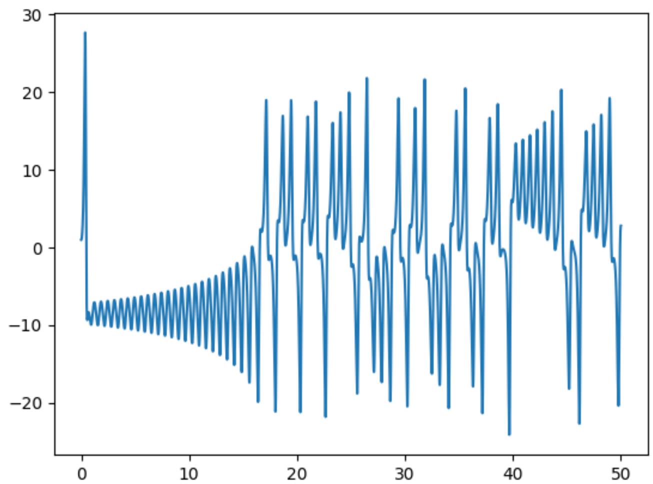
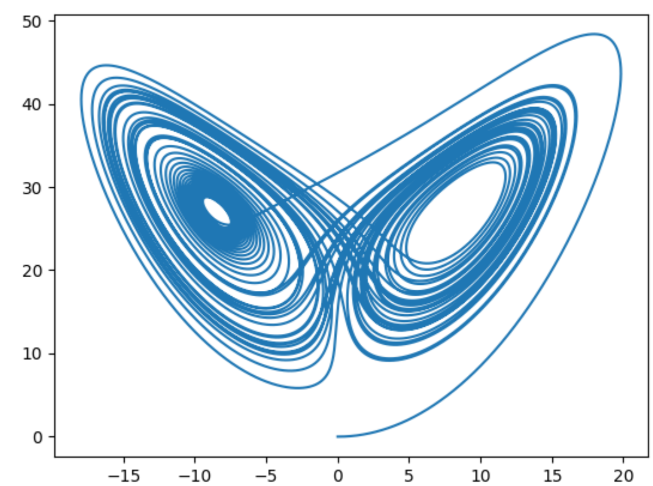
COMPUTATIONAL PHYSICS FINAL

ARDEN RASMUSSEN

1. Lorenz Attractor



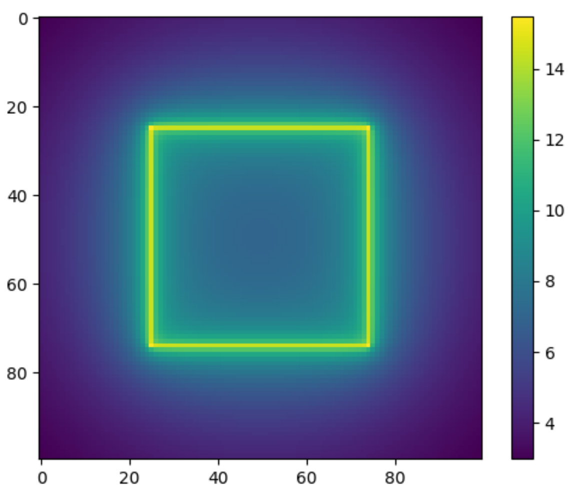


Date: December 10, 2018.

2. The Electric Potential of a Square Wire

$$V(0,0) = 7.050987223017098$$
$$V\left(\frac{1}{4},0\right) = 7.489509173680615$$

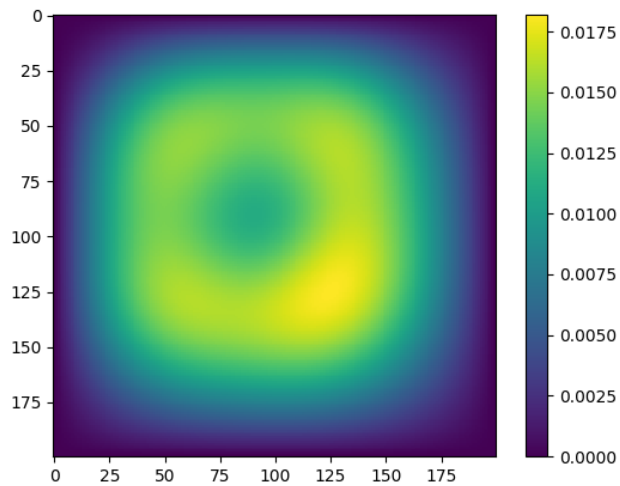
$$\vec{E}(0,0) = \begin{pmatrix} 0.0 \\ 0.0 \end{pmatrix}$$
$$\vec{E}\left(\frac{1}{4},0\right) = \begin{pmatrix} 4.3173101895277455 \\ 4.4408920985006255 \cdot 10^{-11} \end{pmatrix}$$



3. The Potential in the Presence of Boundaries

$$V(0,0) = 0.012354418658953788$$
$$V\left(\frac{1}{4},0\right) = 0.016312469694817238$$

$$\vec{E}(0,0) = \begin{pmatrix} 0.011806814924191639 \\ 0.011806814924191465 \end{pmatrix}$$
$$\vec{E}\left(\frac{1}{4},0\right) = \begin{pmatrix} 0.011079095703054806 \\ 0.009159891802292333 \end{pmatrix}$$


0
25
50
75
100
125
150
175
0 25 50 75 100 125 150 175
0.0175
0.0150
0.0125
0.0100
0.0075
0.0050
0.0025
0.0000

Appendix A. Lorenz Attractor

```python
import numpy as np
import pylab

def RungeKutta(f1, f2, f3, a_init, b_init, c_init, t0, tf, h=0.1):
    A = []
    B = []
    C = []
    T = np.arange(t0, tf, h)
    a = a_init
    b = b_init
    c = c_init
    for t in T:
        A.append(a)
        B.append(b)
        C.append(c)
        k1 = h * f1(a, b, c, t)
        l1 = h * f2(a, b, c, t)
        m1 = h * f3(a, b, c, t)
        k2 = h * f1(a + k1 / 2, b + l1 / 2, c + m1 / 2, t + h / 2)
        l2 = h * f2(a + k1 / 2, b + l1 / 2, c + m1 / 2, t + h / 2)
        m2 = h * f3(a + k1 / 2, b + l1 / 2, c + m1 / 2, t + h / 2)
        k3 = h * f1(a + k2 / 2, b + l2 / 2, c + m2 / 2, t + h / 2)
        l3 = h * f2(a + k2 / 2, b + l2 / 2, c + m2 / 2, t + h / 2)
        m3 = h * f3(a + k2 / 2, b + l2 / 2, c + m2 / 2, t + h / 2)
        k4 = h * f1(a + k3, b + l3, c + m3, t + h)
        l4 = h * f2(a + k3, b + l3, c + m3, t + h)
        m4 = h * f3(a + k3, b + l3, c + m3, t + h)
        a += (k1 + 2 * k2 + 2 * k3 + k4) / 6
        b += (l1 + 2 * l2 + 2 * l3 + l4) / 6
        c += (m1 + 2 * m2 + 2 * m3 + m4) / 6
    return T, A, B, C

def main():
    sigma = 10
    r = 28
    b = 8/3
    dx = lambda x, y, z, t: sigma*(y-x)
    dy = lambda x,y,z,t: r*x-y-x*z
    dz = lambda x,y,z,t: x*y-b*z
    T, X, Y, Z = RungeKutta(dx, dy, dz, 0, 1, 0, 0, 50, 0.0001)
    pylab.plot(T, Y)
    pylab.savefig("P1_a.png")

    pylab.clf()
    pylab.plot(X,Z)
    pylab.savefig("P1_b.png")

```

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

Appendix B. The Electric Potential of a Square Wire

```python
import numpy as np
import pylab

def integrate(func, a, b, h=1e-2, delta=1e-6):
    """Approximates integral using adaptive method"""
    if a > b:
        val = integrate(func, b, a, h=h, delta=delta)
        return (-val[0], val[1])
    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 rho(r):
    if (r[0] == 0.5 or r[0] == -0.5) and -0.5 <= r[1] <= 0.5:
        return 1
    elif (r[1] == 0.5 or r[1] == -0.5) and -0.5 <= r[0] <= 0.5:
        return 1
    return 0

def p1(x1,x2, delta=1e-6):
    r = np.array([x1,x2])
    func_a = lambda x: 1.0/(np.linalg.norm(r-np.array([x,-0.5])))
    func_b = lambda y: 1.0/(np.linalg.norm(r-np.array([-0.5,y])))
    func_c = lambda x: 1.0/(np.linalg.norm(r-np.array([x,0.5])))
    func_d = lambda y: 1.0/(np.linalg.norm(r-np.array([0.5,y])))
    int_a = integrate(func_a, -0.5, 0.5, delta=delta)
    int_b = integrate(func_b, -0.5, 0.5, delta=delta)
    int_c = integrate(func_c, -0.5, 0.5, delta=delta)
    int_d = integrate(func_d, -0.5, 0.5, delta=delta)
    return int_a + int_b + int_c + int_d

def p2():
    print("V(0,0) = {}".format(p1(0,0)))
    print("V(0.25,0) = {}".format(p1(1/4,0)))

def p3():
    def e(x1,x2,h=1e-5,delta=1e-6):
        ex = (p1(x1 + h, x2) - p1(x1-h, x2))/(2*h)
```

```
        ey = (p1(x1, x2+h) - p1(x1, x2-h))/(2*h)
        return [ex, ey]

    print("E(0,0) = {}".format(e(0,0)))
    print("E(0.25,0) = {}".format(e(0.25,0)))


def p4():
    steps = 100
    data = [[p1(x,y, 1e-3) for x in np.linspace(-1,1, steps)] for y
    ↪  in np.linspace(-1,1, steps)]
    pylab.imshow(data)
    pylab.colorbar()
    pylab.savefig("P2_4.png")

def main():
    p2()
    p3()
    p4()
    pass

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

Appendix C. The Potential in the Presence of Boundaries

```python
import numpy as np
import pylab

a = 1

def rho(r):
    if (r[0] == 0.5 or r[0] == -0.5) and -0.5 <= r[1] <= 0.5:
        return 1
    elif (r[1] == 0.5 or r[1] == -0.5) and -0.5 <= r[0] <= 0.5:
        return 1
    return 0

def gauss_seidel(phi, rho, w, tol):
    error = tol + 1000
    N = phi.shape[0]
    phi_min = 0
    while error > tol:
        diff_max = 0
        for i in range(N):
            for j in range(N):
                if i == 0 or i == N - 1 or j == 0 or j == N - 1:
                    phi[i, j] = phi[i, j]
                else:
                    old = phi[i,j]
                    phi[i, j] = (1 + w) / 4 * (
                        phi[i + 1, j] + phi[i - 1, j] + phi[i, j + 1] +
                        phi[i, j - 1]) - w * phi[i, j] + (a**2 / 4) * rho([i, j])
                    diff_max = max(diff_max, np.abs(phi[i,j] - old))
        error = diff_max
    return phi

def main():
    phi = np.zeros([200, 200])
    for i in range(50,150):
        phi[50,i] = 1
        phi[150,i] = 1
        phi[i,50] = 1
        phi[i,150] = 1
    phi = gauss_seidel(phi, rho, 0.8, 1e-4)

    print("V(0,0) = {}".format(phi[100, 100]))
    print("V(0.25,0) = {}".format(phi[125,100]))

    pylab.imshow(phi)
    pylab.colorbar()
```

```
    pylab.savefig("P3_3.png")

    def e(x1,x2, h = 2/200):
        ex = (phi[x1+1, x2] - phi[x1-1,x2])/(2*h)
        ey = (phi[x1, x2+1] - phi[x1,x2-1])/(2*h)
        return [ex, ey]
    print("E(0,0) = {}".format(e(100,100)))
    print("E(0.25,0) = {}".format(e(125,100)))

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