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]:
```python
url = 'https://github.com/t-o-k/scikit-vectors_examples/'
```

In [2]:
```python
# This example has been tested with NumPy v1.15.3, Matplotlib v2.1.1 and Jupyter v4.4.0
```

In [3]:
```python
# Uncomment one of these to get a Matplotlib backend with interactive plots

# %matplotlib auto
# %matplotlib notebook
```

In [4]:
```python
# Get the necessary libraries

import matplotlib.colors as colors
import matplotlib.pyplot as plt
import matplotlib.tri as mtri
from mpl_toolkits.mplot3d import Axes3D
from mpl_toolkits.mplot3d.art3d import Poly3DCollection
import numpy as np

from skvectors import create_class_Cartesian_3D_Vector
```

In [5]:
```python
# Size and resolution for Matplotlib figures

figure_size = (8, 6)
figure_dpi = 100
```

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

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

```python
# The second derivatives of the functions for the curve

def d2_f_x(t):

    return -np.cos(t) - 8 * np.cos(2 * t)


def d2_f_y(t):

    return -np.sin(t) + 8 * np.sin(2 * t)


def d2_f_z(t):

    return -18 * np.sin(3 * t)
```

```python
# Resolutions for plot

nr_of_points_along_curve = 3 * 2**5 + 1
nr_of_points_across_curve = 3 * 2**2 + 1
```

```python
In [10]:   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      }
```

```python
In [11]:   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      )
```

```python
In [12]: 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      )
```

```python
In [13]: 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      )
```

```python
In [14]: 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      )
```

```python
In [15]: 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)
```

```python
In [16]:  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 [17]:   1  # Show the trefoil knot tube
           2
           3  fig = plt.figure(figsize=figure_size, dpi=figure_dpi)
           4  fig.text(0.01, 0.01, url)
           5  ax = Axes3D(fig)
           6  ax.set_aspect(1)
           7  ax.set_title('Trefoil Knot Tube')
           8  for j in range(nr_of_points_along_curve-1):
           9      for i in range(nr_of_points_across_curve-1):
          10          if i % 2 == 0:
          11              facecolor = 'black'
          12          else:
          13              if j % 2 == 0:
          14                  facecolor = 'blue'
          15              else:
          16                  facecolor = 'mediumturquoise'
          17          x0, y0, z0 = surface_points[i  ]
          18          x1, y1, z1 = surface_points[i+1]
          19          p00 = (x0[j  ], y0[j  ], z0[j  ])
          20          p01 = (x0[j+1], y0[j+1], z0[j+1])
          21          p10 = (x1[j  ], y1[j  ], z1[j  ])
          22          p11 = (x1[j+1], y1[j+1], z1[j+1])
          23          triangle_a = Poly3DCollection([ [ p00, p10, p11 ] ])
          24          triangle_a.set_facecolor(facecolor)
          25          triangle_a.set_edgecolor('grey')
          26          ax.add_collection3d(triangle_a)
          27          triangle_b = Poly3DCollection([ [ p11, p01, p00 ] ])
          28          triangle_b.set_color(facecolor)
          29          triangle_b.set_edgecolor('grey')
          30          ax.add_collection3d(triangle_b)
          31  ax.set_xlim(-3.5, +3.5)
          32  ax.set_ylim(-3.5, +3.5)
          33  ax.set_zlim(-3.5, +3.5)
          34  ax.set_xlabel('x-axis')
          35  ax.set_ylabel('y-axis')
          36  ax.set_zlabel('z-axis')
          37  ax.view_init(elev=90, azim=-120)
          38
          39  plt.show()
```
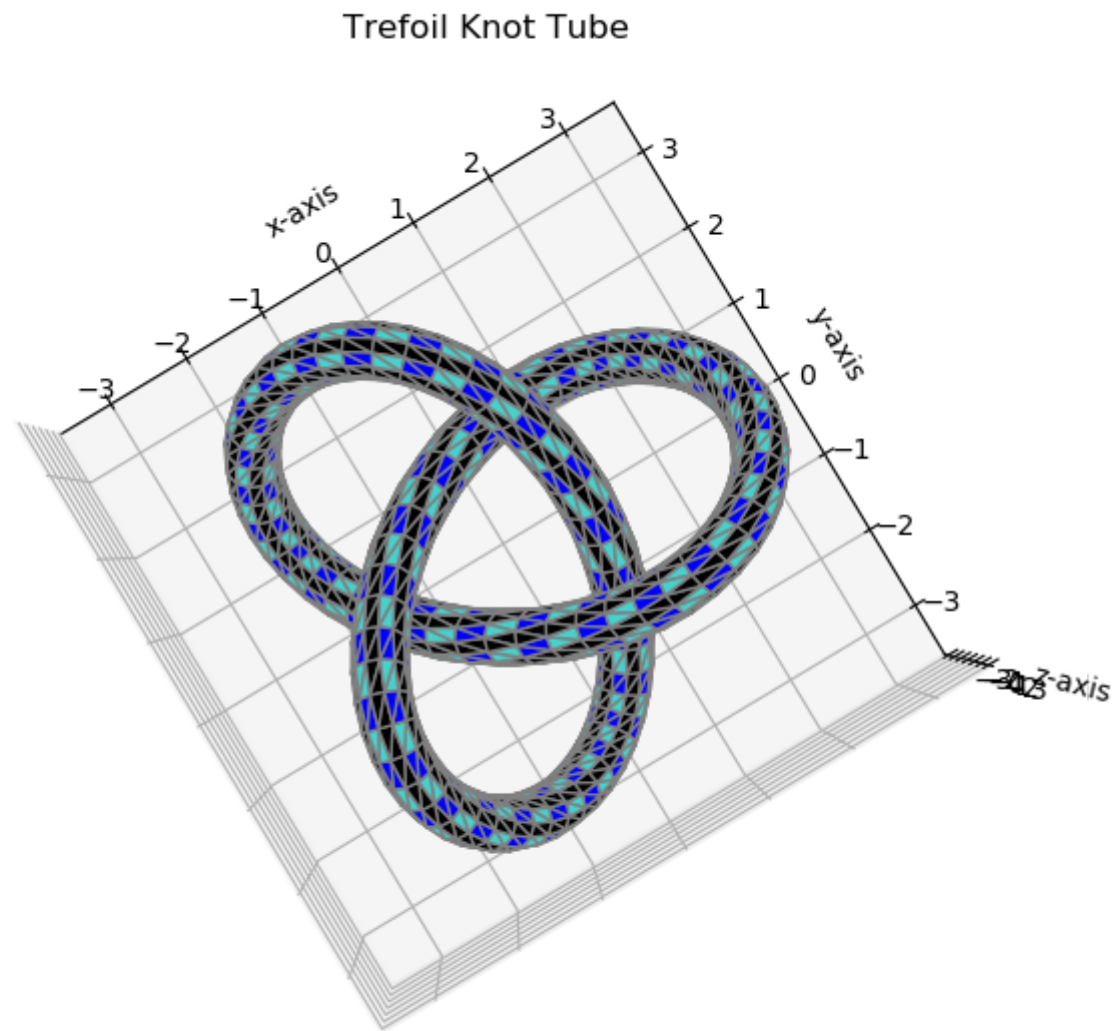
Trefoil Knot Tube

https://github.com/t-o-k/scikit-vectors_examples/

```python
In [19]:    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      )
```

```python
In [21]:    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
```

```python
In [21]:    1  # Initialize position vectors for the points
            2  # (The 1D arrays are beeing 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 [22]:   1  # Alternative ways to do the same
           2
           3  # pp_o = NP_3D_A2(*p_o)
           4
           5  # pp_o = \
           6  #     NP_3D_A2(
           7  #         xx = A2_cunit * p_o.x,
           8  #         yy = A2_cunit * p_o.y,
           9  #         zz = A2_cunit * p_o.z
          10  #     )
          11
          12  # tile_size = (nr_of_points_across_curve, 1)
          13  # pp_o = \
          14  #     NP_3D_A2(
          15  #         xx = np.tile(p_o.x, tile_size),
          16  #         yy = np.tile(p_o.y, tile_size),
          17  #         zz = np.tile(p_o.z, tile_size)
          18  #     )
```

```
In [23]:   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 [24]:   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)
```
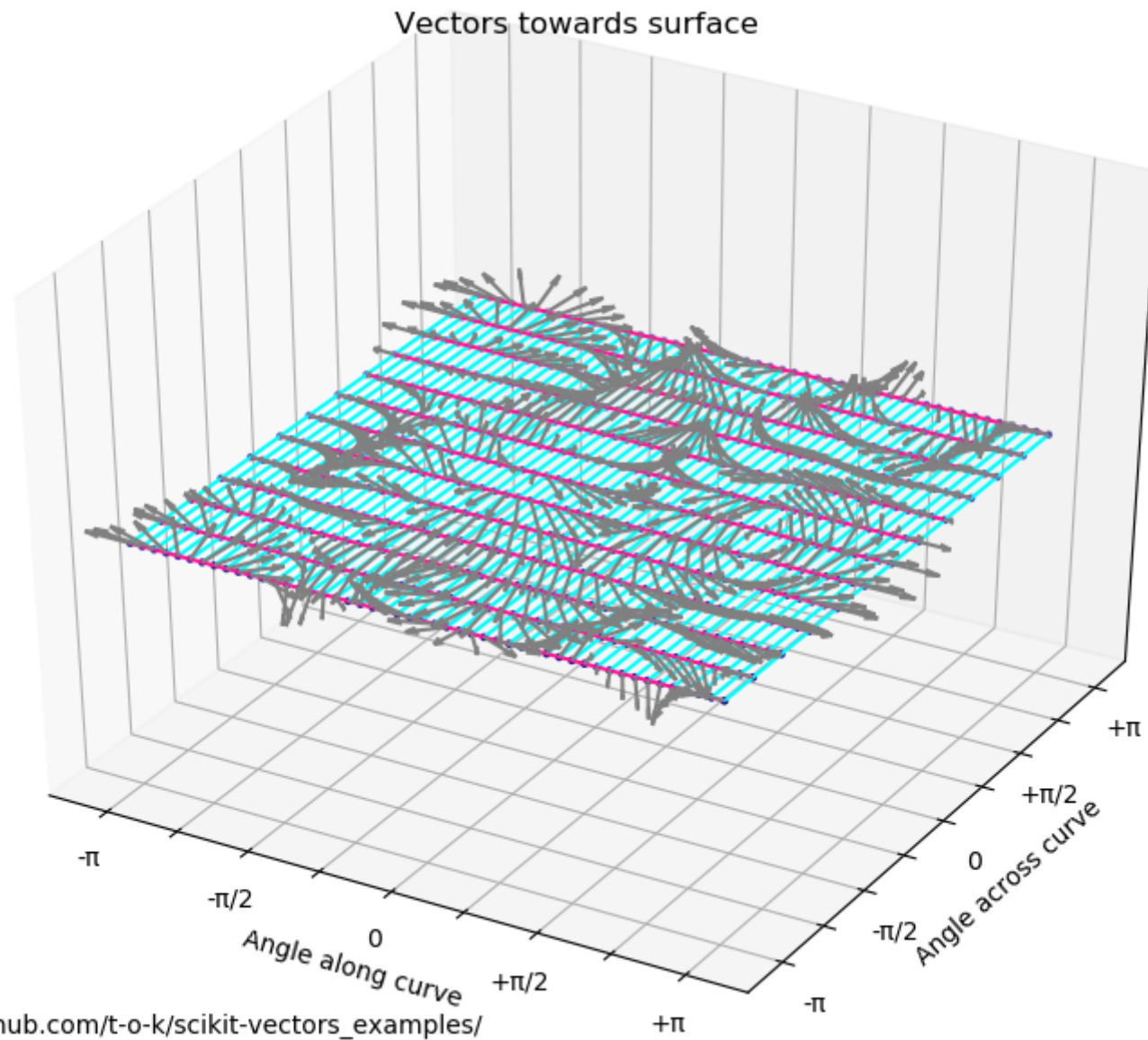
```
In [25]:   1  # Calculate all the vectors along and across the curve towards the surface of the tube
           2
           3  vv_s = vv_n.axis_rotate(vv_t, angles_across)
```

```python
# Prepare some variables for plotting

no_labels = [ ]
no_ticks = [ ]

pi_labels = [ '-π', '', '-π/2', '', '0', '', '+π/2', '', '+π' ]
pi_ticks = \
    [
        n / 4 * np.pi
        for n in [ -4, -3, -2, -1, 0, +1, +2, +3, +4 ]
    ]

vector_length = 0.5

stride_along = 2
stride_across = 1

sl_along = slice(None, None, stride_along)
sl_across = slice(None, None, stride_across)
sl = (sl_across, sl_along)
```
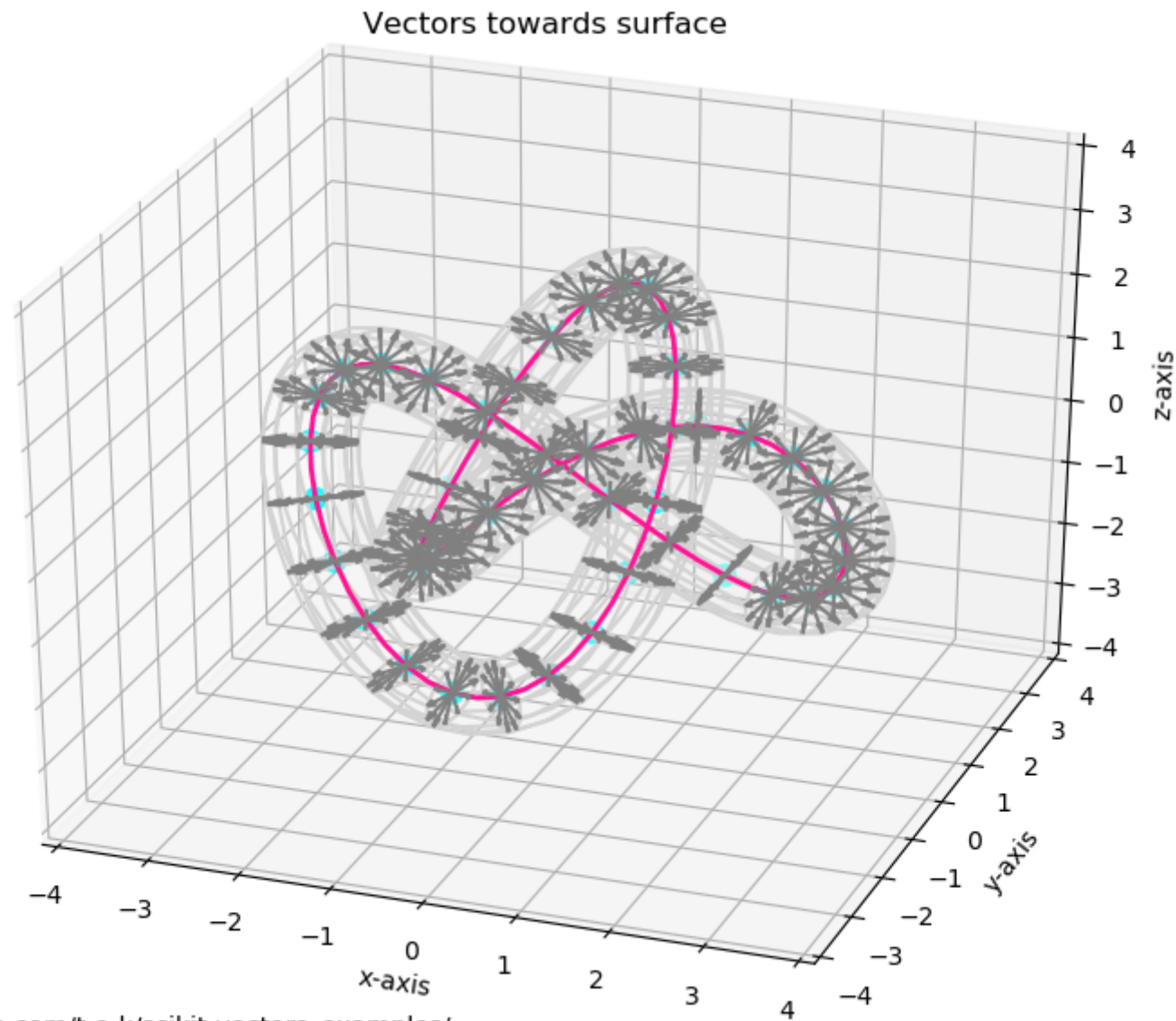
```
In [27]:  1  # Show some of the vectors calculated above
          2
          3  fig = plt.figure(figsize=figure_size, dpi=figure_dpi)
          4  fig.text(0.01, 0.01, url)
          5  ax = Axes3D(fig)
          6  ax.set_title('Vectors towards surface')
          7  ax.scatter(
          8      angles_along[sl], angles_across[sl], zeros[sl],
          9      color = 'darkblue',
         10      marker = '.',
         11      edgecolors = 'face'
         12  )
         13  ax.plot_wireframe(
         14      angles_along, angles_across, zeros,
         15      rstride = 0,
         16      cstride = stride_along,
         17      color = 'cyan'
         18  )
         19  ax.plot_wireframe(
         20      angles_along, angles_across, zeros,
         21      rstride = stride_across,
         22      cstride = 0,
         23      color = 'deeppink'
         24  )
         25  ax.quiver(
         26      angles_along[sl], angles_across[sl], zeros[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.set_xlim(-np.pi-0.5, +np.pi+0.5)
         33  ax.set_ylim(-np.pi-0.5, +np.pi+0.5)
         34  ax.set_zlim(-np.pi-0.5, +np.pi+0.5)
         35  ax.set_xlabel('Angle along curve')
         36  ax.set_ylabel('Angle across curve')
         37  ax.set_xticklabels(pi_labels)
         38  ax.set_yticklabels(pi_labels)
         39  ax.set_zticklabels(no_labels)
         40  ax.set_xticks(pi_ticks)
         41  ax.set_yticks(pi_ticks)
         42  ax.set_zticks(no_ticks)
         43  ax.view_init(elev=36, azim=-60)
```

Vectors towards surface

https://github.com/t-o-k/scikit-vectors_examples/

```python
# Show some of the vectors calculated above

pp_w = pp_o + vv_s *vector_length

fig = plt.figure(figsize=figure_size, dpi=figure_dpi)
fig.text(0.01, 0.01, url)
ax = Axes3D(fig)
ax.set_title('Vectors towards surface')
ax.plot_wireframe(
    pp_w.xx, pp_w.yy, pp_w.zz,
    rstride = 0,
    cstride = stride_along,
    color = 'lightgray'
)
ax.plot(
    p_o.x, p_o.y, p_o.z,
    color = 'deeppink',
    linewidth = 2
)
ax.plot_wireframe(
    pp_w.xx, pp_w.yy, pp_w.zz,
    rstride = stride_across,
    cstride = 0,
    color = 'lightgray'
)
ax.quiver(
    pp_o.xx[sl], pp_o.yy[sl], pp_o.zz[sl],
    vv_s.xx[sl], vv_s.yy[sl], vv_s.zz[sl],
    length = vector_length,
    pivot = 'tail',
    color = 'gray'
)
ax.scatter(
    p_o.x[sl_along], p_o.y[sl_along], p_o.z[sl_along],
    color = 'cyan',
    marker = 'o',
    linewidth = 5
)
ax.set_xlabel('x-axis')
ax.set_ylabel('y-axis')
ax.set_zlabel('z-axis')
ax.set_xlim(-4, +4)
ax.set_ylim(-4, +4)
```

```
44  ax.set_zlim(-4, +4)
45  ax.view_init(elev=30, azim=-70)
46  plt.show()
```
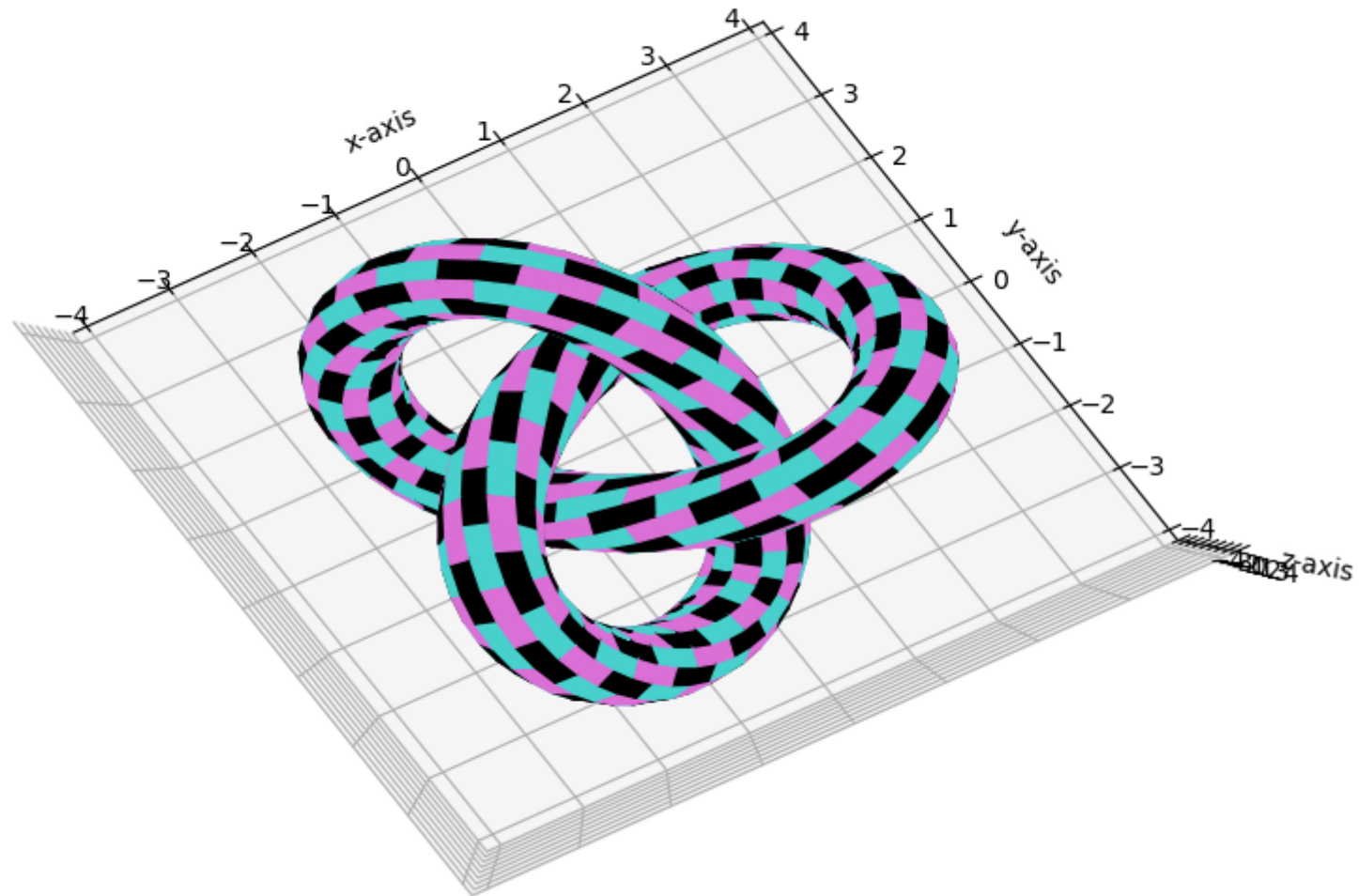

Vectors towards surface

```
In [29]:  1  # Show the trefoil knot tube
          2
          3  fig = plt.figure(figsize=figure_size, dpi=figure_dpi)
          4  fig.text(0.01, 0.01, url)
          5  ax = Axes3D(fig)
          6  ax.set_title('Trefoil Knot Tube with constant radius')
          7  for j in range(nr_of_points_along_curve-1):
          8      for i in range(nr_of_points_across_curve-1):
          9          k = (3 * i + j) % 6
         10  #         k = (2 * i + 3 * j) % 6
         11          if k < 2:
         12              color = 'mediumturquoise'
         13          elif k < 4:
         14              color = 'black'
         15          else:
         16              color = 'orchid'
         17          c00 = (i  , j  )
         18          c01 = (i  , j+1)
         19          c10 = (i+1, j  )
         20          c11 = (i+1, j+1)
         21          p00 = (pp_w.xx[c00], pp_w.yy[c00], pp_w.zz[c00])
         22          p01 = (pp_w.xx[c01], pp_w.yy[c01], pp_w.zz[c01])
         23          p10 = (pp_w.xx[c10], pp_w.yy[c10], pp_w.zz[c10])
         24          p11 = (pp_w.xx[c11], pp_w.yy[c11], pp_w.zz[c11])
         25          triangle_a = Poly3DCollection([ [ p00, p10, p11 ] ])
         26          triangle_a.set_color(color)
         27          ax.add_collection3d(triangle_a)
         28          triangle_b = Poly3DCollection([ [ p11, p01, p00 ] ])
         29          triangle_b.set_color(color)
         30          ax.add_collection3d(triangle_b)
         31  ax.set_xlabel('x-axis')
         32  ax.set_ylabel('y-axis')
         33  ax.set_zlabel('z-axis')
         34  ax.set_xlim(-4, +4)
         35  ax.set_ylim(-4, +4)
         36  ax.set_zlim(-4, +4)
         37  ax.view_init(elev=90, azim=-120)
         38
         39  plt.show()
```

Trefoil Knot Tube with constant radius

```
In [30]:  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
```
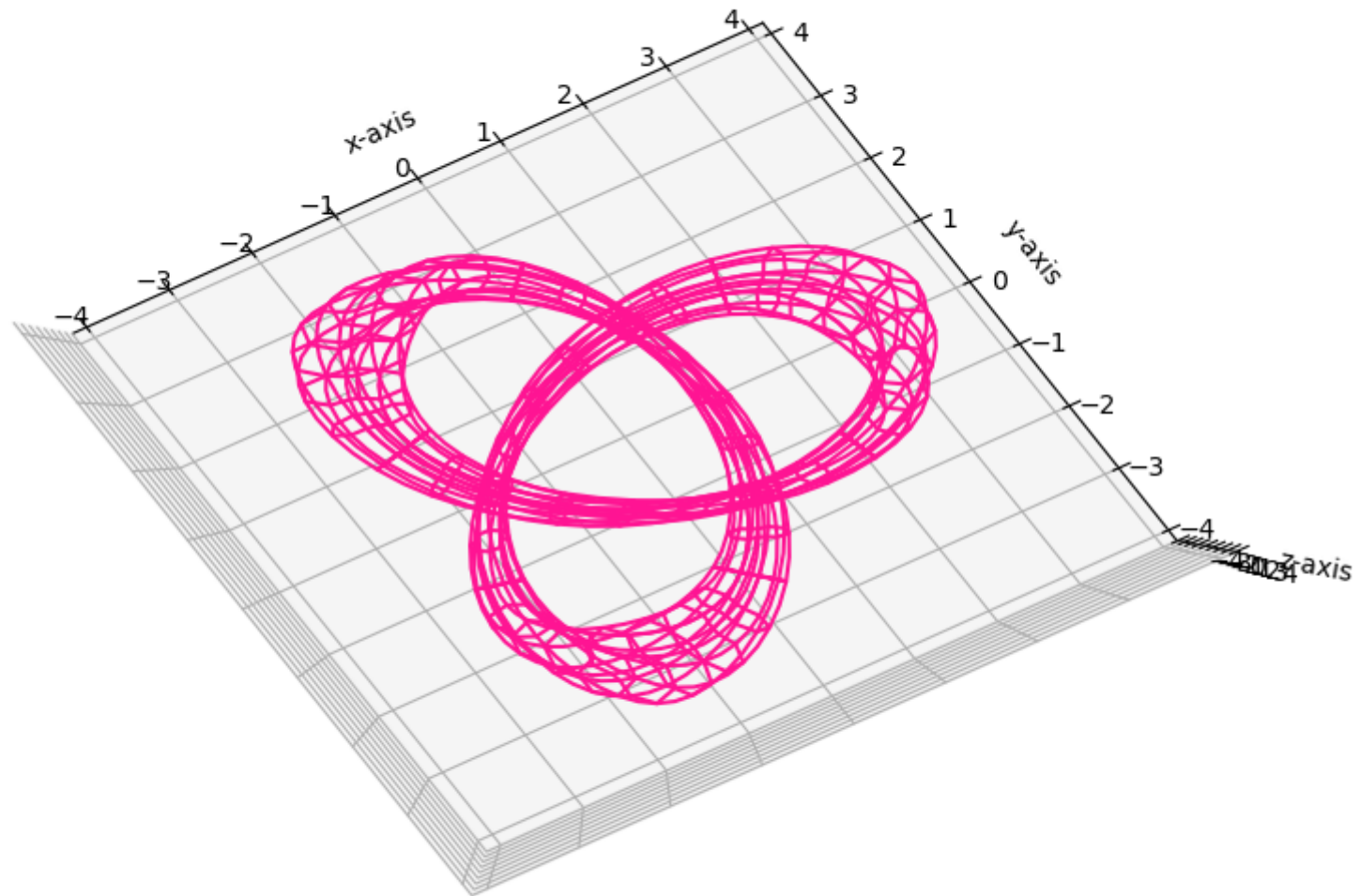
```
In [31]:   1  # Show the varying radii calculated above
           2
           3  fig = plt.figure(figsize=figure_size, dpi=figure_dpi)
           4  fig.text(0.01, 0.01, url)
           5  ax = Axes3D(fig)
           6  ax.set_title('Radii of Trefoil Knot Tube')
           7  ax.plot_wireframe(angles_along, angles_across, rr, color='crimson')
           8  ax.set_xlabel('Angle along curve')
           9  ax.set_ylabel('Angle across curve')
          10  ax.set_zlabel('Radius')
          11  ax.set_xlim(-np.pi, +np.pi)
          12  ax.set_ylim(-np.pi, +np.pi)
          13  ax.set_xticklabels(pi_labels)
          14  ax.set_yticklabels(pi_labels)
          15  ax.set_xticks(pi_ticks)
          16  ax.set_yticks(pi_ticks)
          17  ax.view_init(elev=40, azim=-60)
          18  plt.show()
```

Radii of Trefoil Knot Tube

Radius

0.5

0.4

0.3

0.2

0.1

+π

+π/2

0

-π/2 Angle across curve

-π

Angle along curve

-π/2

0

+π/2

+π

-π
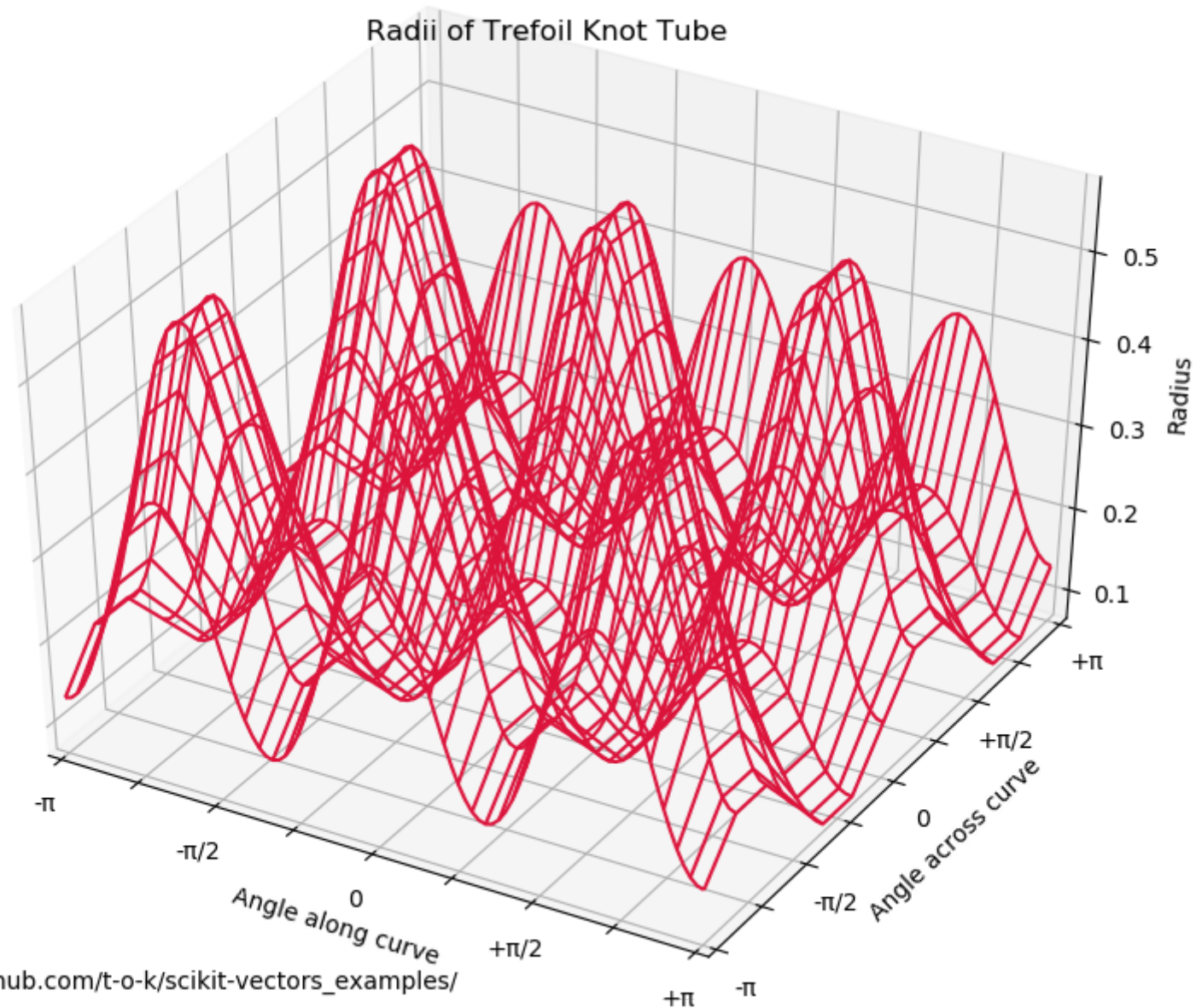
In [32]:
```
1  # Calculate all the position vectors for the points on the surface of the tube
2
3  pp_s = pp_o + vv_s * rr
```

```
In [33]:    1  # Show the trefoil knot tube
            2
            3  fig = plt.figure(figsize=figure_size, dpi=figure_dpi)
            4  fig.text(0.01, 0.01, url)
            5  ax = Axes3D(fig)
            6  ax.set_title('Trefoil Knot Tube with varying radius')
            7  ax.plot_wireframe(*pp_s, color='deeppink')
            8  ax.set_xlabel('x-axis')
            9  ax.set_ylabel('y-axis')
           10  ax.set_zlabel('z-axis')
           11  ax.set_xlim(-4, +4)
           12  ax.set_ylim(-4, +4)
           13  ax.set_zlim(-4, +4)
           14  ax.view_init(elev=90, azim=-120)
           15  plt.show()
```
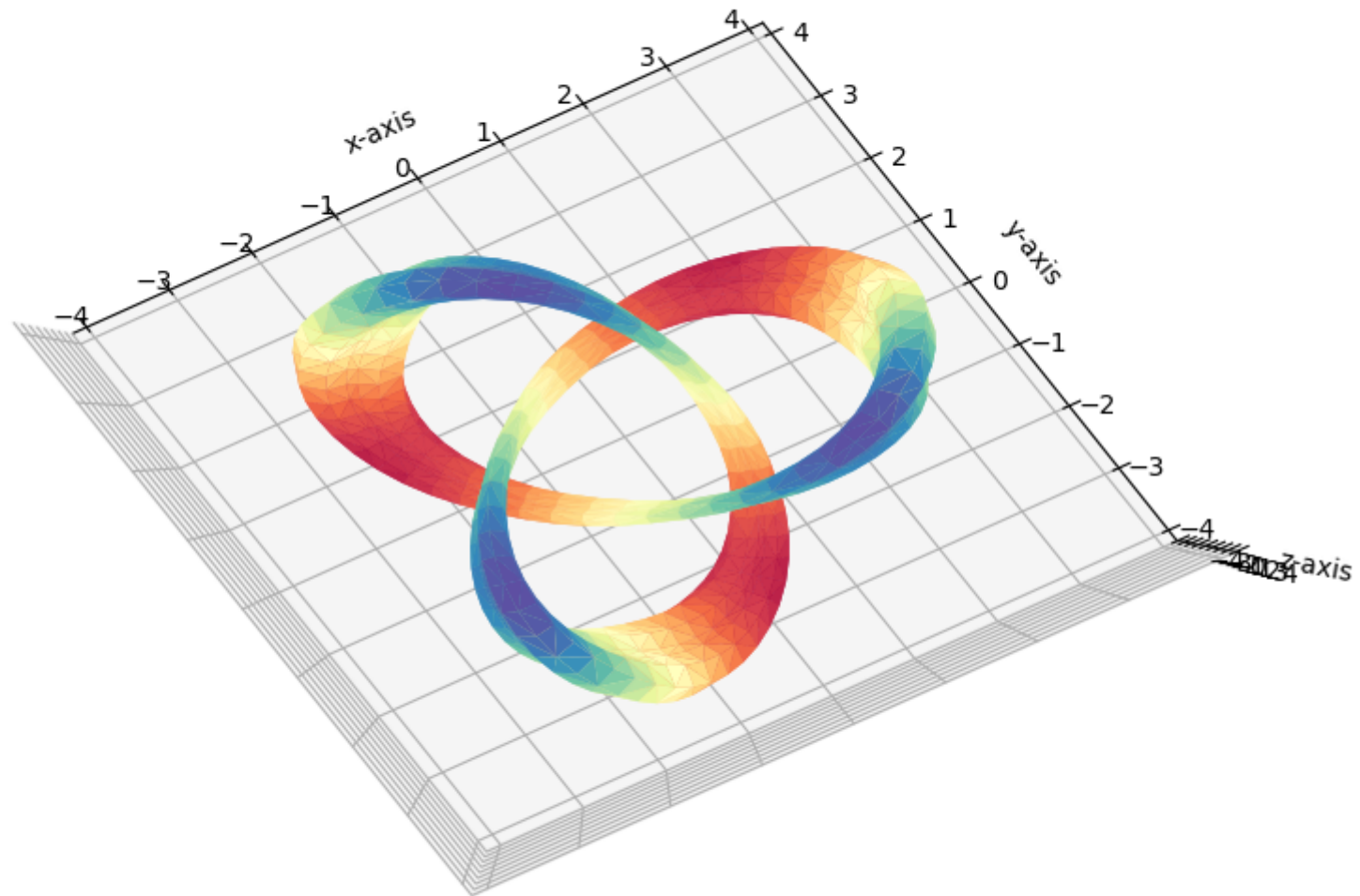
Trefoil Knot Tube with varying radius

```python
# Show the trefoil knot tube

tri = \
    mtri.Triangulation(
        angles_along.flatten(),
        angles_across.flatten()
    )

fig = plt.figure(figsize=figure_size, dpi=figure_dpi)
fig.text(0.01, 0.01, url)
ax = Axes3D(fig)
ax.set_title('Trefoil Knot Tube with varying radius')
ax.plot_trisurf(
    pp_s.xx.flatten(),
    pp_s.yy.flatten(),
    pp_s.zz.flatten(),
    triangles = tri.triangles,
    cmap = plt.cm.Spectral
)
ax.set_xlabel('x-axis')
ax.set_ylabel('y-axis')
ax.set_zlabel('z-axis')
ax.set_xlim(-4, +4)
ax.set_ylim(-4, +4)
ax.set_zlim(-4, +4)
ax.view_init(elev=90, azim=-120)
plt.show()
```

Trefoil Knot Tube with varying radius

```
In [35]:    1  # Show the trefoil knot tube
            2
            3  fig = plt.figure(figsize=figure_size, dpi=figure_dpi)
            4  fig.text(0.01, 0.01, url)
            5  ax = Axes3D(fig)
            6  ax.set_title('Trefoil Knot Tube with varying radius')
            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  #          if (i + j) % 2 == 0:
           11              color = 'navy'
           12          else:
           13              if j % 2 == 0:
           14                  color = 'lightseagreen'
           15              else:
           16                  color = 'deeppink'
           17          p00 = pp_s(lambda cv: cv[i  , j  ])
           18          p01 = pp_s(lambda cv: cv[i  , j+1])
           19          p10 = pp_s(lambda cv: cv[i+1, j  ])
           20          p11 = pp_s(lambda cv: cv[i+1, j+1])
           21          triangle_a = Poly3DCollection([ [ p00, p10, p11 ] ])
           22          triangle_a.set_color(color)
           23          ax.add_collection3d(triangle_a)
           24          triangle_b = Poly3DCollection([ [ p11, p01, p00 ] ])
           25          triangle_b.set_color(color)
           26          ax.add_collection3d(triangle_b)
           27  ax.set_xlabel('x-axis')
           28  ax.set_ylabel('y-axis')
           29  ax.set_zlabel('z-axis')
           30  ax.set_xlim(-4, +4)
           31  ax.set_ylim(-4, +4)
           32  ax.set_zlim(-4, +4)
           33  ax.view_init(elev=90, azim=-120)
           34  plt.show()
```
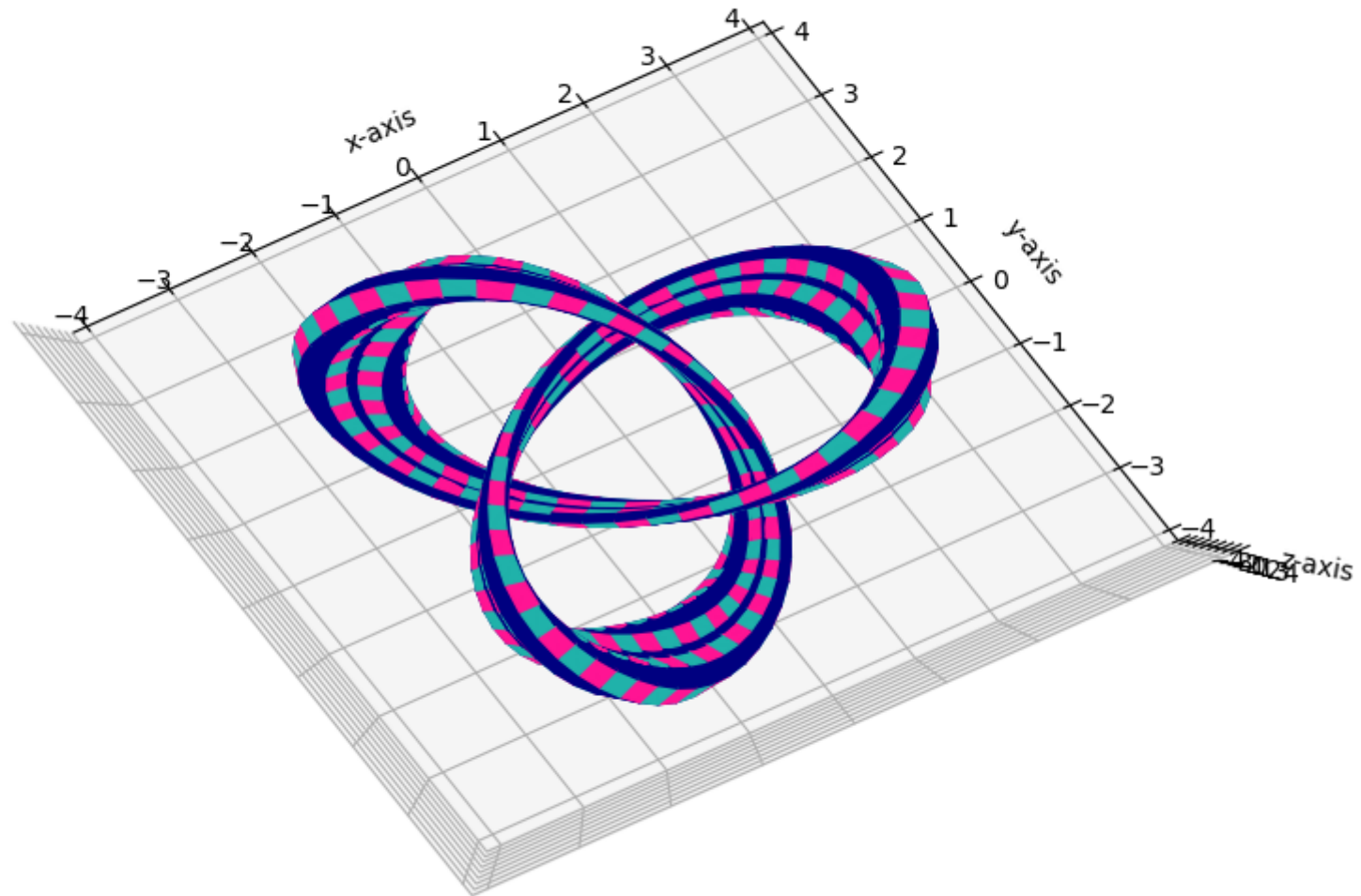
Trefoil Knot Tube with varying radius

https://github.com/t-o-k/scikit-vectors_examples/

In [ ]: 1