COGSCI131–Spring 2019 — Homework 2Solutions

Yuchen Zhou, SID 3034489901

1.

(a) Suppose that an ant wandered randomly by taking steps (x,y), one per second, where at each ant step, x and y come from a normal distribution with a mean of 0 and a standard deviation of 1.0mm (assume this for all questions below). Plot a trace of the ants path over the course of an hour.

- use numpy library to simulate this movement:

```
import numpy
import matplotlib.pyplot as plt

fig=plt.figure()
ax=fig.add_subplot(111)
dx=numpy.random.normal(0,1,3600)
dy=numpy.random.normal(0,1,3600)
x=[0]*3600
y=[0]*3600
for i in range(3600):
    x[i]=dx[i-1]+x[i-1]
    y[i]=dy[i-1]+y[i-1]
ax.scatter(x,y,color="black",marker=".")
#ax.plot(x,y)
plt.savefig("1aa.png")
plt.show()
```

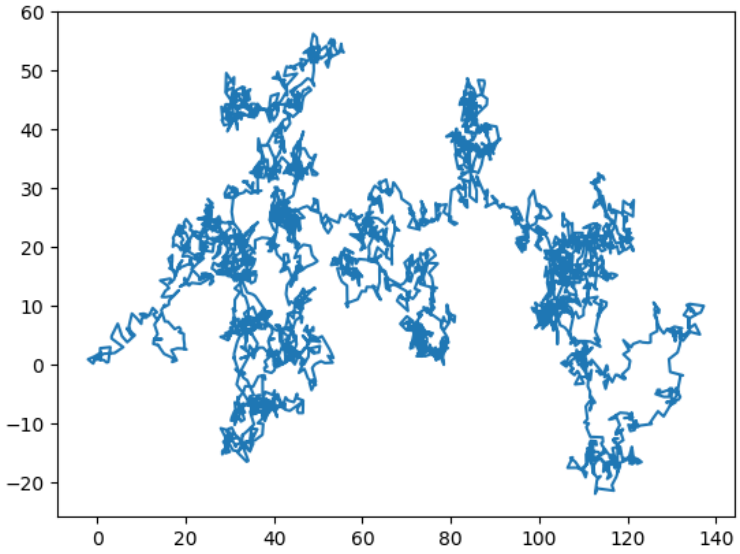

Figure 1: Ant's path in an hour

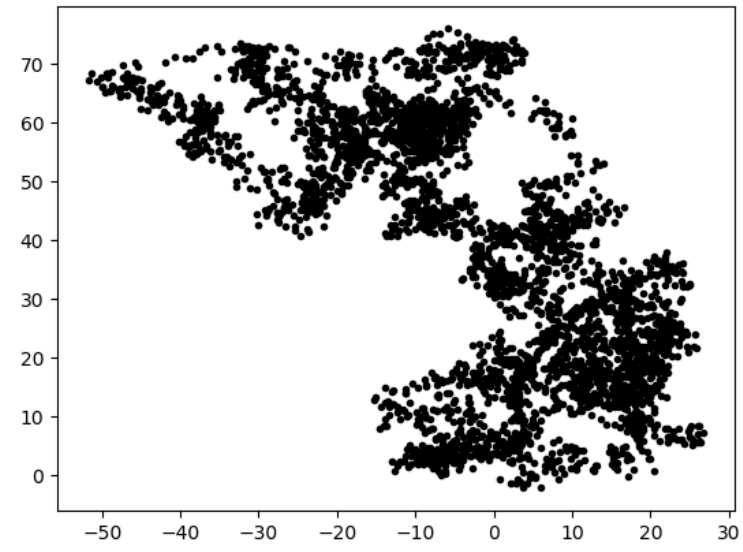

Figure 2: Ant's path in an hour(discrete knots)

(b) Lets think about why ants need to perform path integration. Suppose that instead of path integration, when an ant found food, it just continued to wander with random steps until it got back to the nest. Using a simulation, find the probability that an ant who finds food after 1 hour will make its way back to within 10mm of the nest over the course of the next hour (note that if it comes within 10mm of a nest, it stops). Is this a good strategy? Why or why not?

- assume the position that the ant wanders after one hour as the food position
- wandering for another hour or till ant reaches the nest
- simulate for 50000 times

```
import numpy

k=50000
p=0
for kk in range(k):
    dx1=numpy.random.normal(0,1,3600)
    dx2=numpy.random.normal(0,1,3600)
    dy1=numpy.random.normal(0,1,3600)
    dy2=numpy.random.normal(0,1,3600)
    x1=[0]*3601
    y1=[0]*3601
    x2=[0]*3601
    y2=[0]*3601
    for i in range(3600):
        x1[i+1]=dx1[i]+x1[i]
        y1[i+1]=dy1[i]+y1[i]
    x2[0]=x1[-1]
    y2[0]=y1[-1]
    #print(x2[0],y2[0])
    for i in range(3600):
        if(x2[i]**2+y2[i]**2>100):
            x2[i+1]=dx2[i]+x2[i]
            y2[i+1]=dy2[i]+y2[i]
        else:
            p+=1
            break
print(p/k)
```

- The probability that an ant who finds food after 1 hour will make its way back to within 10mm of the nest over the course of the next hour is 20.734%
- It's a really bad strategy. Paying an hour to go home is bad, and it's worth to find that chances to get back successfully is only 20%.

(c) If the ant searches for an hour, finds food, and then searches for the nest by continuing to walk at random, what is the average closest distance it will come to the nest? Find this with a simulation.

- simulate for 10000 times

```
import numpy

k=10000
meannear=0;
for kk in range(k):
    dx1=numpy.random.normal(0,1,3600)
    dx2=numpy.random.normal(0,1,3600)
    dy1=numpy.random.normal(0,1,3600)
    dy2=numpy.random.normal(0,1,3600)
    x1=[0]*3601
    y1=[0]*3601
    x2=[0]*3601
    y2=[0]*3601
    for i in range(3600):
        x1[i+1]=dx1[i]+x1[i]
        y1[i+1]=dy1[i]+y1[i]
    x2[0]=x1[-1]
    y2[0]=y1[-1]
    nearest=pow(x2[0]**2+y2[0]**2,0.5)
    for i in range(3600):
        x2[i+1]=dx2[i]+x2[i]
        y2[i+1]=dy2[i]+y2[i]
        if pow(x2[i+1]**2+y2[i+1]**2,0.5)>nearest:
            nearest=pow(x2[i+1]**2+y2[i+1]**2,0.5)
    meannear+=nearest
print(meannear/k)
```

- Mean distance to nest is 138.37.

2.

Now lets think about path integration. Assume that each step (x,y) is remembered (integrated) internally in the ants brain with a standard deviation on each component of S. Thus, if we store the total X component, it gets updated with a new x step via X X+x+ex where ex Gaussian(0,S) and similarly for Y (Y y + ey with ey Gaussian(0,S)). Suppose that, upon finding food after an hour (as above, one step per second), the ant then heads straight back to where it thinks the nest is (e.g. it travels back via the vector (-X,-Y)). Thus, the outbound trip is noisy, but the return trip is noiseless. Run a simulation to see how far the ant will end from the nest for various S from 1.0mm down to 0.0001mm. Plot the mean distance the ant ends from the nest as a function of S. Be sure to show a range of S values that make it clear whats going on.

- Precise part of x and y can actually be ignored:

$$X = \Sigma_i(x_i + ex_i) \tag{1}$$
$$X_{precise} = -\Sigma_i x_i \tag{2}$$
$$\therefore Dis = 0 - (X + X_{precise}) = \Sigma_i ex_i \tag{3}$$

  Thus, distance to nest is just the sum of the noise.

- Similar situation to y.
- Simulate mean distance to s for 50 times each, plot distance to log10(s) from s=0.0001 to s=10 by step length 0.01:

```
import numpy
import matplotlib.pyplot as plt
import math
def dis(s):
    ddis=0;
    for i in range(50):
        dx=numpy.random.normal(0,s,3600)
        dy=numpy.random.normal(0,s,3600)
        x=0
        y=0
        for i in range(3600):
            x=dx[i]+x
            y=dy[i]+y
        ddis+=pow(x**2+y**2,0.5)
    return ddis/50
distance=[]
fig=plt.figure()
ax=fig.add_subplot(111)
i=[-4]
distance.append(dis(10 ** i[-1]))
while i[-1]<=1:
    i.append(i[-1]+0.01)
    distance.append(dis(10 ** i[-1]))
ax.plot(i,distance)
plt.ylabel('Distance to nest')
plt.xlabel('log10(s)')
plt.show()
```

- Distance to nest grows incredibly fast when s increases.

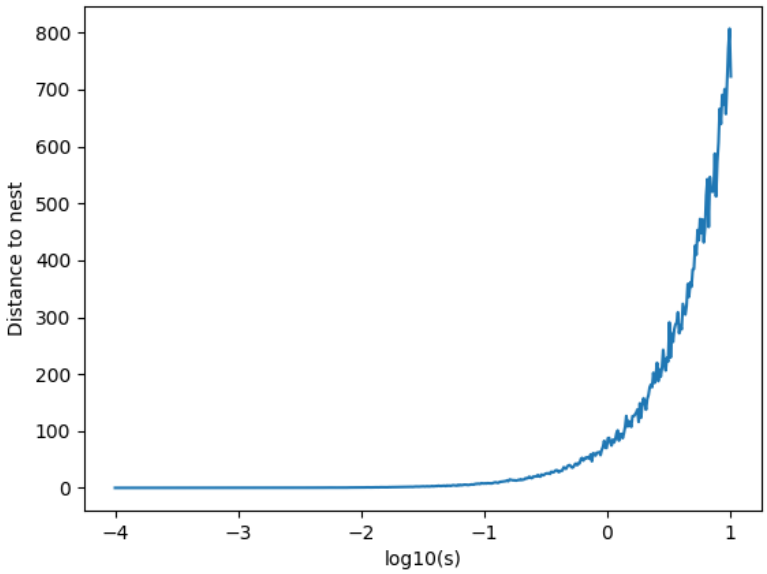

Figure 3: Distance to nest varies to log10(s)

3.

(a) Next, lets just assume that it requires exp(0.1/S) energy units to run an integrator with a standard deviation of S for an hour. Suppose further that if you end up at a distance d from the nest after your return trip, it will take you d2 energy units to find the nest. By using a simulation, plot the average energy expended while on a foraging trip (out for an hour and back) as a function of S. Be sure you have found a range of S to plot that shows the shape of the curve near its minimum.

- Energy can be written as

$$E = e^{\frac{0.1}{s}} + d^2$$

Simulate from $log_{10}(s) = -3.5$ to $log_{10}(s) = 5$ by step length 0.01:

```
import numpy
import matplotlib.pyplot as plt
import math

def E(s):
    return math.exp(0.1/s)+dis(s)**2

def dis(s):
    ddis=0;
    for i in range(10):
        dx=numpy.random.normal(0,s,3600)
        dy=numpy.random.normal(0,s,3600)
        x=0
        y=0
        for i in range(3600):
            x=dx[i]+x
            y=dy[i]+y
        ddis+=pow(x**2+y**2,0.5)
    return ddis/10

energy=[]
fig=plt.figure()
ax=fig.add_subplot(111)
i=[-3]
energy.append(math.log10(E(10 ** i[-1])))
while i[-1]<=0:
    i.append(i[-1]+0.01)
    energy.append(math.log10(E(10 ** i[-1])))
ax.plot(i,energy)
plt.ylabel('log10(Energy)')
plt.xlabel('log10(s)')
plt.show()
```

- Plot figure, take y as log10(Energy) and x as log10(s).
- Then focus on curve in range of $log_{10}(s) = -3$ to $log_{10}(s) = 0$
- Find minimum Energy on curve to be about $10^{1.13} = 0.074$ when $s = 10^{-1.31} = 0.049$

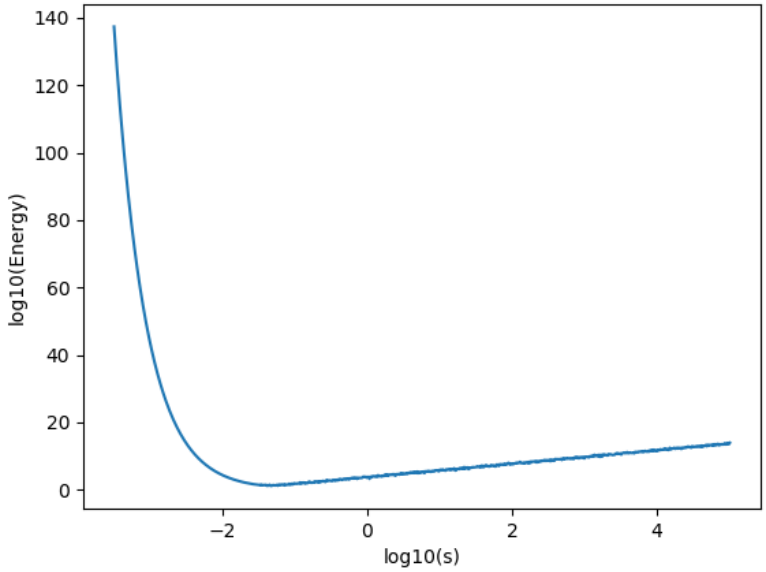

Figure 4: log10(Energy) to log10(s) curve

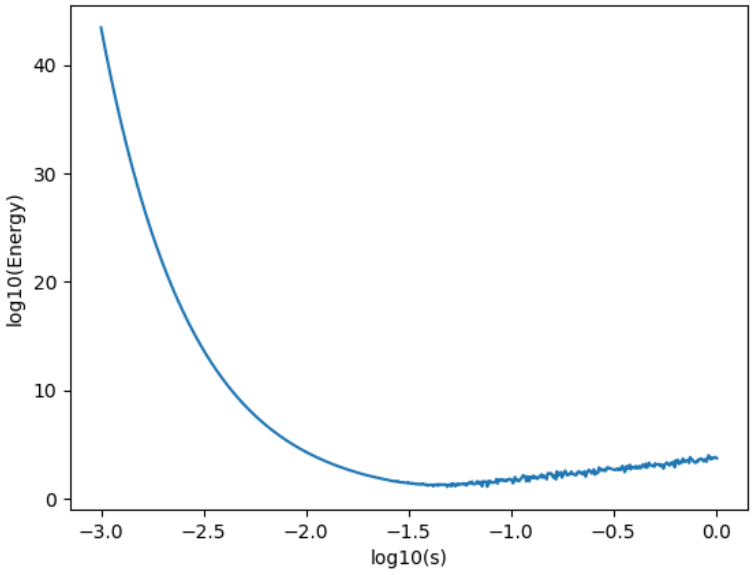

Figure 5: log10(Energy) to log10(s) curve

(b) What is the evolutionary significance of the minimum of the plot in 3a?

- It turns out that there is a optimal point to find food with minimal energy.
- Also, the second derivative near that point is less than 0, which suggest that it's a steady point. It's reachable.
- We can infer that ants will gradually develop an accurate memory of each step with a standard deviation close to 10 to the point's x coordinate, which saves the most of energy.