3.

```python
#!/bin/env python3
"""Graph a family of level surfaces for a function of three variables.

Example function:
f(x,y,z)=z/sqrt(x**2+y**2)"""
import sys
import numpy as np
import sympy
from sympy.abc import x, y, z, C
from matplotlib import cm, pyplot as plt
from mpl_toolkits.mplot3d import Axes3D



def repeat(an_object):
    while True:
        yield an_object


def prompt_expression():
    """Gather a mathematical expression from the user. Raises OSError on
    data/pipe mismatch.

    $ ./graph.py  # Pass expression via interactive prompt
    f(x,y,z)=z/sqrt(x**2+y**2)

    $ ./graph.py "z/sqrt(x**2+y**2)"  # Pass expression as parameter

    $ echo "z/sqrt(x**2+y**2)" | ./graph.py -  # Pass expression over pipe"""
    if len(sys.argv) > 1:
        if sys.argv[1] == '-':
            if sys.stdin.isatty():
                raise OSError('\'-\' flag with no pipe')
            else:
                return sys.stdin.read()
        else:
            return sys.argv[1]
    elif sys.stdin.isatty():
        return input('f(x,y,z)=')
    else:
        raise OSError('pipe with no \'-\' flag')


def eval_expression(expression: str):
    """Evaluate a string expression into a sympy expression. Available
    namespace is that of sympy.* and x, y, and z variables."""
    return eval(expression, vars(sympy), {'x': x, 'y': y, 'z': z})


def solve_z(expression: sympy.Expr):
    """Solve C=f(x,y,z) for z for some sympy expression f(x,y,z). Returns a
    list of expressions with variables x, y, and C."""
    return sympy.solve(sympy.Eq(C, expression), z)


def solve_z_str(expression: str):
    """Solve C=f(x,y,z) for z for some string expression f(x,y,z)."""
    return solve_z(eval_expression(expression))


def solve_z_prompt():
    """Solve C=f(x,y,z) for z for some expression f(x,y,z) that is prompted
    for to the user."""
    return solve_z_str(prompt_expression())


def make_lambda(expression: sympy.Expr, constant):
    """Form a function of two variables from the sympy expression while
    substituting the arbitrary constant C."""
    return sympy.Lambda((x, y), expression.subs(C, constant))


# named after enumerate() but for colors
def e_color_ate(constants):
    """Generate corresponding colors for each constant. Yields
    color, constant."""
    index_max = len(constants) - 1

    for index, constant in enumerate(constants):
        color = cm.jet(index / index_max, 0.93)
        yield color, constant


def map_f_xy(f, mesh):
    """Map a numpy (x,y) meshgrid to a function of two variables."""
    domain = np.array(mesh).transpose(1, 2, 0)
    z_range = np.empty(mesh[0].shape)

    for i, domain_i in enumerate(domain):
        for j, (x_ij, y_ij) in enumerate(domain_i):
            z_ij = f(x_ij, y_ij)
            z_range[i][j] = z_ij if z_ij.is_real else np.nan

    return z_range


class LevelSurfacePlotter(Axes3D):
    """Plot functions of two variables or level curves of functions of three
    variables."""

    def plot_f_xy(self, f, mesh_domain, **kwargs):
        """Plot f(x,y) in three dimensions whereas mesh_domain is a numpy
        meshgrid of (x,y) coordinates."""
        super().plot_surface(
            *mesh_domain, map_f_xy(f, mesh_domain),
            **kwargs)

    def plot_level_surfaces(
            self, solutions: list, x_limits, y_limits, C_limits,
            **kwargs
    ):
        """Plot a number of expressions that contain x, y, and C. The
        parameters *_limits are lists of possible values."""
        for color, constant in e_color_ate(C_limits):
            for f_xy in map(make_lambda, solutions, repeat(constant)):
                self.plot_f_xy(
                    f_xy, np.meshgrid(x_limits, y_limits),
                    color=color, **kwargs)


def main():
    fig = plt.figure()
    ax = LevelSurfacePlotter(fig)

    x_limits = np.arange(-10, 10 + .5, .5)
    y_limits = np.arange(-10, 10 + .5, .5)
    C_limits = list(np.arange(100, -100 - 10, -20))
    C_limits.remove(0)

    ax.plot_level_surfaces(solve_z_prompt(), x_limits, y_limits, C_limits)
    plt.show()


if __name__ == '__main__':
    main()
```