```
#!/usr/bin/python3

from pylab import *
from numpy import *


def simpson(func, a, b, h=0.1):
    """Approximates integral using simpson method"""
    n = int(abs(b - a) / h)
    n -= 1 if n % 2 == 1 else 0
    I1 = (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)])))
    h2 = 2 * h
    n2 = int(abs(b - a) / h2)
    n2 -= 1 if n2 % 2 == 1 else 0
    I2 = (h2 / 3.0) * (
        func(a) + func(b) +
        (4.0 * sum([func(a + (k * h2)) for k in range(1, n2, 2)])) +
        (2.0 * sum([func(a + (k * h2)) for k in range(2, n2 - 1, 2)])))
    return (I1, abs(I1 - I2) / 3)


def adaptive(input_func, a, b, h=0.1, delta=10e-6):
    """Approximates integral using adaptive method"""
    if a > b:
        val = adaptive(input_func, b, a, h=h, delta=delta)
        return (-val[0], val[1])
    if a == -inf and b == inf:
        func = lambda x: (input_func(-x/(1-x))+input_func(x/(1-x))) / pow(1-x,2)
        a = 0
        b = 1
    elif a != -inf and b == inf:
        func = lambda x: input_func((x / (1 - x)) + a) / pow(1 - x, 2)
        a = 0
        b = 1
    else:
        func = input_func
    try:
        func(a)
    except ZeroDivisionError:
        a += 0.00000000000001
    try:
        func(b)
    except ZeroDivisionError:
        b -= 0.00000000000001
    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, epsilon


def mag2(x, xp, y, yp, z, zp):
    return (x - xp)**2 + (y - yp)**2 + (z - zp)**2

```

```

def V(x, y, z, integrate=simpson):
    pre = 1 / (4 * pi)

    def func(theta):
        xp = 3 * cos(theta)
        yp = 2 * sin(theta)
        zp = 0
        return pre / sqrt(mag2(x, xp, y, yp, z, zp)) * sqrt(
            mag2(xp, 0, yp, 0, zp, 0))

    return integrate(func, 0, 2 * pi, h=0.1)


def E(x, y, z, integrate=simpson):
    pre = 1 / (4 * pi)

    def func(theta):
        xp = 3 * cos(theta)
        yp = 2 * sin(theta)
        zp = 0
        return (pre * sqrt(mag2(x, xp, y, yp, z, zp))) / pow(
            mag2(x, xp, y, yp, z, zp), 3 / 2) * sqrt(mag2(xp, 0, yp, 0, zp, 0))

    return integrate(func, 0, 2 * pi, h=0.1)


def p2():
    """The potential and electric field of a linear charge distribution"""

    def b():
        print("Potential:", V(1, 4, 7))
        print("Electric Field:", E(1, 4, 7))

    def c():
        print("Potential:", V(1, 2, 5, integrate=adaptive))
        print("ELectric Field:", E(1, 2, 5, integrate=adaptive))

    def d():
        res = 100
        scale = 3.5 / res
        data = [[V(x * scale, y * scale, 1)[0] for x in range(-res, res + 1)] for y in range(-res, res + 1)]
        imshow(data)
        show()

    b()
    c()
    d()


if __name__ == "__main__":
    p2()
```