26 plt.show()
```

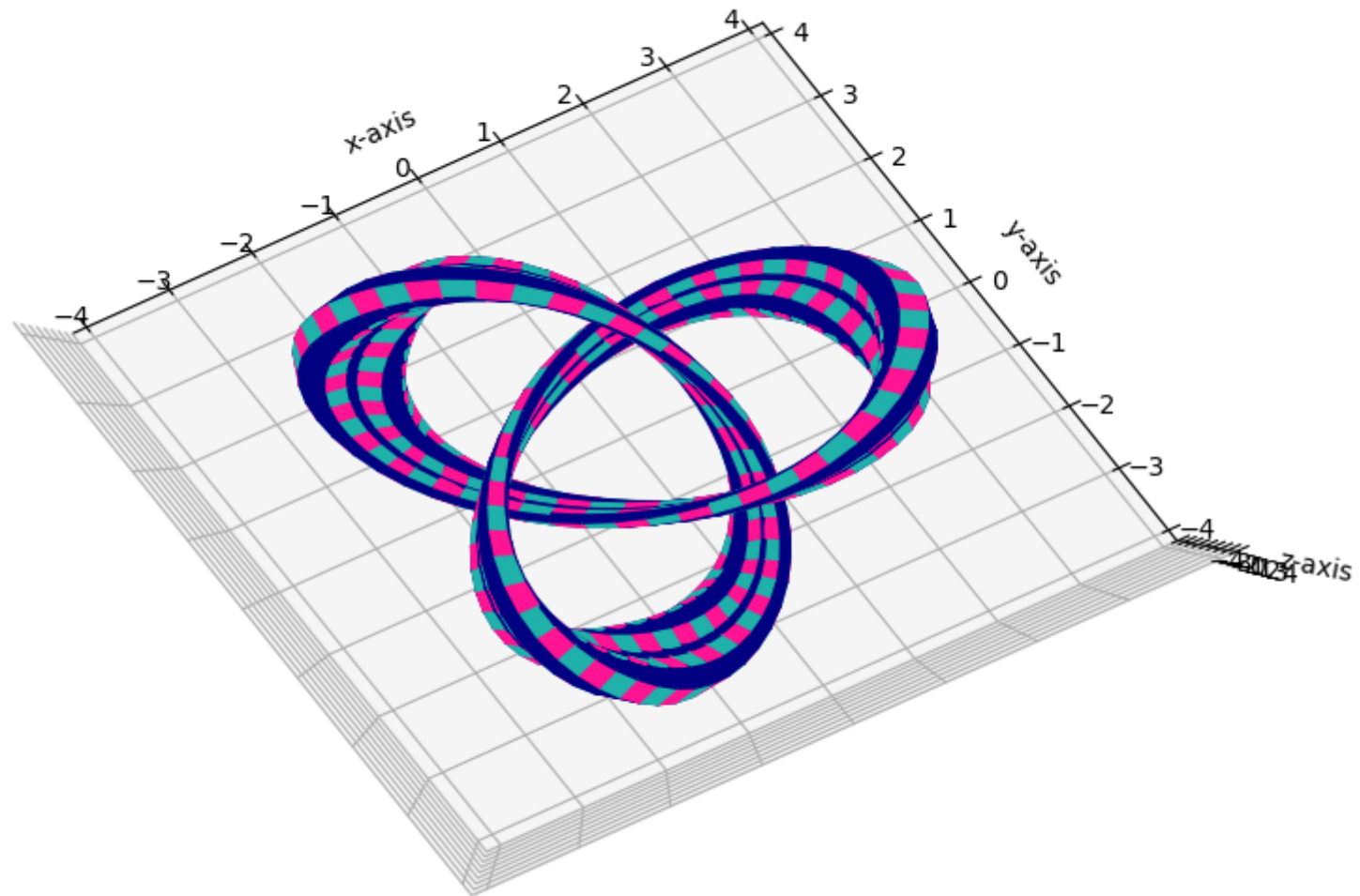
Trefoil Knot Tube with varying radius



In [34]:

```
1  # Show the trefoil knot tube
2
3  fig = plt.figure(figsize=figure_size, dpi=figure_dpi)
4  ax = Axes3D(fig)
5  ax.set_title('Trefoil Knot Tube with varying radius')
6  for j in range(nr_of_points_along_curve-1):
7      for i in range(nr_of_points_across_curve-1):
8          if i % 2 == 0:
9              # if (i + j) % 2 == 0:
10                 color = 'navy'
11             else:
12                 if j % 2 == 0:
13                     color = 'lightseagreen'
14                 else:
15                     color = 'deeppink'
16             p00 = pp_s(lambda cv: cv[i , j ])
17             p01 = pp_s(lambda cv: cv[i , j+1])
18             p10 = pp_s(lambda cv: cv[i+1, j ])
19             p11 = pp_s(lambda cv: cv[i+1, j+1])
20             triangle_a = Poly3DCollection([ [ p00, p10, p11 ] ])
21             triangle_a.set_color(color)
22             triangle_a.set_edgecolor(color)
23             ax.add_collection3d(triangle_a)
24             triangle_b = Poly3DCollection([ [ p11, p01, p00 ] ])
25             triangle_b.set_color(color)
26             triangle_b.set_edgecolor(color)
27             ax.add_collection3d(triangle_b)
28 ax.set_xlabel('x-axis')
29 ax.set_ylabel('y-axis')
30 ax.set_zlabel('z-axis')
31 ax.set_xlim(-4, +4)
32 ax.set_ylim(-4, +4)
33 ax.set_zlim(-4, +4)
34 ax.view_init(elev=90, azim=-120)
35 plt.show()
```


Trefoil Knot Tube with varying radius



In []:

1