Offset Curves along a parametric curve

- using Matplotlib, NumPy and scikit-vectors

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

In [1]:
```
url = 'https://github.com/t-o-k/scikit-vectors_examples/'
```

In [2]:
```
# This example has been tested with NumPy v1.13.3, Matplotlib v2.1.1 and Jupyter v4.4.0.
```

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

# %matplotlib auto
# %matplotlib notebook
```

In [4]:
```
import matplotlib.pyplot as plt
from matplotlib.patches import Polygon
from matplotlib.collections import PatchCollection
import numpy as np

from skvectors import create_class_Cartesian_2D_Vector
```

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

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

In [6]:

```
# The functions for a parametric "flower" curve

def f_x(t):

    return 0.5 + 0.46 * (3 * np.cos(5 * t - np.pi / 6) + 2 * np.sin(3 * t)) / 5


def f_y(t):

    return 0.5 + 0.46 * (3 * np.sin(5 * t - np.pi / 6) + 2 * np.cos(3 * t)) / 5
```

In [7]:

```
# Numerical approximation of the first derivative of a univariate function

def first_derivative(fn, h=1e-4):

    h2 = 2 * h


    def d1_fn(t):

        return (fn(t + h) - fn(t - h)) / h2


    return d1_fn
```

In [8]:

```
# Create derivative functions for the curve

d1_f_x = first_derivative(f_x)
d1_f_y = first_derivative(f_y)
```

In [9]:

```
no_of_points_along_curve = 800
```

In [10]:

```
# Necessary NumPy functions

np_functions = \
    {
        'not': np.logical_not,
        'and': np.logical_and,
        'or': np.logical_or,
        'all': np.all,
        'any': np.any,
        'min': np.minimum,
        'max': np.maximum,
        'abs': np.absolute,
        'trunc': np.trunc,
        'ceil': np.ceil,
        'copysign': np.copysign,
        'log10': np.log10,
        'cos': np.cos,
        'sin': np.sin,
        'atan2': np.arctan2,
        'pi': np.pi
    }
```

In [11]:

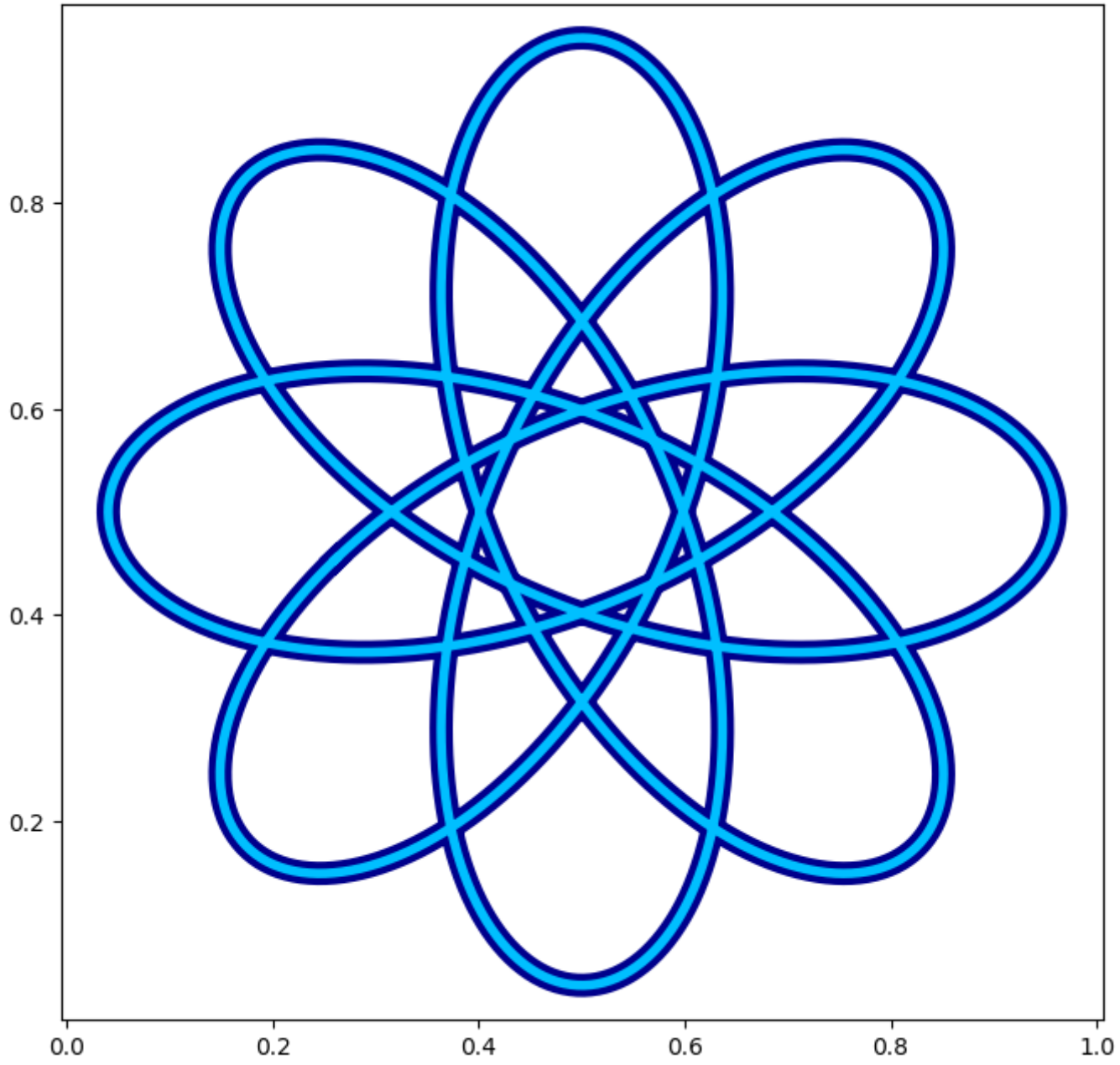
```
# Create a vector class that can hold all the points along the curve

NP2 = \
    create_class_Cartesian_2D_Vector(
        name = 'NP2',
        component_names = 'xy',
        brackets = '<>',
        sep = ', ',
        cnull = np.zeros(no_of_points_along_curve),
        cunit = np.ones(no_of_points_along_curve),
        functions = np_functions
    )
```

In [12]:

```
# Calculate the points along the curve

angles_along_curve = np.linspace(-np.pi, +np.pi, no_of_points_along_curve, endpoint=True)

p_o = \
    NP2(
        x = f_x(angles_along_curve),
        y = f_y(angles_along_curve)
    )
```

In [13]:

```
# Show the curve by drawing a line above a thicker line

fig, ax = plt.subplots(figsize=figure_size, dpi=figure_dpi)
fig.text(0.30, 0.05, url)
ax.plot(p_o.x, p_o.y, color='darkblue', linewidth=10)
ax.plot(p_o.x, p_o.y, color='deepskyblue', linewidth=4)
ax.axis('equal')
plt.show()
```



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

In [14]:

```
# Calculate vectors from the first derivatives at the points along the curve

v_d1 = \
    NP2(
        x = d1_f_x(angles_along_curve),
        y = d1_f_y(angles_along_curve)
    )
```

In [15]:

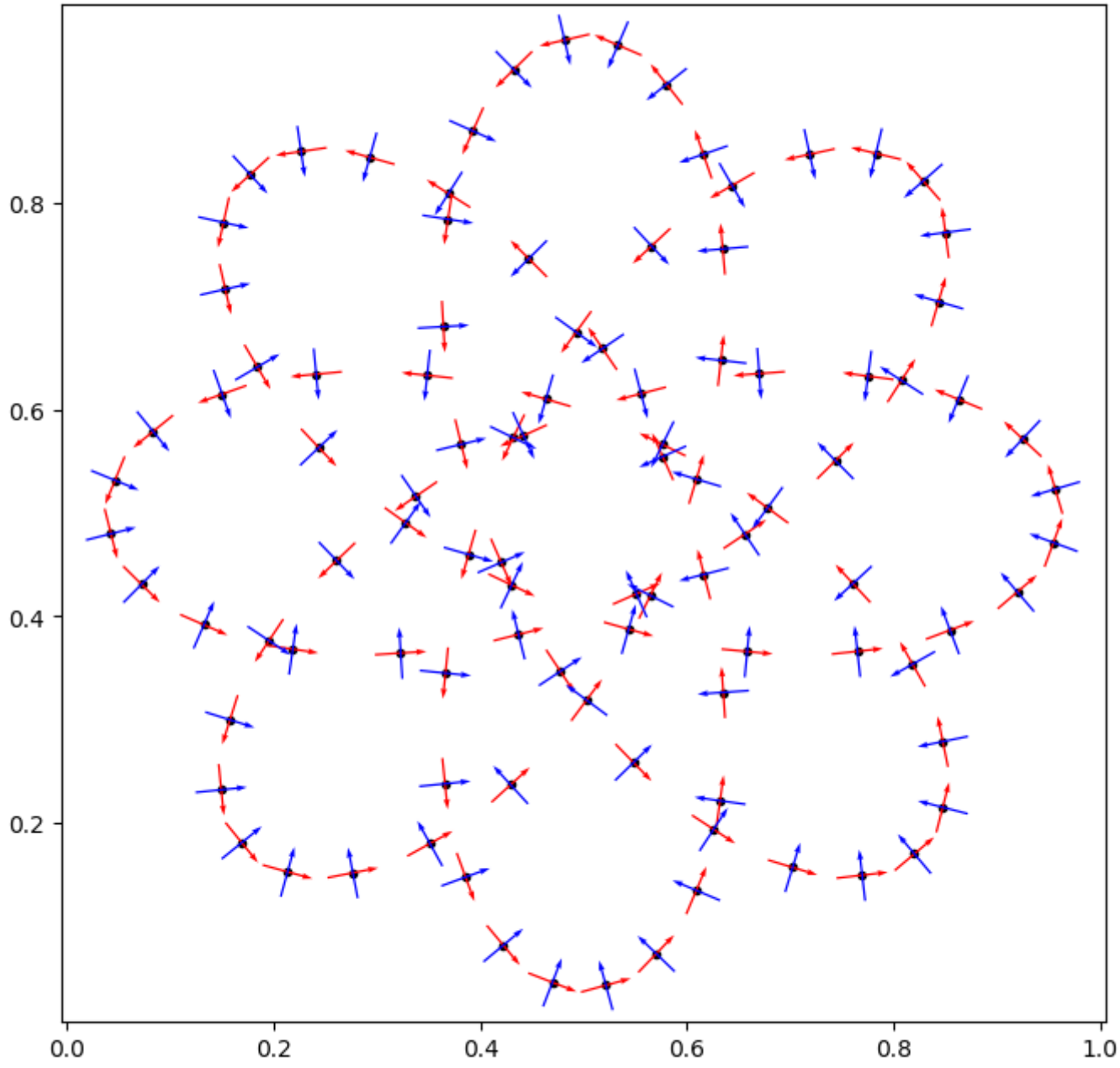
```
# Calculate tangent vectors at the points along the curve

v_t = v_d1.normalize()
```

In [16]:

```
# Calculate normal vectors at the points along the curve

v_n = v_t.perp()
```

In [17]:

```
# Show some of the tangent vectors and the normal vectors along the curve

s = 8  # stride

fig, ax = plt.subplots(figsize=figure_size, dpi=figure_dpi)
fig.text(0.30, 0.05, url)
ax.scatter(
    p_o.x[::s], p_o.y[::s],
    color = 'black',
    marker = '.'
)
ax.quiver(
    p_o.x[::s], p_o.y[::s],
    v_t.x[::s], v_t.y[::s],
    width = 0.002,
    color = 'red',
    scale = 20,
    scale_units = 'xy',
    pivot = 'middle'
)
ax.quiver(
    p_o.x[::s], p_o.y[::s],
    v_n.x[::s], v_n.y[::s],
    width = 0.002,
    color = 'blue',
    scale = 20,
    scale_units = 'xy',
    pivot = 'middle'
)
ax.axis('equal')
plt.show()
```



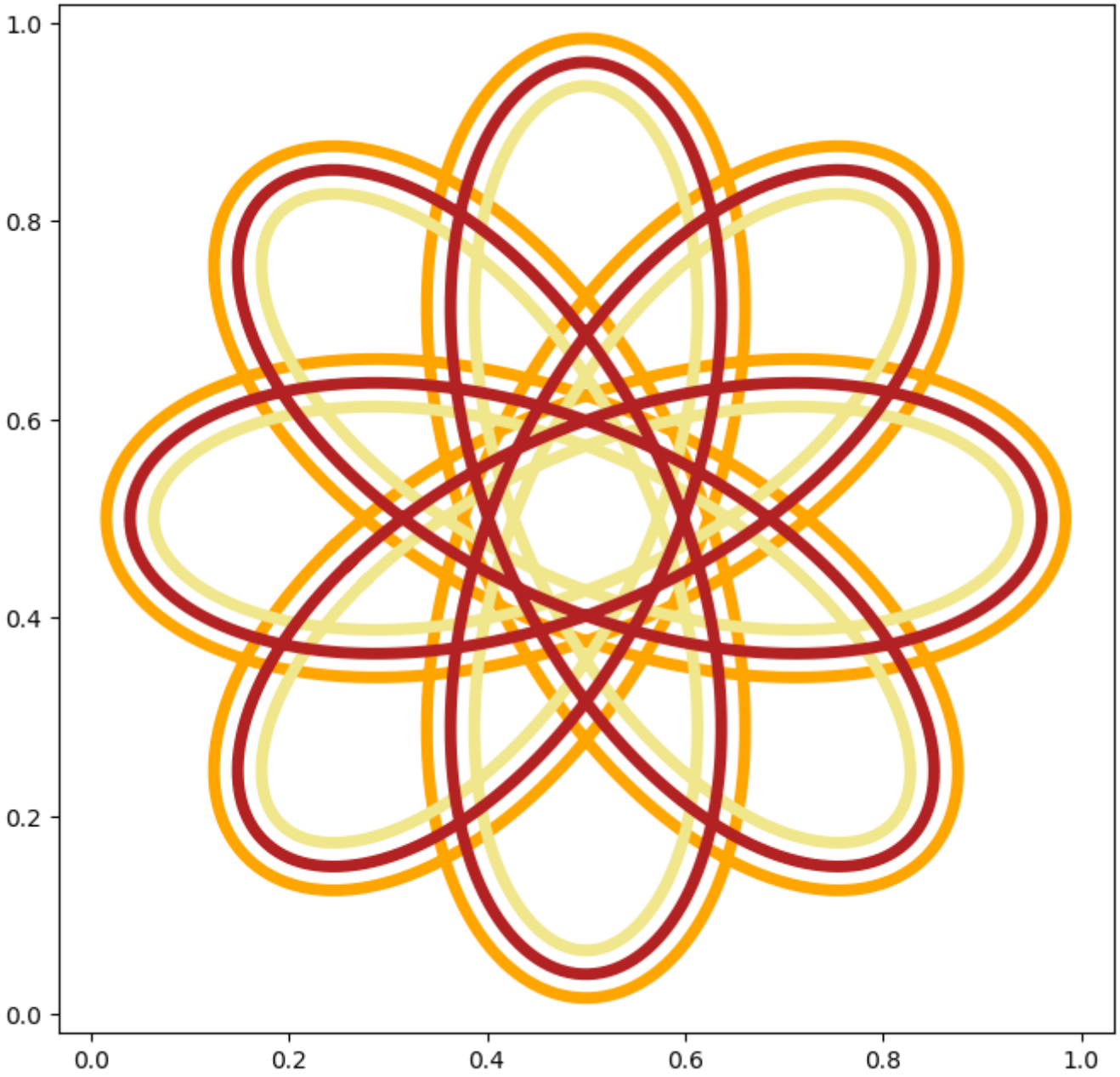
https://github.com/t-o-k/scikit-vectors_examples/

In [18]:

```
# Calculate points for two offset curves

d = 0.048 / 2
p_dm = p_o - d * v_n
p_dp = p_o + d * v_n
```

In [19]:

```
# Show the curve together with the two offset curves

lw = 5

fig, ax = plt.subplots(figsize=figure_size, dpi=figure_dpi)
fig.text(0.30, 0.05, url)
ax.plot(*p_dm, c='orange', linewidth=lw)
ax.plot(*p_dp, c='khaki', linewidth=lw)
ax.plot(*p_o, c='firebrick', linewidth=lw)
ax.axis('equal')
plt.show()
```



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

In [20]:

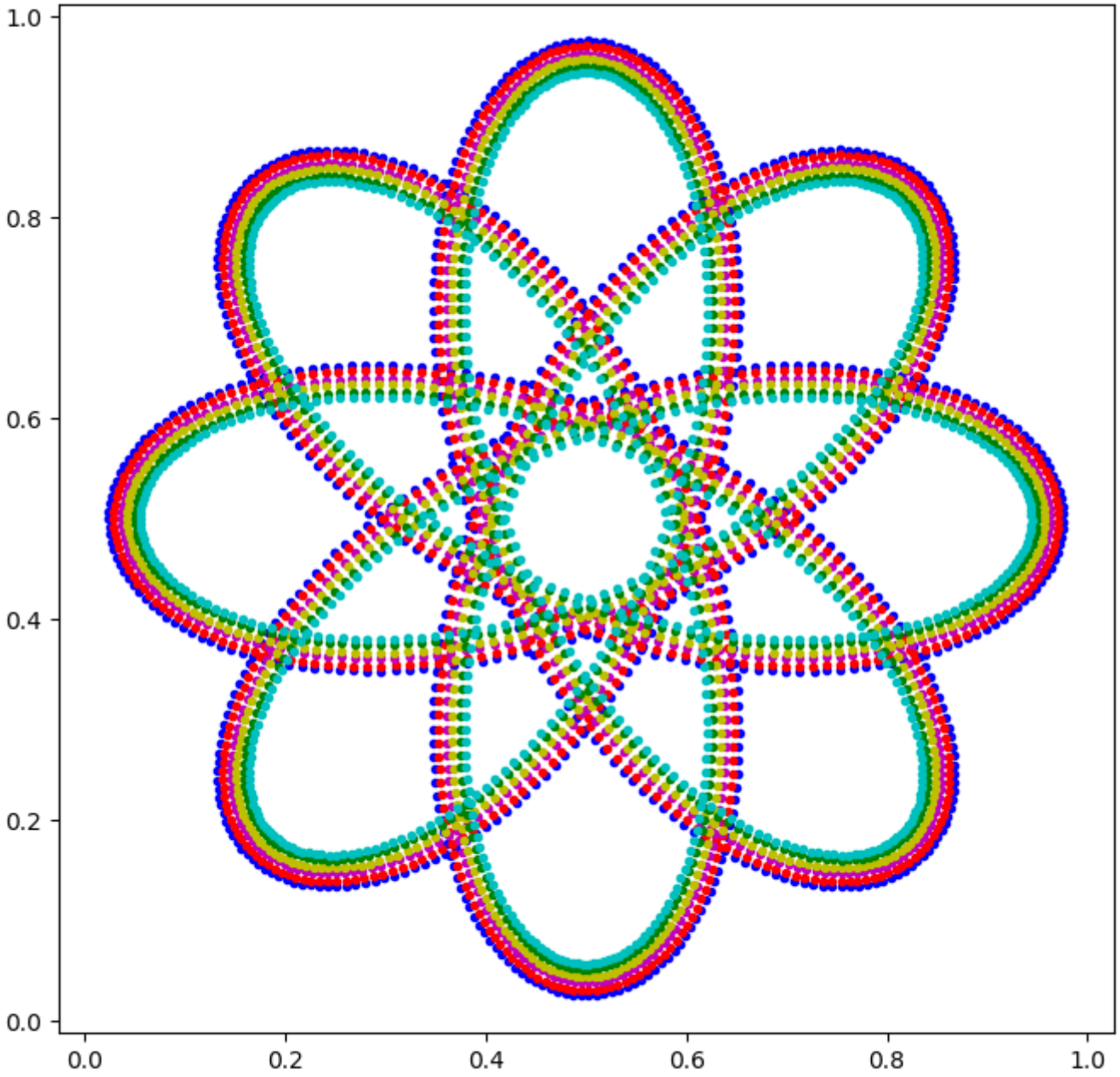
```
# Calculate points for four more offset curves

a = 0.0133 - 0.0025
p_am = p_o - a * v_n
p_ap = p_o + a * v_n

b = 0.0133 + 0.0025
p_bm = p_o - b * v_n
p_bp = p_o + b * v_n
```

In [21]:

```
# Calculate points for two more offset curves closer to the "center"

c = 0.0025
p_cm = p_o - c * v_n
p_cp = p_o + c * v_n
```

In [22]:

```
# Show these points for these six offset curves

fig, ax = plt.subplots(figsize=figure_size, dpi=figure_dpi)
fig.text(0.30, 0.05, url)
ax.scatter(p_bm.x, p_bm.y, c='b', marker='.')
ax.scatter(p_am.x, p_am.y, c='r', marker='.')
ax.scatter(p_cm.x, p_cm.y, c='m', marker='.')
ax.scatter(p_cp.x, p_cp.y, c='y', marker='.')
ax.scatter(p_ap.x, p_ap.y, c='g', marker='.')
ax.scatter(p_bp.x, p_bp.y, c='c', marker='.')
ax.axis('equal')
plt.show()
```



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

In [23]:

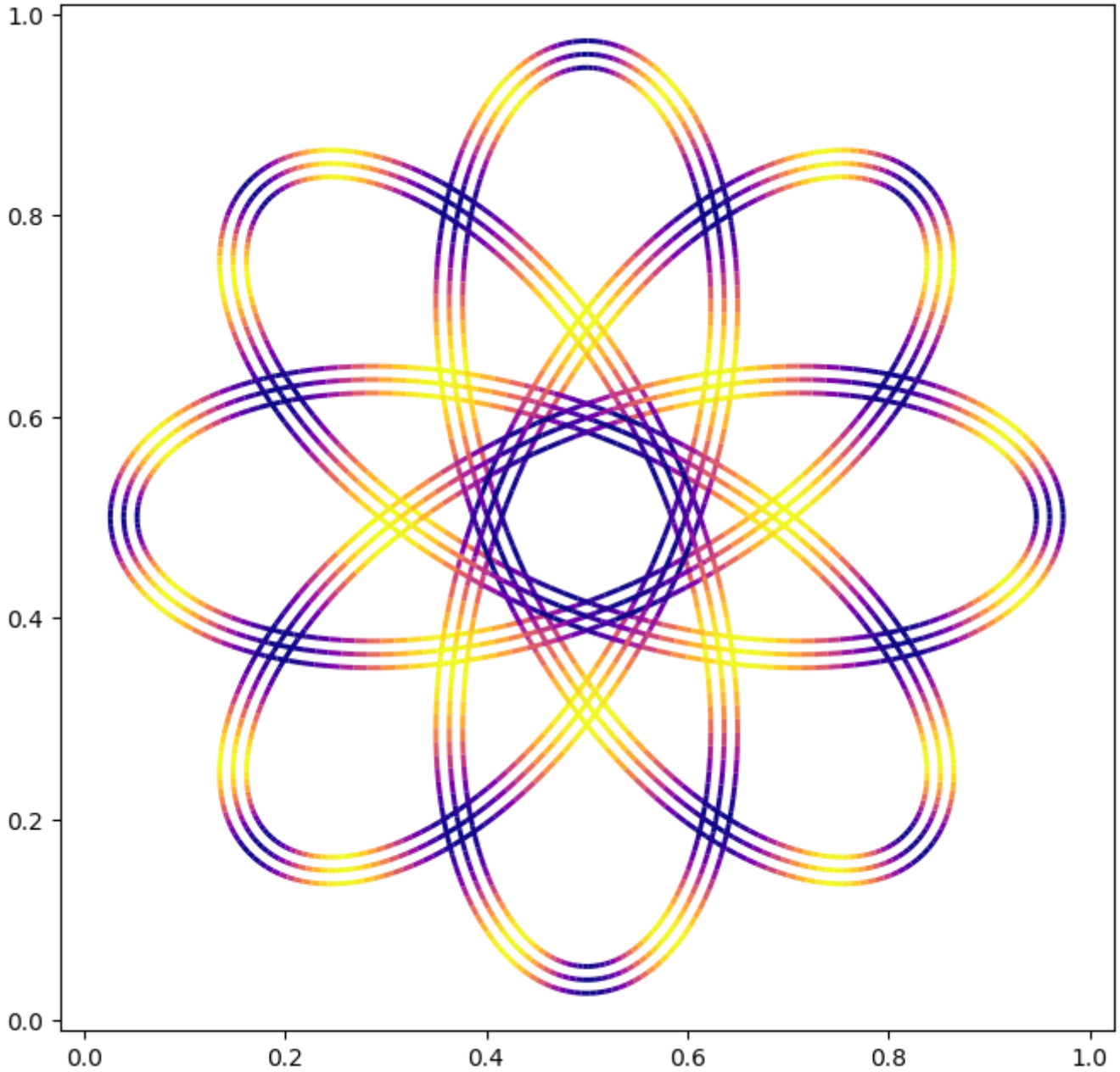
```
# Prepare for plotting with quad patches (between pairs of offset curves) instead of lines

def Patches(p_e, p_f, color='grey'):

    no_of_points = len(p_e.cnull)

    return \
        [
            Polygon(
                [
                    [ p_e.x[j], p_e.y[j] ],
                    [ p_f.x[j], p_f.y[j] ],
                    [ p_f.x[k], p_f.y[k] ],
                    [ p_e.x[k], p_e.y[k] ]
                ],
                closed = True,
                color = color
            )
            for j, k in zip(range(0, no_of_points-1), range(1, no_of_points))
        ]

patches_outer = Patches(p_am, p_bm)
patches_inner = Patches(p_ap, p_bp)
patches_center = Patches(p_cm, p_cp)
```

In [24]:

```
# Prepare colors that cycle along the curves

c = 2 * np.pi / (no_of_points_along_curve / 32)
d = 2 * np.pi * (1 / 2 + 1 / 16)

color_value = \
    np.array(
        [
            (np.sin(i * c - d) + 1) / 2
            for i in range(no_of_points_along_curve)
        ]
    )
```

In [25]:

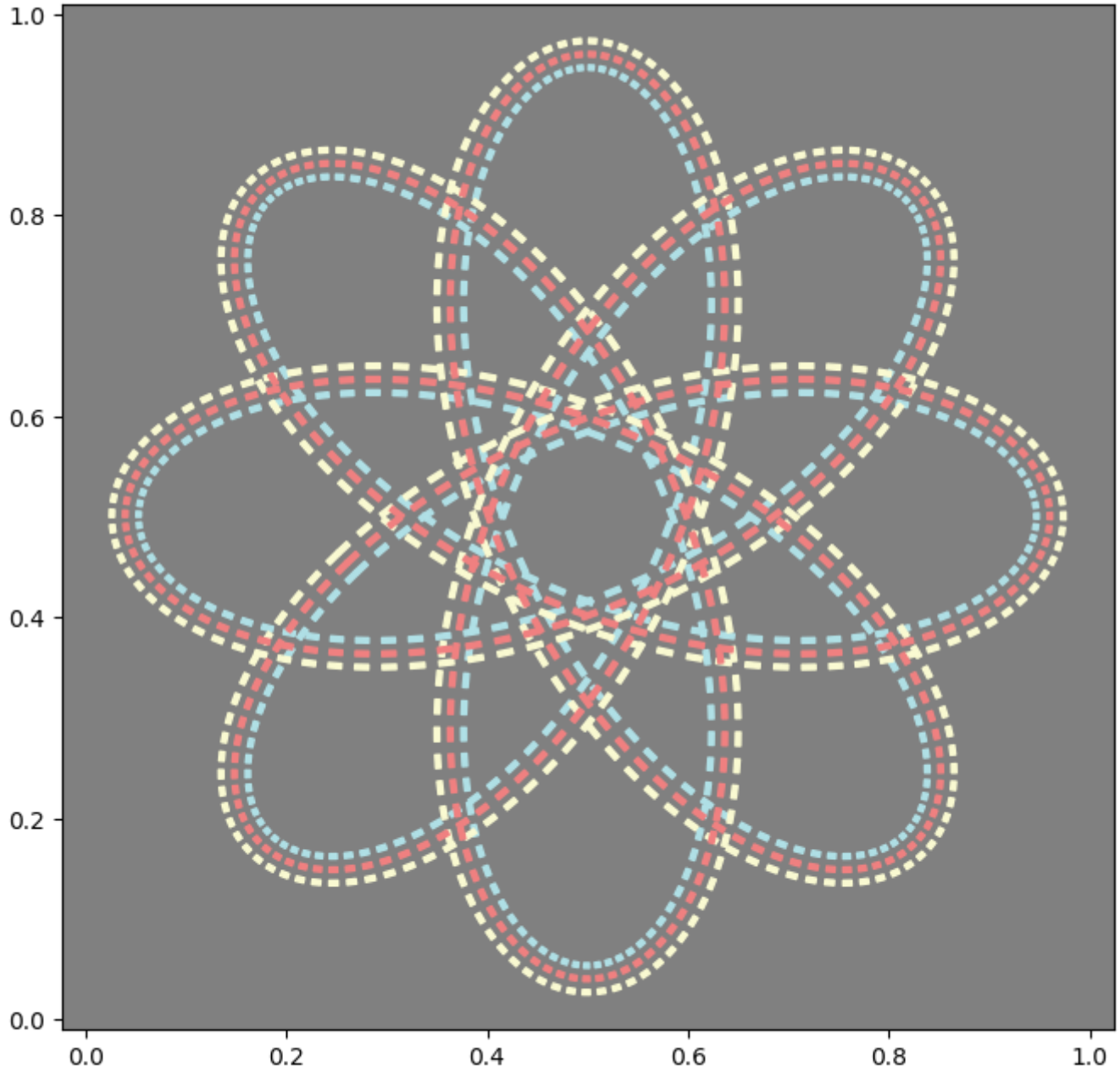
```
# Show the curves with colors cycling

fig = plt.figure(figsize=figure_size, dpi=figure_dpi)
fig.text(0.30, 0.05, url)
ax = fig.add_subplot(1, 1, 1)
ax.add_collection(
    PatchCollection(
        patches_inner,
        # match_original = True,
        array = color_value,
        cmap = plt.cm.plasma
    )
)
ax.add_collection(
    PatchCollection(
        patches_outer,
        # match_original = True,
        array = color_value,
        cmap = plt.cm.plasma
    )
)
ax.add_collection(
    PatchCollection(
        patches_center,
        # match_original = True,
        array = color_value,
        cmap = plt.cm.plasma
    )
)
ax.axis('equal')
plt.show()
```



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

In [26]:

```
# Show every other patch along the curves

s = 2  # stride

fig, ax = plt.subplots(figsize=figure_size, dpi=figure_dpi)
fig.text(0.30, 0.05, url)
ax.add_collection(
    PatchCollection(
        patches_inner[::s],
        facecolor='powderblue',
        edgecolor='powderblue'
    )
)
ax.add_collection(
    PatchCollection(
        patches_outer[::s],
        facecolor='lightgoldenrodyellow',
        edgecolor='lightgoldenrodyellow'
    )
)
ax.add_collection(
    PatchCollection(
        patches_center[::s],
        facecolor='lightcoral',
        edgecolor='lightcoral'
    )
)
ax.set_facecolor('gray')
ax.axis('equal')
plt.show()
```



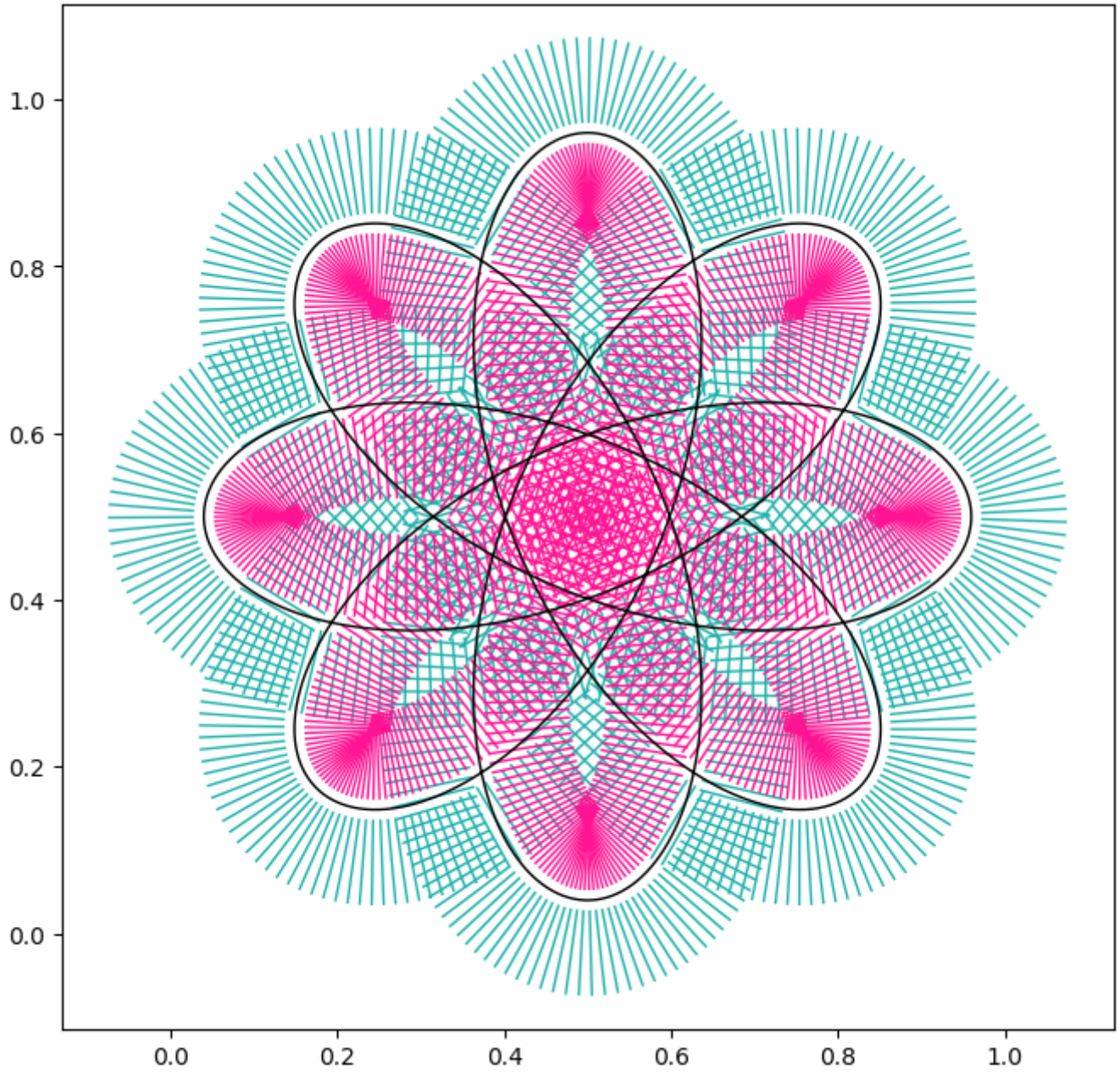
https://github.com/t-o-k/scikit-vectors_examples/

In [27]:

```
# Calculate points for some other offset curves that are further apart

a = 0.0133
p_am = p_o - a * v_n
p_ap = p_o + a * v_n

b = 0.0133 + 0.1000
p_bm = p_o - b * v_n
p_bp = p_o + b * v_n
```

In [28]:

```
# Now try to be a bit artistic...

lw = 1

fig, ax = plt.subplots(figsize=figure_size, dpi=figure_dpi)
fig.text(0.30, 0.05, url)
# ax.plot(*p_am, c='seagreen', linewidth=lw)
# ax.plot(*p_bm, c='seagreen', linewidth=lw)
# ax.plot(*p_ap, c='crimson', linewidth=lw)
# ax.plot(*p_bp, c='crimson', linewidth=lw)
# ax.plot(*zip(p_am, p_ap), c='black', linewidth=lw)
ax.plot(*zip(p_am, p_bm), c='lightseagreen', linewidth=lw)
ax.plot(*zip(p_ap, p_bp), c='deeppink', linewidth=lw)
ax.plot(*p_o, c='black', linewidth=lw)
ax.axis('equal')
plt.show()
```


1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
https://github.com/t-o-k/scikit-vectors_examples/

In [ ]:
```
1
```