

Orienting boxes to Frenet frames along a curve

- using Matplotlib, NumPy and scikit-vectors

Copyright (c) 2017 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 # Uncomment one of these to get a Matplotlib backend with interactive plots
        2
        3 # %matplotlib auto
        4 # %matplotlib notebook
```

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

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

```
In [4]: 1 # The functions for the curve
        2
        3 a, b, c = 4, 3, 2
        4
        5
        6 def f_x(t):
        7     return +a * np.cos(t)
        8
        9
        10
        11 def f_y(t):
        12     return +b * np.sin(t)
        13
        14
        15
        16 def f_z(t):
        17     return +c * np.sin(3 * t)
        18
```

```
In [5]: 1 # Numerical approximation of the first derivative
        2
        3 def first_derivative(fn, h=1e-4):
        4
        5     h2 = 2 * h
        6
        7
        8     def d1_fn(t):
        9
        10         return (fn(t + h) - fn(t - h)) / h2
        11
        12
        13     return d1_fn
```

```
In [6]: 1 # Numerical approximation of the second derivative
2
3 def second_derivative(fn, h=1e-4):
4     hh = h**2
5
6
7     def d2_fn(t):
8
9         return (fn(t + h) - 2 * fn(t) + fn(t - h)) / hh
10
11
12
13     return d2_fn
```

```
In [7]: 1 # Create derivative functions for the curve
2
3 d1_f_x = first_derivative(f_x)
4 d1_f_y = first_derivative(f_y)
5 d1_f_z = first_derivative(f_z)
6
7 d2_f_x = second_derivative(f_x)
8 d2_f_y = second_derivative(f_y)
9 d2_f_z = second_derivative(f_z)
```

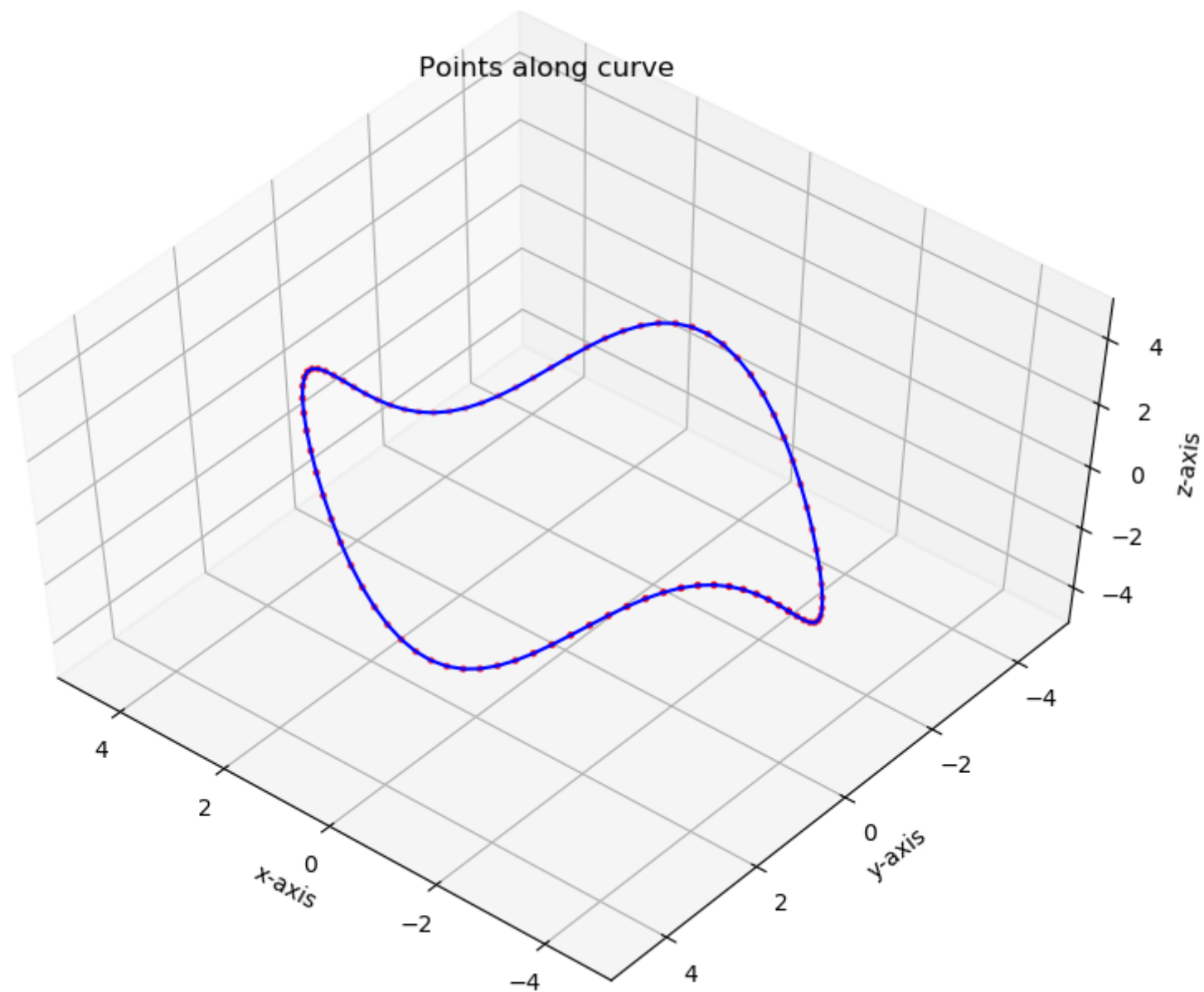
In [8]:

```
1  # Make a vector class that can hold all the points along the curve
2
3  no_of_points_along_curve = 90
4
5  NP3 = \
6      create_class_Cartesian_3D_Vector(
7          name = 'NP3',
8          component_names = 'xyz',
9          brackets = '<>',
10         sep = ', ',
11         cnull = np.zeros(no_of_points_along_curve),
12         cunit = np.ones(no_of_points_along_curve),
13         functions = \
14             {
15                 'not': np.logical_not,
16                 'and': np.logical_and,
17                 'or': np.logical_or,
18                 'all': np.all,
19                 'any': np.any,
20                 'min': np.minimum,
21                 'max': np.maximum,
22                 'abs': np.absolute,
23                 'trunc': np.trunc,
24                 'ceil': np.ceil,
25                 'copysign': np.copysign,
26                 'log10': np.log10,
27                 'cos': np.cos,
28                 'sin': np.sin,
29                 'atan2': np.arctan2,
30                 'pi': np.pi
31             }
32     )
```

```
In [9]: 1 # Calculate the position vectors for the points along the curve
        2
        3 t = np.linspace(-np.pi, +np.pi, no_of_points_along_curve)
        4
        5 p_o = \
        6     NP3(
        7         x = f_x(t),
        8         y = f_y(t),
        9         z = f_z(t)
       10     )
```

In [10]:

```
1  # Show the curve
2
3  fig = plt.figure(figsize=figure_size, dpi=figure_dpi)
4  ax = Axes3D(fig)
5  ax.set_title('Points along curve')
6  ax.scatter(p_o.x, p_o.y, p_o.z, c='r', marker='.')
7  ax.plot(p_o.x, p_o.y, p_o.z, c='b')
8  ax.set_xlim(-5, +5)
9  ax.set_ylim(-5, +5)
10 ax.set_zlim(-5, +5)
11 ax.set_xlabel('x-axis')
12 ax.set_ylabel('y-axis')
13 ax.set_zlabel('z-axis')
14 ax.view_init(elev=55, azimuth=130)
15
16 plt.show()
```



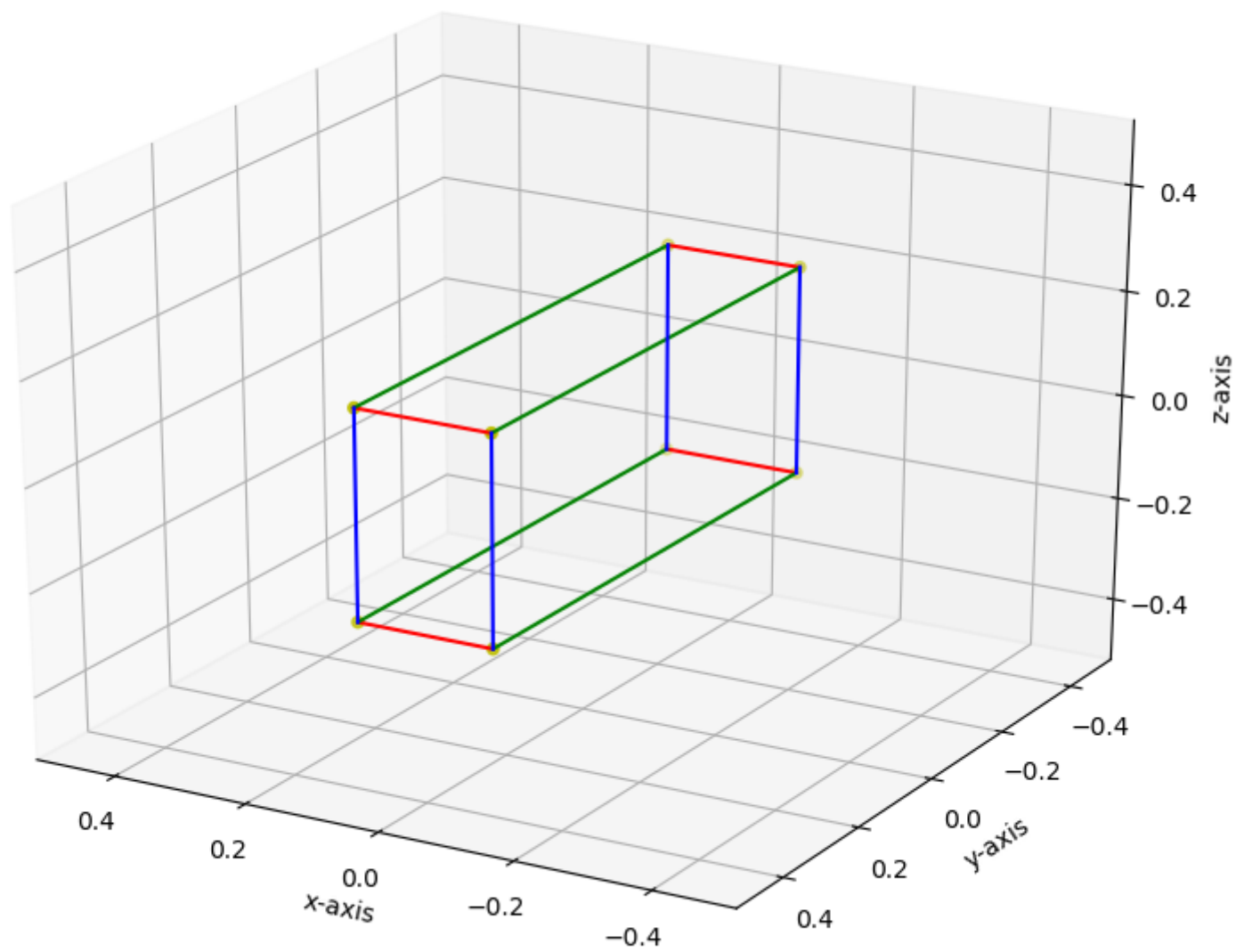
```
In [11]: 1 # The corners for a box
2
3 d, e, f = 1, 4, 2
4 scale = 1 / 10
5
6 d, e, f = scale * d, scale * e, scale * f
7
8 V3 = create_class_Cartesian_3D_Vector('V3', 'xyz')
9
10 box_corners = \
11     [
12         V3(-d, -e, -f),
13         V3(+d, -e, -f),
14         V3(+d, +e, -f),
15         V3(-d, +e, -f),
16         V3(-d, -e, +f),
17         V3(+d, -e, +f),
18         V3(+d, +e, +f),
19         V3(-d, +e, +f)
20     ]
```

```
In [12]: 1 # The edges of the box
2
3 line_indices = \
4     [
5         (0, 1),
6         (2, 3),
7         (4, 5),
8         (6, 7),
9         (1, 2),
10        (3, 0),
11        (5, 6),
12        (7, 4),
13        (0, 4),
14        (1, 5),
15        (2, 6),
16        (3, 7)
17    ]
18 line_colors = 'rrrrggggbbbb'
```


In [13]:

```
1  # Show the box corners and edges
2
3  fig = plt.figure(figsize=figure_size, dpi=figure_dpi)
4  ax = Axes3D(fig)
5  ax.set_title('Box corners and edges')
6  x, y, z = zip(*box_corners)
7  ax.scatter(x, y, z, c='y', marker='o')
8  for (i0, i1), color in zip(line_indices, line_colors):
9      ax.plot(
10         [ box_corners[i0].x, box_corners[i1].x ],
11         [ box_corners[i0].y, box_corners[i1].y ],
12         [ box_corners[i0].z, box_corners[i1].z ],
13         color = color
14     )
15  ax.set_xlim(-0.5, +0.5)
16  ax.set_ylim(-0.5, +0.5)
17  ax.set_zlim(-0.5, +0.5)
18  ax.set_xlabel('x-axis')
19  ax.set_ylabel('y-axis')
20  ax.set_zlabel('z-axis')
21  ax.view_init(elev=25, azimuth=120)
22
23  plt.show()
```

Box corners and edges

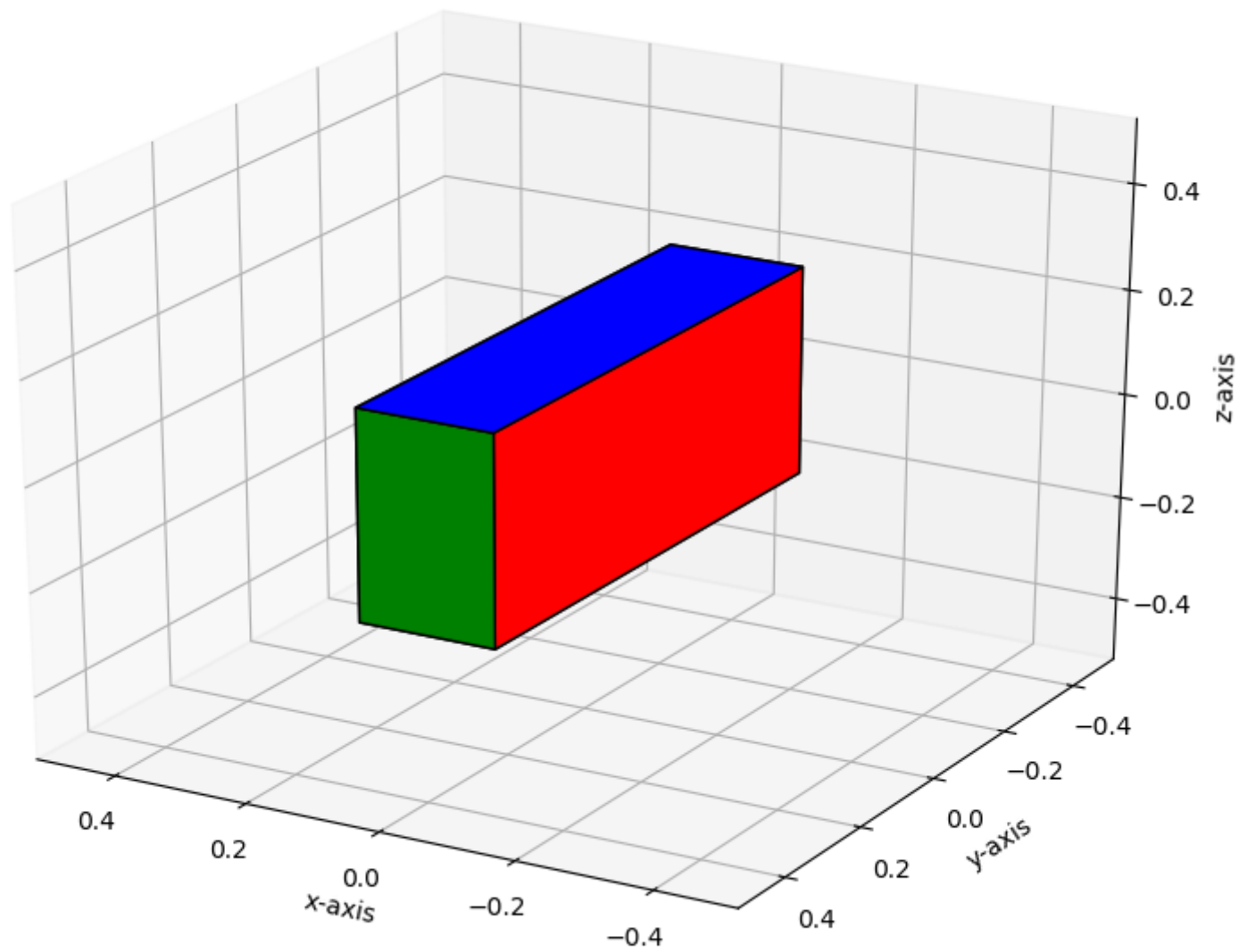


In [14]:

```
1  # The sides of the box
2
3  rectangle_indices = \
4      [
5          (0, 3, 7, 4),
6          (1, 2, 6, 5),
7          (0, 1, 5, 4),
8          (3, 2, 6, 7),
9          (0, 1, 2, 3),
10         (4, 5, 6, 7)
11     ]
12  rectangle_colors = 'rrggbb'
```

```
In [15]: 1 # Show the box sides
2
3 fig = plt.figure(figsize=figure_size, dpi=figure_dpi)
4 ax = Axes3D(fig)
5 ax.set_title('Box sides')
6 for indices, color in zip(rectangle_indices, rectangle_colors):
7     vertices = \
8         [
9             box_corners[i]
10            for i in indices
11        ]
12     rectangle = Poly3DCollection([ vertices ])
13     rectangle.set_color(color)
14     rectangle.set_edgecolor('k')
15     ax.add_collection3d(rectangle)
16 x, y, z = zip(*box_corners)
17 ax.set_xlim(-0.5, +0.5)
18 ax.set_ylim(-0.5, +0.5)
19 ax.set_zlim(-0.5, +0.5)
20 ax.set_xlabel('x-axis')
21 ax.set_ylabel('y-axis')
22 ax.set_zlabel('z-axis')
23 ax.view_init(elev=25, azimuth=120)
24
25 plt.show()
```

Box sides



```
In [16]: 1 # Calculate the vectors for the first derivatives at the points along the curve
2
3 v_d1 = \
4     NP3(
5         x = d1_f_x(t),
6         y = d1_f_y(t),
7         z = d1_f_z(t)
8     )
```

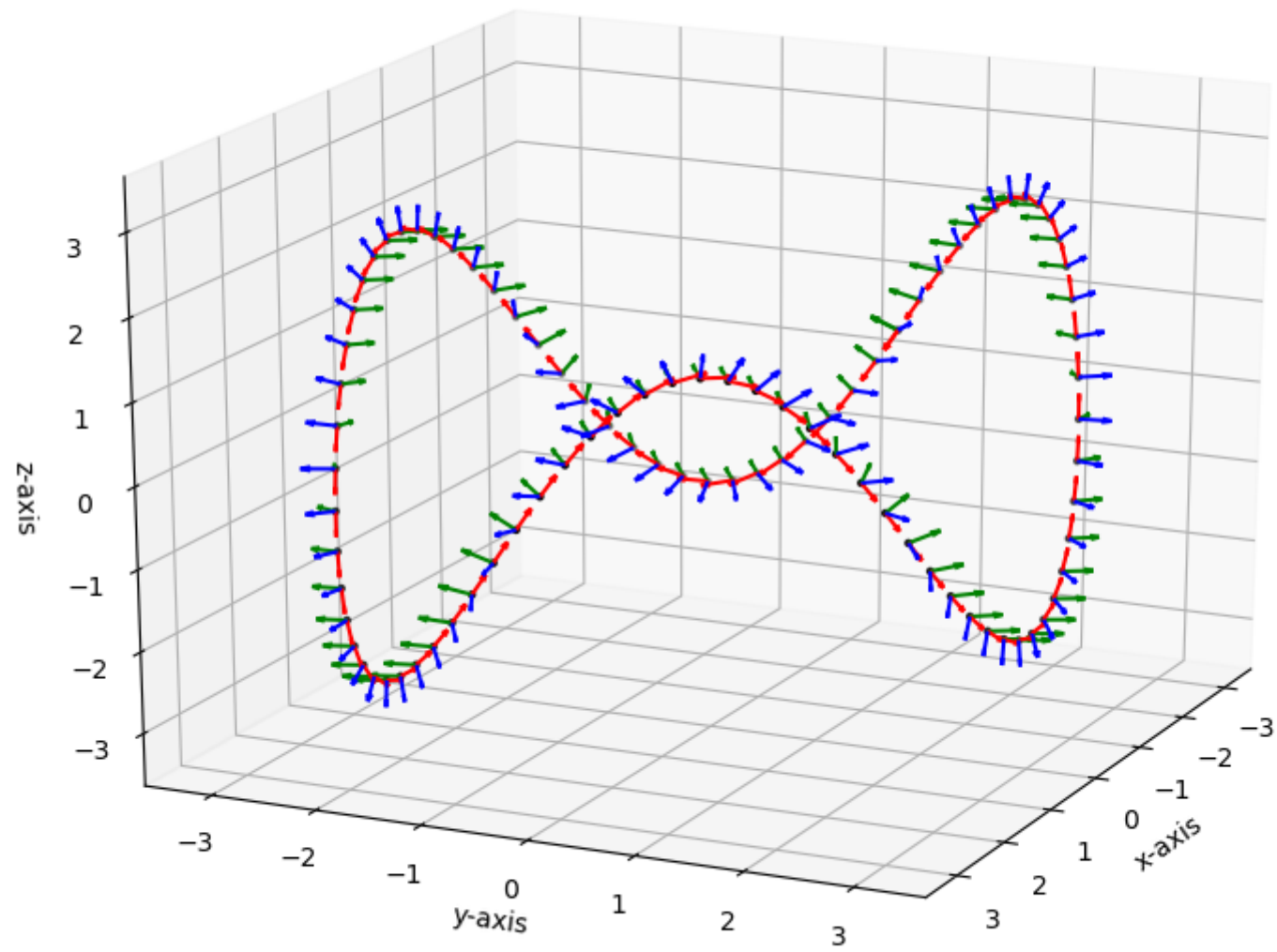
```
In [17]: 1 # Calculate the vectors from the second derivatives at the points along the curve
2
3 v_d2 = \
4     NP3(
5         x = d2_f_x(t),
6         y = d2_f_y(t),
7         z = d2_f_z(t)
8     )
```

```
In [18]: 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 [19]:

```
1  # Show the Frenet frames vectors
2
3  vector_length = 0.3
4
5  fig = plt.figure(figsize=figure_size, dpi=figure_dpi)
6  ax = Axes3D(fig)
7  ax.set_title('Frenet Frames')
8  ax.scatter(p_o.x, p_o.y, p_o.z, c='k', marker='.')
9  ax.quiver(
10     p_o.x, p_o.y, p_o.z,
11     v_t.x, v_t.y, v_t.z,
12     length = vector_length,
13     pivot = 'tail',
14     color = 'r'
15 )
16 ax.quiver(
17     p_o.x, p_o.y, p_o.z,
18     v_b.x, v_b.y, v_b.z,
19     length = vector_length,
20     pivot = 'tail',
21     color = 'g'
22 )
23 ax.quiver(
24     p_o.x, p_o.y, p_o.z,
25     v_n.x, v_n.y, v_n.z,
26     length = vector_length,
27     pivot = 'tail',
28     color = 'b'
29 )
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=20, azim=25)
37 # ax.view_init(elev=60, azim=35)
38
39 plt.show()
```

Frenet Frames

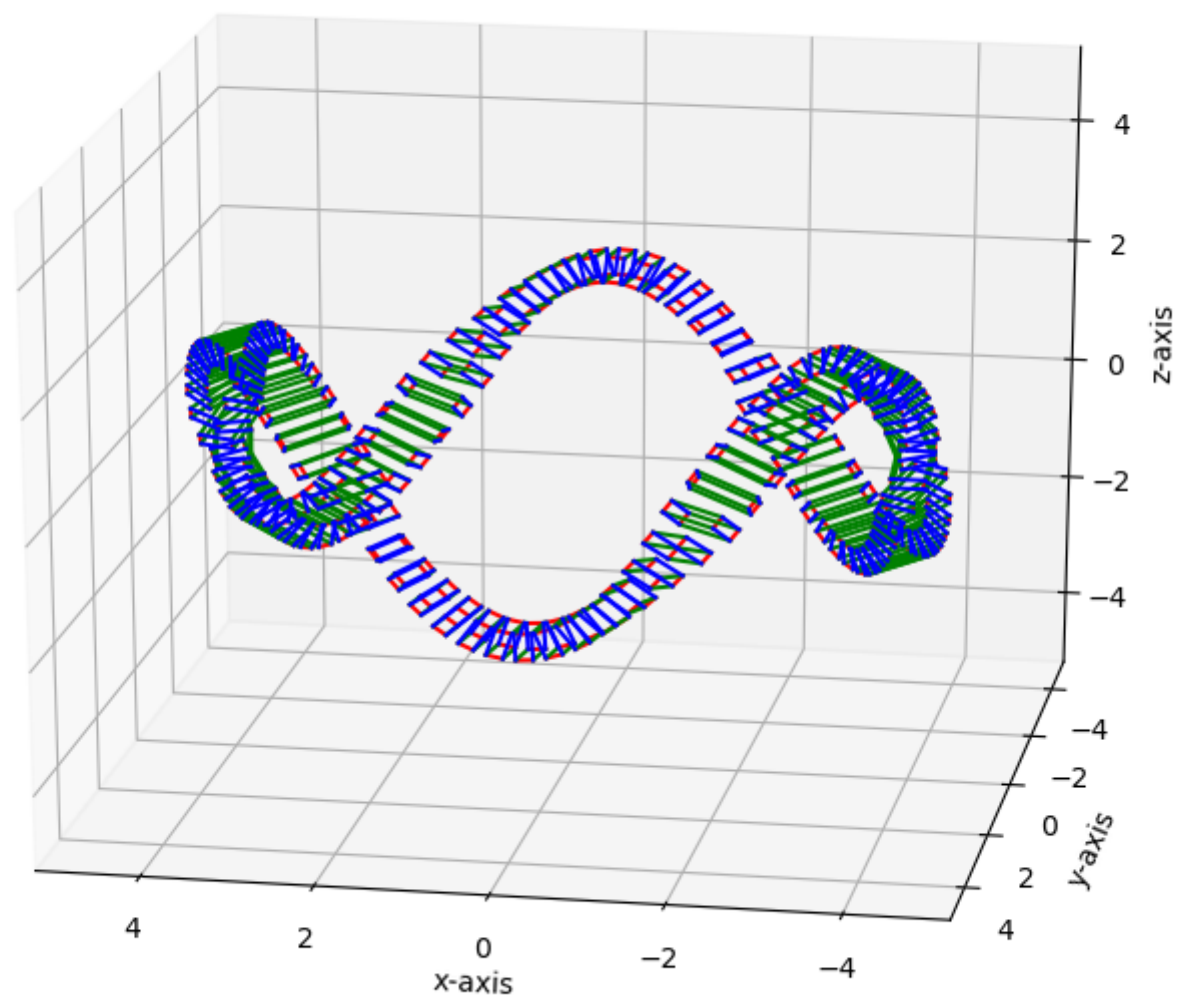


In [20]:

```
1  # Reorient all the box corners in all the Frenet frames
2
3  box_corners_transformed = \
4      [
5          p_o + corner.x * v_t + corner.y * v_b + corner.z * v_n
6          for corner in box_corners
7      ]
```

```
In [21]: 1 # Show the edges for all the reoriented boxes along the curve
2
3 fig = plt.figure(figsize=figure_size, dpi=figure_dpi)
4 ax = Axes3D(fig)
5 ax.set_title('Reoriented boxes')
6 for (i0, i1), color in zip(line_indices, line_colors):
7     p0 = box_corners_transformed[i0]
8     p1 = box_corners_transformed[i1]
9     for k in range(no_of_points_along_curve-1):
10         ax.plot(
11             [ p0.x[k], p1.x[k] ],
12             [ p0.y[k], p1.y[k] ],
13             [ p0.z[k], p1.z[k] ],
14             color = color
15         )
16 ax.set_xlim(-5, +5)
17 ax.set_ylim(-5, +5)
18 ax.set_zlim(-5, +5)
19 ax.set_xlabel('x-axis')
20 ax.set_ylabel('y-axis')
21 ax.set_zlabel('z-axis')
22 ax.view_init(elev=20, azimuth=100)
23
24 plt.show()
```

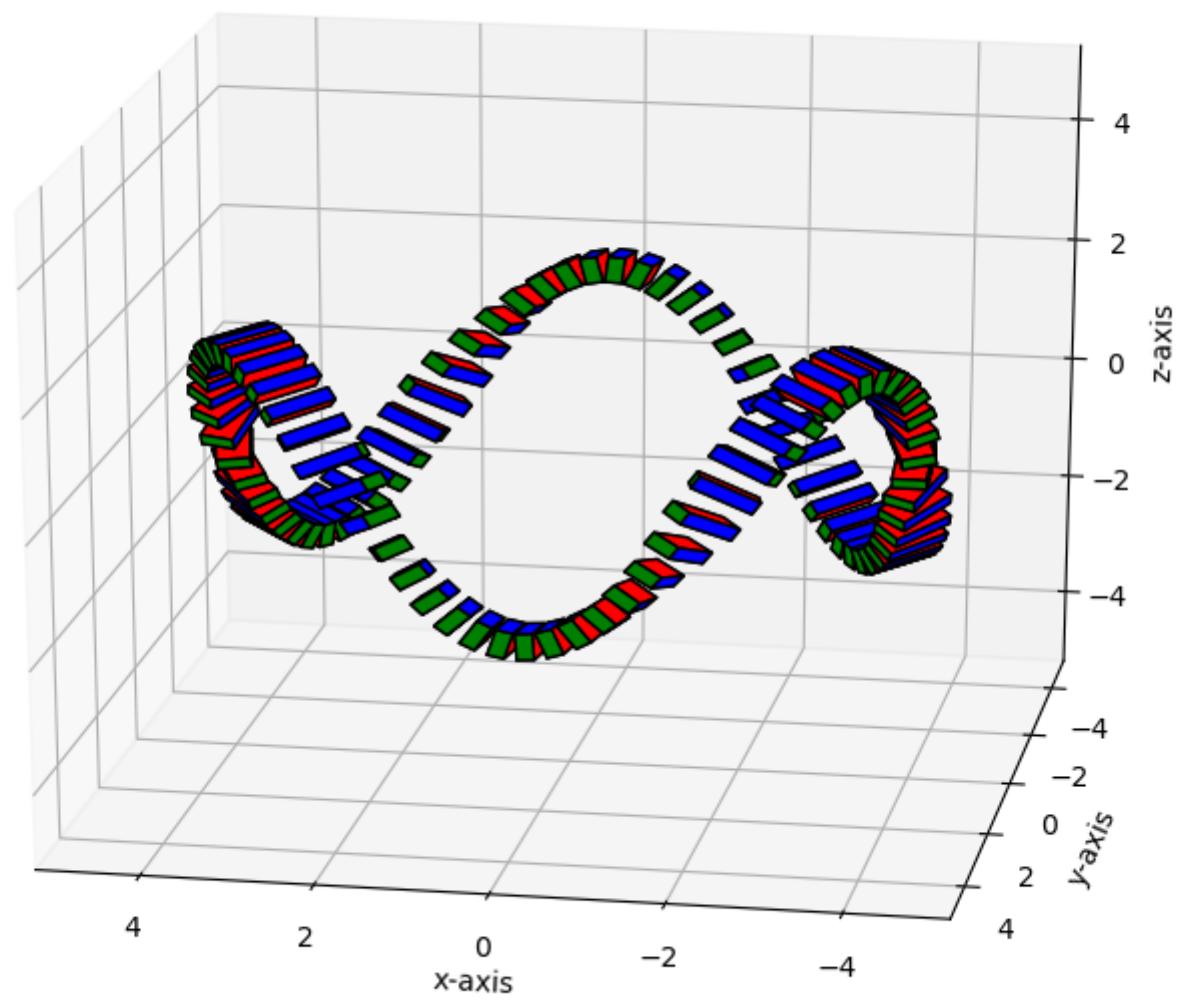
Reoriented boxes



In [22]:

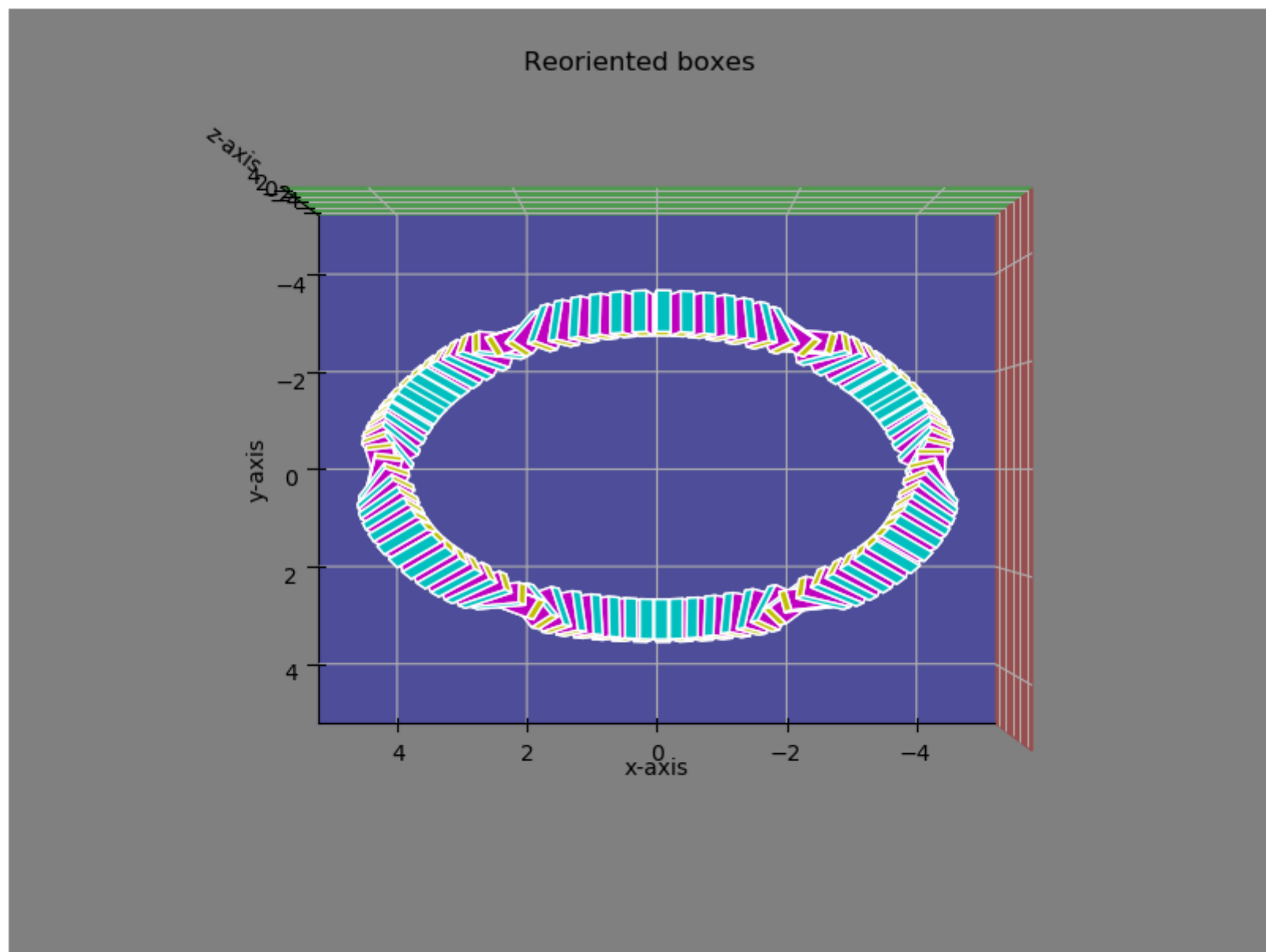
```
1  # Show the sides for all the reoriented boxes along the curve
2
3  fig = plt.figure(figsize=figure_size, dpi=figure_dpi)
4  ax = Axes3D(fig)
5  ax.set_title('Reoriented boxes')
6  for (i0, i1, i2, i3), color in zip(rectangle_indices, rectangle_colors):
7      p0 = box_corners_transformed[i0]
8      p1 = box_corners_transformed[i1]
9      p2 = box_corners_transformed[i2]
10     p3 = box_corners_transformed[i3]
11     for k in range(no_of_points_along_curve-1):
12         vertices = \
13             (
14                 (p0.x[k], p0.y[k], p0.z[k]),
15                 (p1.x[k], p1.y[k], p1.z[k]),
16                 (p2.x[k], p2.y[k], p2.z[k]),
17                 (p3.x[k], p3.y[k], p3.z[k])
18             )
19         rectangle = Poly3DCollection([ vertices ])
20         rectangle.set_color(color)
21         rectangle.set_edgecolor('black')
22         ax.add_collection3d(rectangle)
23 ax.set_xlabel('x-axis')
24 ax.set_ylabel('y-axis')
25 ax.set_zlabel('z-axis')
26 ax.set_xlim(-5, +5)
27 ax.set_ylim(-5, +5)
28 ax.set_zlim(-5, +5)
29 ax.view_init(elev=20, azimuth=100)
30
31 plt.show()
```

Reoriented boxes



In [23]:

```
1  # Change some colors - and do it somewhat differently
2
3  color_r = (0.6, 0.3, 0.3, 1.0)
4  color_g = (0.3, 0.6, 0.3, 1.0)
5  color_b = (0.3, 0.3, 0.6, 1.0)
6  rectangle_colors = 'mmyycc'
7
8  fig = plt.figure(figsize=figure_size, dpi=figure_dpi)
9  ax = Axes3D(fig)
10 ax.set_title('Reoriented boxes')
11 ax.set_facecolor('gray')
12 ax.w_xaxis.set_pane_color(color_r)
13 ax.w_yaxis.set_pane_color(color_g)
14 ax.w_zaxis.set_pane_color(color_b)
15 for indices, color in zip(rectangle_indices, rectangle_colors):
16     corners = [ box_corners_transformed[i] for i in indices ]
17     for k in range(no_of_points_along_curve-1):
18         fn = lambda cv: cv[k] # To fetch element no. k from the np.array in each vector component
19         vertices = tuple(p(fn) for p in corners)
20         rectangle = Poly3DCollection([ vertices ])
21         rectangle.set_color(color)
22         rectangle.set_edgecolor('white')
23         ax.add_collection3d(rectangle)
24 ax.set_xlabel('x-axis')
25 ax.set_ylabel('y-axis')
26 ax.set_zlabel('z-axis')
27 ax.set_xlim(-5, +5)
28 ax.set_ylim(-5, +5)
29 ax.set_zlim(-5, +5)
30 ax.view_init(elev=90, azim=90)
31
32 plt.show()
```



In []:

1