

Creating Bezier surfaces

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

```
In [2]: 1 # This example has been tested with NumPy v1.15.3, Matplotlib v2.1.1 and Jupyter v4.4.0
```

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

```
In [4]: 1 import operator
2 from functools import reduce
3 import matplotlib.pyplot as plt
4 import matplotlib.tri as mtri
5 from mpl_toolkits.mplot3d import Axes3D
6 import numpy as np
7
8 from skvectors import create_class_Cartesian_3D_Vector
```

```
In [5]: 1 # Size and resolution for Matplotlib figures
2
3 figure_size = (8, 6)
4 figure_dpi = 100
```

In [6]:

```
1 class Bicubic_Bezier():
2
3     blend_fns = \
4     [
5         lambda s: (1 - s)**3,
6         lambda s: 3 * s * (1 - s)**2,
7         lambda s: 3 * s**2 * (1 - s),
8         lambda s: s**3
9     ]
10
11     @staticmethod
12     def _sum(values):
13
14         return reduce(operator.add, values)
15
16
17     def __init__(self, points4x4):
18
19         self.points4x4 = points4x4
20
21
22     def __call__(self, u, v):
23
24         return \
25             self._sum(
26                 self.blend_fns[j](u) *
27                 self._sum(
28                     self.blend_fns[i](v) * self.points4x4[i][j]
29                     for i in range(4)
30                 )
31                 for j in range(4)
32             )
```

In [7]:

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

In [8]:

```
1 control_grid_shape = (4, 4)
2
3 ControlGrid3D = \
4     create_class_Cartesian_3D_Vector(
5         name = 'ControlGrid3D',
6         component_names = 'xyz',
7         cnull = np.zeros(control_grid_shape),
8         cunit = np.ones(control_grid_shape),
9         functions = np_functions
10    )
```

In [9]:

```
1 p3d_ctrl = \
2     ControlGrid3D(
3         x = \
4             np.array(
5                 [
6                     [ 0.0, 1.0, 2.0, 3.0 ],
7                     [ 0.0, 1.0, 2.0, 4.0 ],
8                     [ 0.0, 1.0, 2.0, 2.5 ],
9                     [ 0.0, 1.0, 2.0, 3.0 ],
10                ]
11            ),
12        y = \
13            np.array(
14                [
15                    [ 0.0, 0.0, 1.0, 0.0 ],
16                    [ 1.0, 1.0, 2.0, 1.0 ],
17                    [ 2.0, 2.0, 3.0, 2.0 ],
18                    [ 3.0, 3.0, 5.0, 3.0 ],
19                ]
20            ),
21        z = \
22            np.array(
23                [
24                    [ 2.0, 0.0, 0.0, -3.0 ],
25                    [ -2.0, -3.0, -2.0, 3.0 ],
26                    [ 0.0, -4.0, 0.0, 2.0 ],
27                    [ 2.0, 0.0, 0.0, -3.0 ],
28                ]
29            )
30    )
```

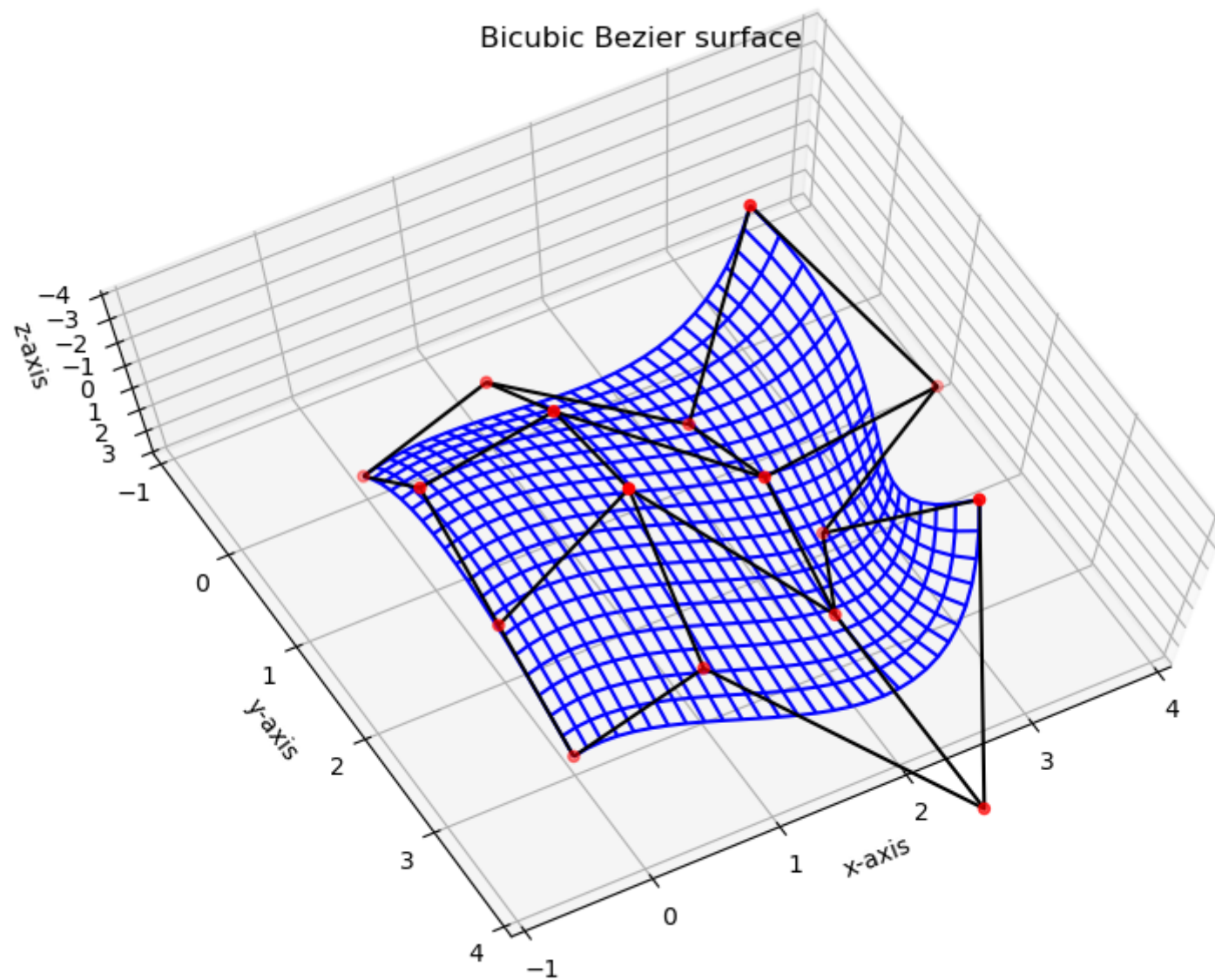
In [10]:

```
1 surface_shape = nr_u, nr_v = (20, 30)
2
3 Surface3D = \
4     create_class_Cartesian_3D_Vector(
5         name = 'Surface3D',
6         component_names = 'xyz',
7         cnull = np.zeros(surface_shape),
8         cunit = np.ones(surface_shape),
9         functions = np_functions
10    )
```

```
In [11]: 1 bb_x = Bicubic_Bezier(p3d_ctrl.x)
          2 bb_y = Bicubic_Bezier(p3d_ctrl.y)
          3 bb_z = Bicubic_Bezier(p3d_ctrl.z)
```

```
In [12]: 1 u, v = \
          2     np.meshgrid(
          3         np.arange(0, nr_v) / (nr_v - 1),
          4         np.arange(0, nr_u) / (nr_u - 1)
          5     )
          6
          7 bezier_points = \
          8     Surface3D(
          9         x = bb_x(u, v),
         10         y = bb_y(u, v),
         11         z = bb_z(u, v)
         12     )
```

```
In [13]: 1 fig = plt.figure(figsize=figure_size, dpi=figure_dpi)
2 fig.text(0.01, 0.01, url)
3 ax = Axes3D(fig)
4 ax.set_title('Bicubic Bezier surface')
5 ax.plot_wireframe(*p3d_ctrl, color='black')
6 ax.scatter(p3d_ctrl.x, p3d_ctrl.y, p3d_ctrl.z, c='r', marker='o')
7 ax.plot_wireframe(bezier_points.x, bezier_points.y, bezier_points.z, color='blue')
8 ax.set_xlabel('x-axis')
9 ax.set_ylabel('y-axis')
10 ax.set_zlabel('z-axis')
11 ax.set_xlim(-1, +4)
12 ax.set_ylim(-1, +4)
13 ax.set_zlim(-4, +3)
14 ax.view_init(elev=-105, azimuth=-61)
15 plt.show()
```



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

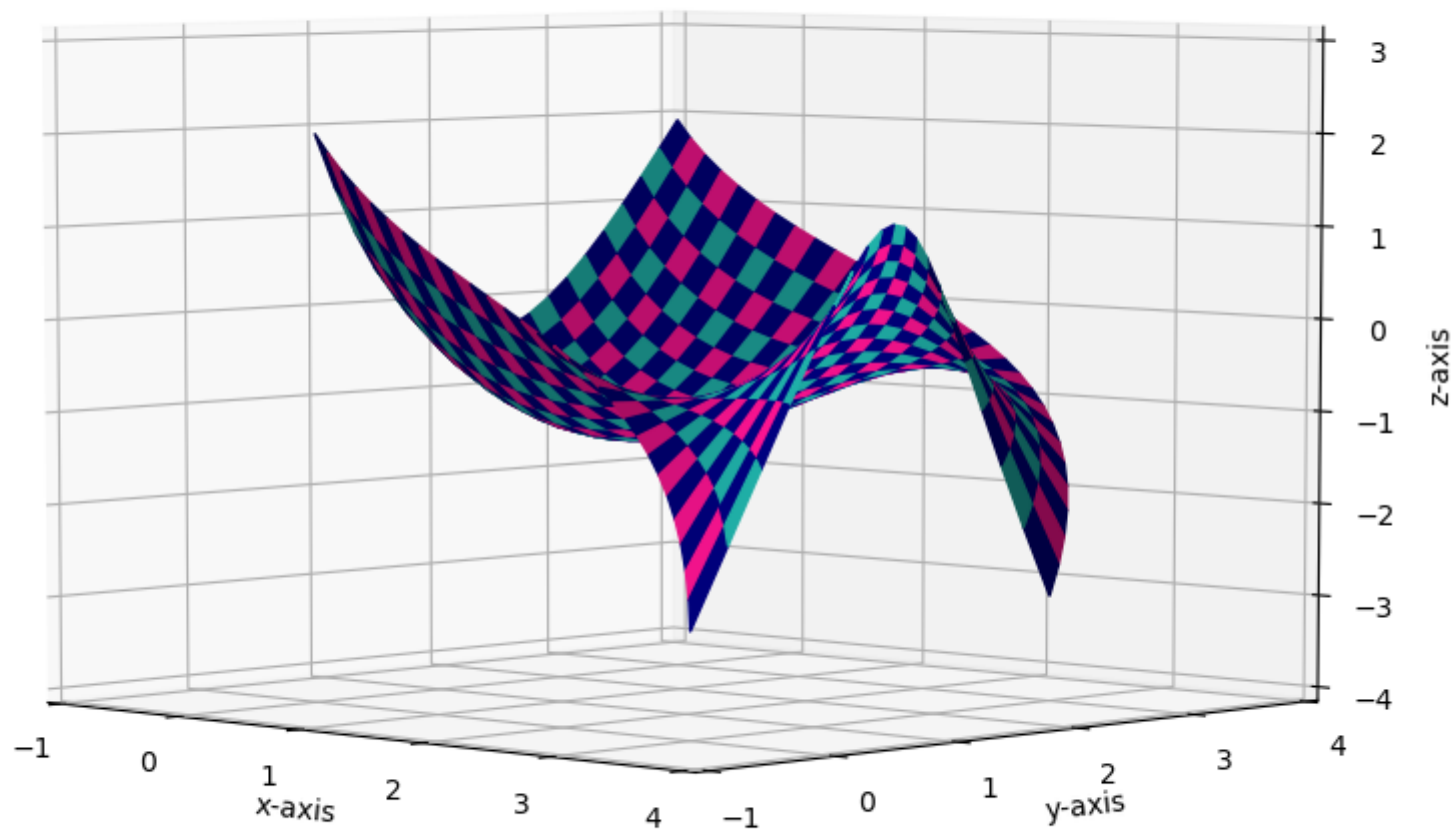
In [14]:

```
1  # Select colors for the faces
2
3  def select_color(i, j):
4
5      if (i + j) % 2 == 0:
6          color = 'navy'
7      elif j % 2 == 0:
8          color = 'lightseagreen'
9      else:
10         color = 'deeppink'
11
12     return color
13
14
15 face_colors = \
16     [
17         [
18             select_color(i, j)
19             for j in range(nr_v-1)
20         ]
21         for i in range(nr_u-1)
22     ]
```



```
In [15]: 1 fig = plt.figure(figsize=figure_size, dpi=figure_dpi)
2 fig.text(0.01, 0.01, url)
3 ax = Axes3D(fig)
4 ax.set_title('Bicubic Bezier surface')
5 ax.plot_surface(
6     bezier_points.x, bezier_points.y, bezier_points.z,
7     rstride = 1, cstride = 1,
8     facecolors = face_colors,
9     # cmap = plt.cm.inferno,
10    # shade = False
11 )
12 ax.set_xlabel('x-axis')
13 ax.set_ylabel('y-axis')
14 ax.set_zlabel('z-axis')
15 ax.set_xlim(-1, +4)
16 ax.set_ylim(-1, +4)
17 ax.set_zlim(-4, +3)
18 ax.view_init(elev=5, azimuth=-46)
19 plt.show()
```

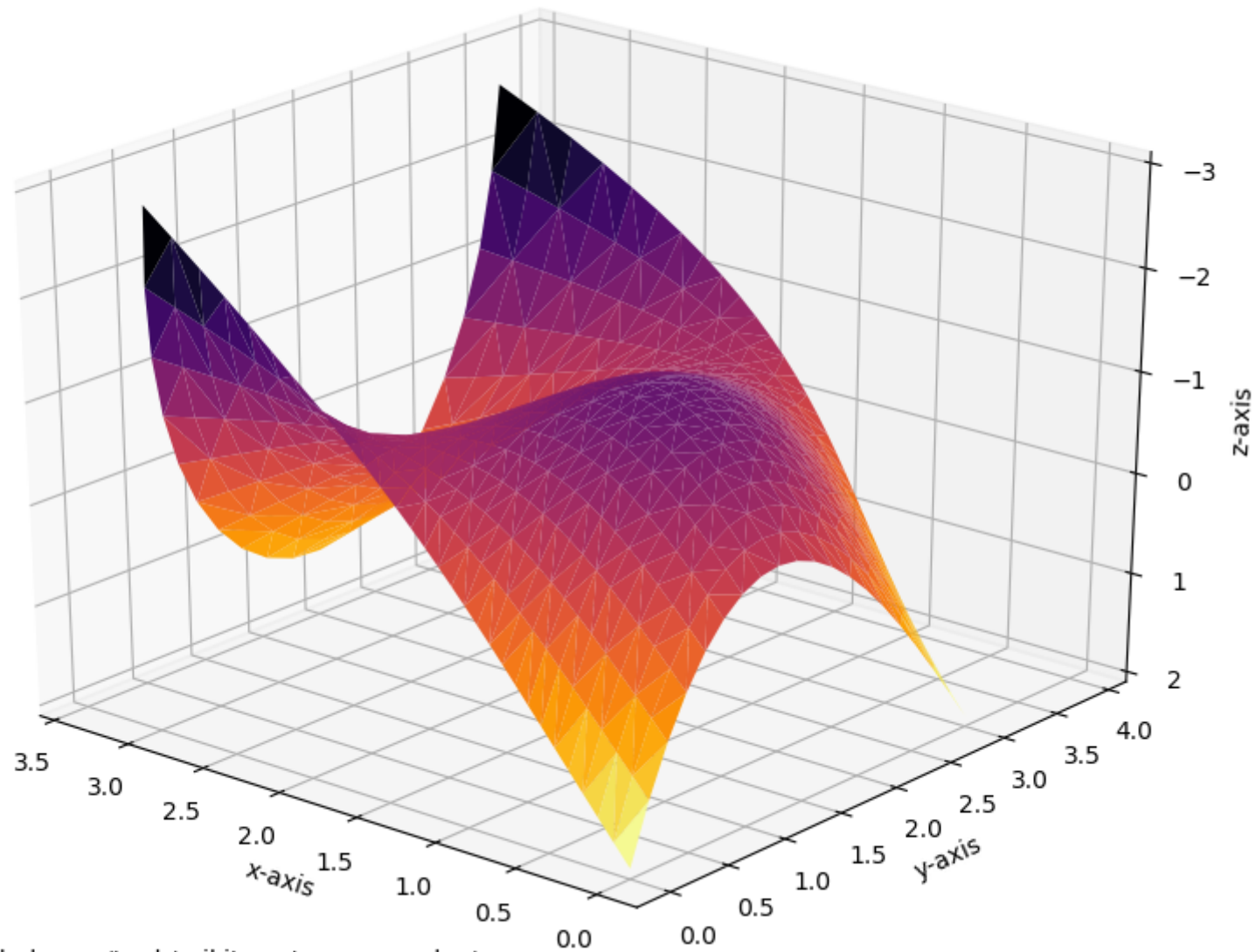
Bicubic Bezier surface



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

```
In [16]: 1 tri = \
2         mtri.Triangulation(
3             u.flatten(),
4             v.flatten()
5         )
6
7 fig = plt.figure(figsize=figure_size, dpi=figure_dpi)
8 fig.text(0.01, 0.01, url)
9 ax = Axes3D(fig)
10 ax.set_title('Bicubic Bezier surface')
11 ax.plot_trisurf(
12     bezier_points.x.flatten(),
13     bezier_points.y.flatten(),
14     bezier_points.z.flatten(),
15     triangles = tri.triangles,
16     cmap = plt.cm.inferno
17 )
18 ax.set_xlabel('x-axis')
19 ax.set_ylabel('y-axis')
20 ax.set_zlabel('z-axis')
21 ax.view_init(elev=-154, azimuth=50)
22 plt.show()
```

Bicubic Bezier surface



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

In [17]:

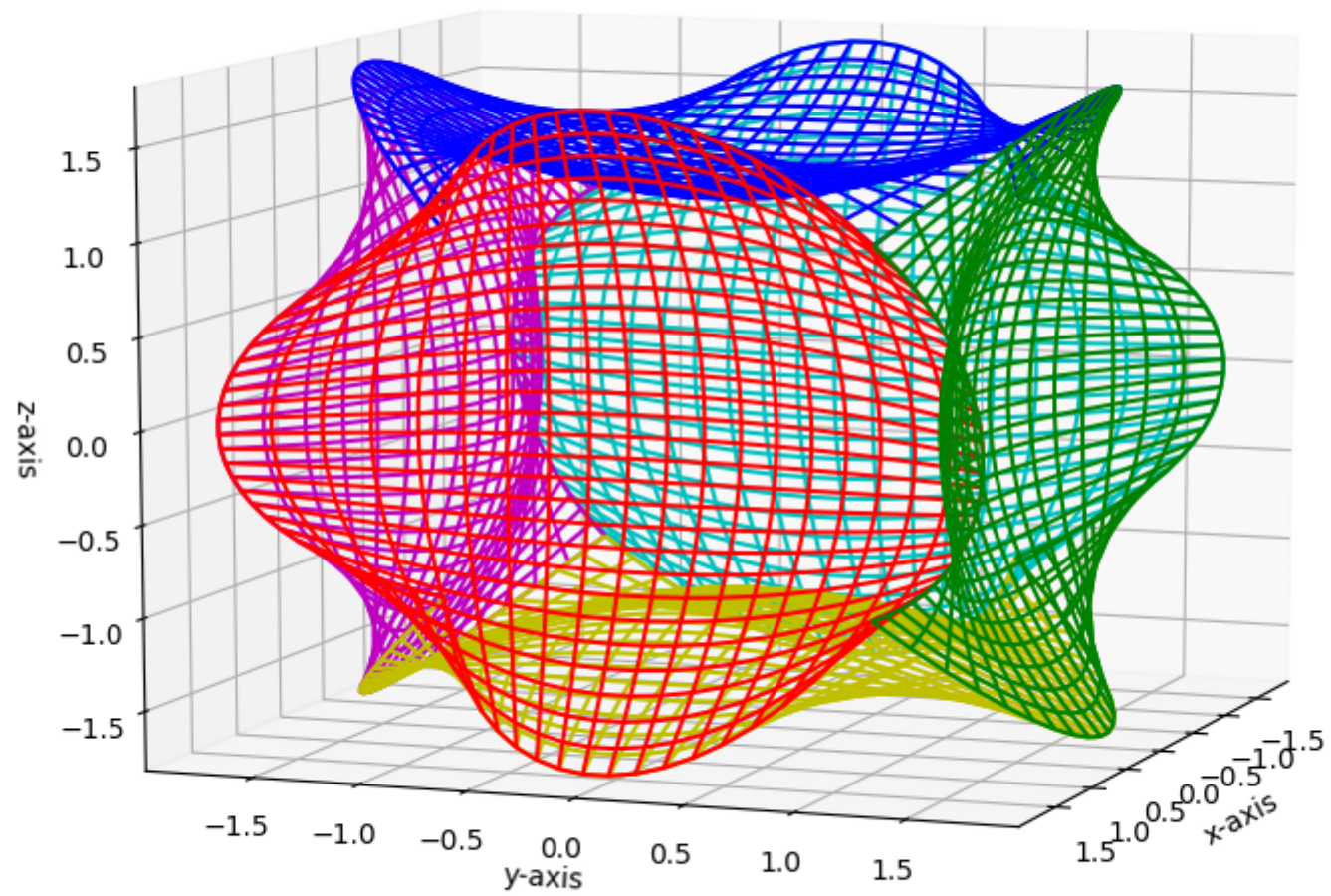
```
1 p3d_ctrl = \
2     ControlGrid3D(
3         x = \
4             np.array(
5                 [
6                     [ 1.0, 2.0, 2.0, 1.0 ],
7                     [ 2.0, 0.5, 0.5, 2.0 ],
8                     [ 2.0, 0.5, 0.5, 2.0 ],
9                     [ 1.0, 2.0, 2.0, 1.0 ]
10                ]
11            ),
12        y = \
13            np.array(
14                [
15                    [ -1.0, -2.0, -2.0, -1.0 ],
16                    [ -0.5, -0.5, -0.5, -0.5 ],
17                    [ 0.5, 0.5, 0.5, 0.5 ],
18                    [ 1.0, 2.0, 2.0, 1.0 ]
19                ]
20            ),
21        z = \
22            np.array(
23                [
24                    [ -1.0, -0.5, 0.5, 1.0 ],
25                    [ -2.0, -0.5, 0.5, 2.0 ],
26                    [ -2.0, -0.5, 0.5, 2.0 ],
27                    [ -1.0, -0.5, 0.5, 1.0 ]
28                ]
29            )
30    )
```

In [18]:

```
1  bb_x = Bicubic_Bezier(p3d_ctrl.x)
2  bb_y = Bicubic_Bezier(p3d_ctrl.y)
3  bb_z = Bicubic_Bezier(p3d_ctrl.z)
4
5  vxp = +Surface3D.basis_x()
6  vxn = -Surface3D.basis_x()
7  vyp = +Surface3D.basis_y()
8  vyn = -Surface3D.basis_y()
9  vzp = +Surface3D.basis_z()
10 vxn = -Surface3D.basis_z()
11
12 bezier_points_xp = \
13     Surface3D(
14         x = bb_x(u, v),
15         y = bb_y(u, v),
16         z = bb_z(u, v)
17     )
18
19 bezier_points_yp = bezier_points_xp.reorient(vxp, vyp)
20 bezier_points_yn = bezier_points_xp.reorient(vxp, vyn)
21 bezier_points_zp = bezier_points_xp.reorient(vxp, vzp)
22 bezier_points_zn = bezier_points_xp.reorient(vxp, vzn)
23 bezier_points_xn = bezier_points_yp.reorient(vyp, vxn)
24
25 bezier_surfaces = \
26     [
27         bezier_points_xp,
28         bezier_points_xn,
29         bezier_points_yp,
30         bezier_points_yn,
31         bezier_points_zp,
32         bezier_points_zn
33     ]
```

```
In [19]: 1 fig = plt.figure(figsize=figure_size, dpi=figure_dpi)
2 fig.text(0.01, 0.01, url)
3 ax = Axes3D(fig)
4 ax.set_title('Cube like shape made with Bicubic Bezier surfaces')
5 for surface, color in zip(bezier_surfaces, 'rcgmb'):
6     ax.plot_wireframe(*surface, color=color)
7 ax.set_xlabel('x-axis')
8 ax.set_ylabel('y-axis')
9 ax.set_zlabel('z-axis')
10 # ax.set_xlim(-1, +5)
11 # ax.set_ylim(-4, +3)
12 # ax.set_zlim(-1, +4)
13 ax.view_init(elev=10, azimuth=20)
14 plt.show()
```

Cube like shape made with Bicubic Bezier surfaces

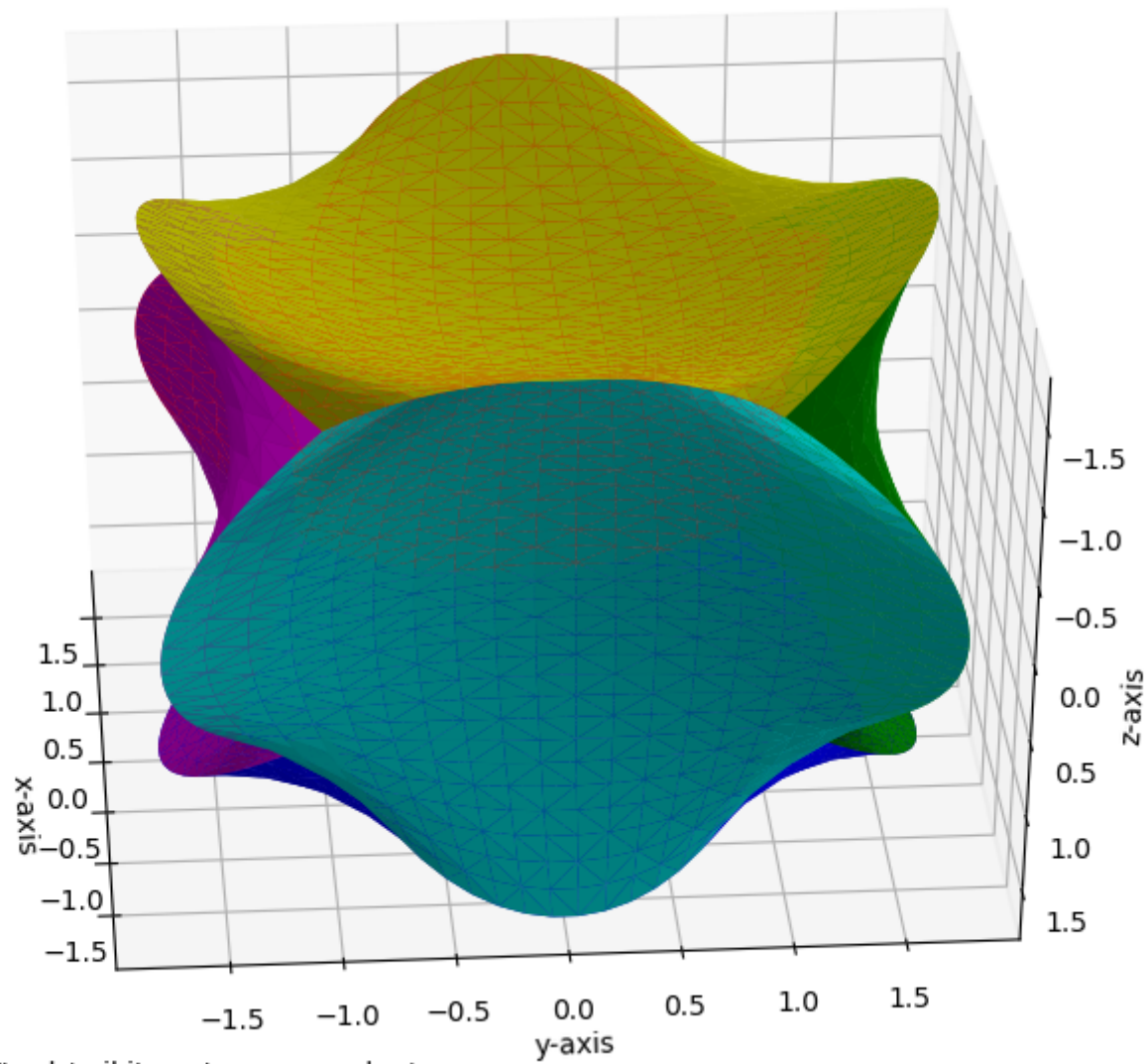


https://github.com/t-o-k/scikit-vectors_examples/

In [20]:

```
1 tri = \
2     mtri.Triangulation(
3         u.flatten(),
4         v.flatten()
5     )
6
7 fig = plt.figure(figsize=figure_size, dpi=figure_dpi)
8 fig.text(0.01, 0.01, url)
9 ax = Axes3D(fig)
10 ax.set_title('Cube like shape made with Bicubic Bezier surfaces')
11 for surface, color in zip(bezier_surfaces, 'rcgmby'):
12     ax.plot_trisurf(
13         *surface(np.ndarray.flatten),
14         triangles = tri.triangles,
15         color = color
16     )
17 ax.set_xlabel('x-axis')
18 ax.set_ylabel('y-axis')
19 ax.set_zlabel('z-axis')
20 ax.view_init(elev=-145, azimuth=4)
21 plt.show()
```

Cube like shape made with Bicubic Bezier surfaces



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

In []: 1

