```python
"""
The Magnetic Field from Current Distribution
"""

from math import *
from pylab import show, imshow
import numpy as np


def simpson(func, a, b, h=0.001):
    """Approximates integral using simpson method"""
    n = int(abs(b - a) / h)
    n -= 1 if n % 2 == 1 else 0
    return (h / 3.0) * (
        func(a) + func(b) +
        (4.0 * sum([func(a + (k * h)) for k in range(1, n, 2)])) +
        (2.0 * sum([func(a + (k * h)) for k in range(2, n - 1, 2)])))

def adaptive(func, a, b, h=0.001, delta=10e-4):
    """Approximates integral using adaptive method"""
    n = int(abs(b - a) / h)
    i0 = h * ((0.5 * (func(a) + func(b))) + sum(
        [func(a + k * h) for k in range(1, int((b - a) / h))]))
    epsilon = delta + 10
    while epsilon > delta:
        h /= 2
        n *= 2
        i1 = (0.5 * i0) + (h * sum([func(a + k * h) for k in range(1, n, 2)]))
        epsilon = abs(i1 - i0) / 3
        i0 = i1
    return i1


def B(x, y, z):
    func_x = lambda theta: (2 * z * sin(theta)) / pow((x - 3 * cos(theta))**2 + (y - 2 * sin(theta))**2 + z**2, 3 / 2)
    func_y = lambda theta: -(3 * z * cos(theta)) / pow((x - 3 * cos(theta))**2 + (y - 2 * sin(theta))**2 + z**2, 3 / 2)
    func_z = lambda theta: (3 * y * cos(theta) - 2 * x * sin(theta)) / pow((x - 3 * cos(theta))**2 + (y - 2 * sin(theta))**2 + z**2, 3 / 2)
    comp_x = simpson(func_x, 0, 2 * pi, 0.1)
    comp_y = simpson(func_y, 0, 2 * pi, 0.1)
    comp_z = simpson(func_z, 0, 2 * pi, 0.1)
    return (comp_x, comp_y, comp_z)

def B_adaptive(x, y, z):
    func_x = lambda theta: (2 * z * sin(theta)) / pow((x - 3 * cos(theta))**2 + (y - 2 * sin(theta))**2 + z**2, 3 / 2)
    func_y = lambda theta: -(3 * z * cos(theta)) / pow((x - 3 * cos(theta))**2 + (y - 2 * sin(theta))**2 + z**2, 3 / 2)
    func_z = lambda theta: (3 * y * cos(theta) - 2 * x * sin(theta)) / pow((x - 3 * cos(theta))**2 + (y - 2 * sin(theta))**2 + z**2, 3 / 2)
    comp_x = adaptive(func_x, 0, 2 * pi, 0.1, 10e-4)
    comp_y = adaptive(func_y, 0, 2 * pi, 0.1, 10e-4)
    comp_z = adaptive(func_z, 0, 2 * pi, 0.1, 10e-4)
    return (comp_x, comp_y, comp_z)


def main():

    def a():
        print(B(1, 4, 7))

    def b():
        print(B_adaptive(1,2,5))
    def c():
        data = []
        for y in np.arange(-5, 5, 0.1):
            row = []
            for x in np.arange(-5, 5, 0.1):
                b = B_adaptive(x, y, 1)
                row.append(np.sqrt(b[0]**2+b[1]**2+b[2]**2))
            data.append(row)
        imshow(data)
        show()

    a()
    b()
    c()


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