

3.

```
1 #!/bin/env python3
2 """Graph a family of level surfaces for a function of three variables.
3
4 Example function:
5 f(x,y,z)=z/sqrt(x**2+y**2)"""
6 import sys
7 import numpy as np
8 import sympy
9 from sympy.abc import x, y, z, C
10 from matplotlib import cm, pyplot as plt
11 from mpl_toolkits.mplot3d import Axes3D
12
13
14 def repeat(an_object):
15     while True:
16         yield an_object
17
18
19 def prompt_expression():
20     """Gather a mathematical expression from the user. Raises OSError on
21     data/pipe mismatch.
22
23     $ ./graph.py # Pass expression via interactive prompt
24     f(x,y,z)=z/sqrt(x**2+y**2)
25
26     $ ./graph.py "z/sqrt(x**2+y**2)" # Pass expression as parameter
27
28     $ echo "z/sqrt(x**2+y**2)" | ./graph.py - # Pass expression over pipe"""
29     if len(sys.argv) > 1:
30         if sys.argv[1] == '-':
31             if sys.stdin.isatty():
32                 raise OSError('\'-\' flag with no pipe')
33             else:
34                 return sys.stdin.read()
35         else:
36             return sys.argv[1]
37     elif sys.stdin.isatty():
38         return input('f(x,y,z)=')
39     else:
40         raise OSError('pipe with no \'-\' flag')
41
42
43 def eval_expression(expression: str):
44     """Evaluate a string expression into a sympy expression. Available
45     namespace is that of sympy.* and x, y, and z variables."""
46     return eval(expression, vars(sympy), {'x': x, 'y': y, 'z': z})
47
48
49 def solve_z(expression: sympy.Expr):
50     """Solve C=f(x,y,z) for z for some sympy expression f(x,y,z). Returns a
51     list of expressions with variables x, y, and C."""
52     return sympy.solve(sympy.Eq(C, expression), z)
53
54
55 def solve_z_str(expression: str):
56     """Solve C=f(x,y,z) for z for some string expression f(x,y,z)."""
57     return solve_z(eval_expression(expression))
58
59
60 def solve_z_prompt():
61     """Solve C=f(x,y,z) for z for some expression f(x,y,z) that is prompted
62     for to the user."""
63     return solve_z_str(prompt_expression())
64
65
66 def make_lambda(expression: sympy.Expr, constant):
67     """Form a function of two variables from the sympy expression while
68     substituting the arbitrary constant C."""
69     return sympy.Lambda((x, y), expression.subs(C, constant))
70
71
72 # named after enumerate() but for colors
73 def e_color_ate(constants):
74     """Generate corresponding colors for each constant. Yields
75     color, constant."""
76     index_max = len(constants) - 1
77
78     for index, constant in enumerate(constants):
79         color = cm.jet(index / index_max, 0.93)
80         yield color, constant
81
82
83 def map_f_xy(f, mesh):
84     """Map a numpy (x,y) meshgrid to a function of two variables."""
85     domain = np.array(mesh).transpose(1, 2, 0)
86     z_range = np.empty(mesh[0].shape)
87
88     for i, domain_i in enumerate(domain):
89         for j, (x_ij, y_ij) in enumerate(domain_i):
90             z_ij = f(x_ij, y_ij)
91             z_range[i][j] = z_ij if z_ij.is_real else np.nan
92
93     return z_range
94
95
96 class LevelSurfacePlotter(Axes3D):
97     """Plot functions of two variables or level curves of functions of three
98     variables."""
99
100     def plot_f_xy(self, f, mesh_domain, **kwargs):
101         """Plot f(x,y) in three dimensions whereas mesh_domain is a numpy
102         meshgrid of (x,y) coordinates."""
103         super().plot_surface(
104             *mesh_domain, map_f_xy(f, mesh_domain),
105             **kwargs)
106
107     def plot_level_surfaces(
108         self, solutions: list, x_limits, y_limits, C_limits,
109         **kwargs
110     ):
111         """Plot a number of expressions that contain x, y, and C. The
112         parameters *_limits are lists of possible values."""
113         for color, constant in e_color_ate(C_limits):
114             for f_xy in map(make_lambda, solutions, repeat(constant)):
115                 self.plot_f_xy(
116                     f_xy, np.meshgrid(x_limits, y_limits),
117                     color=color, **kwargs)
118
119
120 def main():
121     fig = plt.figure()
122     ax = LevelSurfacePlotter(fig)
123
124     x_limits = np.arange(-10, 10 + .5, .5)
125     y_limits = np.arange(-10, 10 + .5, .5)
126     C_limits = list(np.arange(100, -100 - 10, -20))
127     C_limits.remove(0)
128
129     ax.plot_level_surfaces(solve_z_prompt(), x_limits, y_limits, C_limits)
130     plt.show()
131
132
133 if __name__ == '__main__':
134     main()
```