# HW_3

February 12, 2020

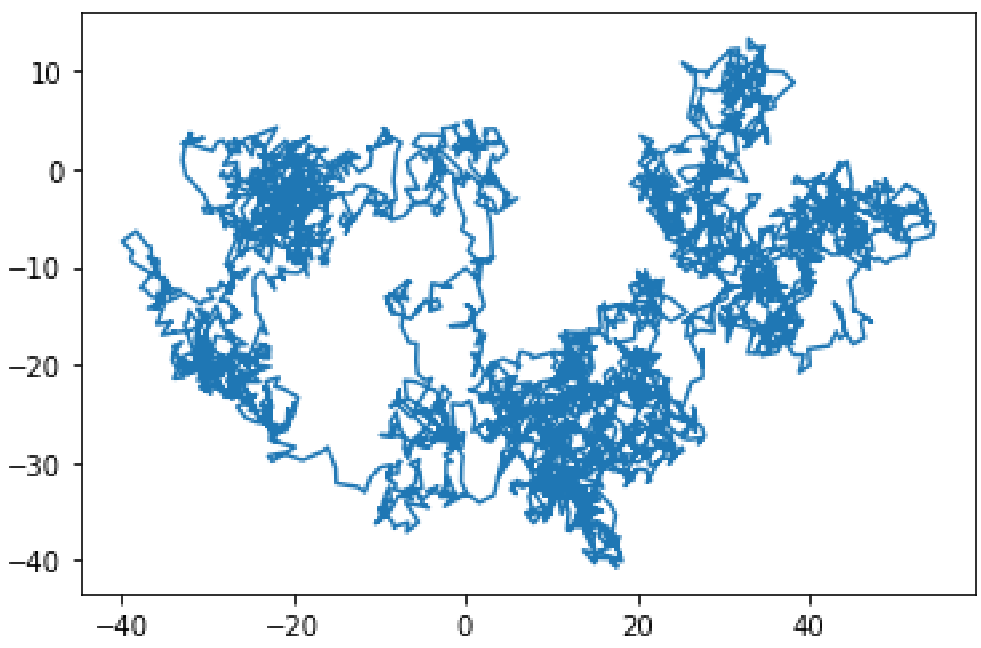
[3]:
```python
import matplotlib.pyplot as plt
import numpy as np
import random
```

## 1 QUESTION 1A

[ ]:
```python

```

[179]:
```python
x_path = np.cumsum(np.random.normal(0, 1, 3600))
y_path = np.cumsum(np.random.normal(0, 1, 3600))



plt.plot(x_path, y_path)
plt.show()
```

```
[ 0.4947312   0.64337701  1.8537653  … 34.13736172 35.06760611
 34.60323712]
```



## 2 QUESTION 1B

[206]:
```
def wander(start_x, start_y):
    output_x = start_x + np.cumsum(np.random.normal(0, 1.0, 3600))
    output_y = start_y + np.cumsum(np.random.normal(0, 1.0, 3600))
    return output_x, output_y



def wander_back():
    distances = []
    endpoint = wander(0, 0)
    return_trip = wander(endpoint[0][-1], endpoint[1][-1])
    for i, j in zip(return_trip[0], return_trip[1]):
        distance = np.sqrt(((i - 0)**2) + ((j - 0)**2))
        distances.append(distance)
    if min(distances) < 5:
        return True
    else:
        return False



def sim(trials):
    count = 0
    success = 0
    while count < trials:
        if wander_back():
            success += 1
        count += 1
    return (success / trials) * 100

sim(1000)
```

[206]:
```
15.2
```

The simulation trials show us that random wandering is quite inefficient, with roughly a **15%** chance of returning within 5.0mm of the nest during the next hour after finding food.

## 3 QUESTION 1C

[177]:
```
def wander_close(trials):
    counter = 0
    min_vals = []
    while counter < trials:
        values = []
        endpoint = wander(0, 0)
        return_trip = wander(endpoint[0][-1], endpoint[1][-1])
        for i, j in zip(return_trip[0], return_trip[1]):
            distance = np.sqrt((i - 0)**2 + (j - 0)**2)
            values.append(distance)
        min_vals.append(min(values))
        counter += 1
    return min_vals

minimums = wander_close(1000)
np.mean(minimums)
```

[177]:
```
47.37273553607588
```

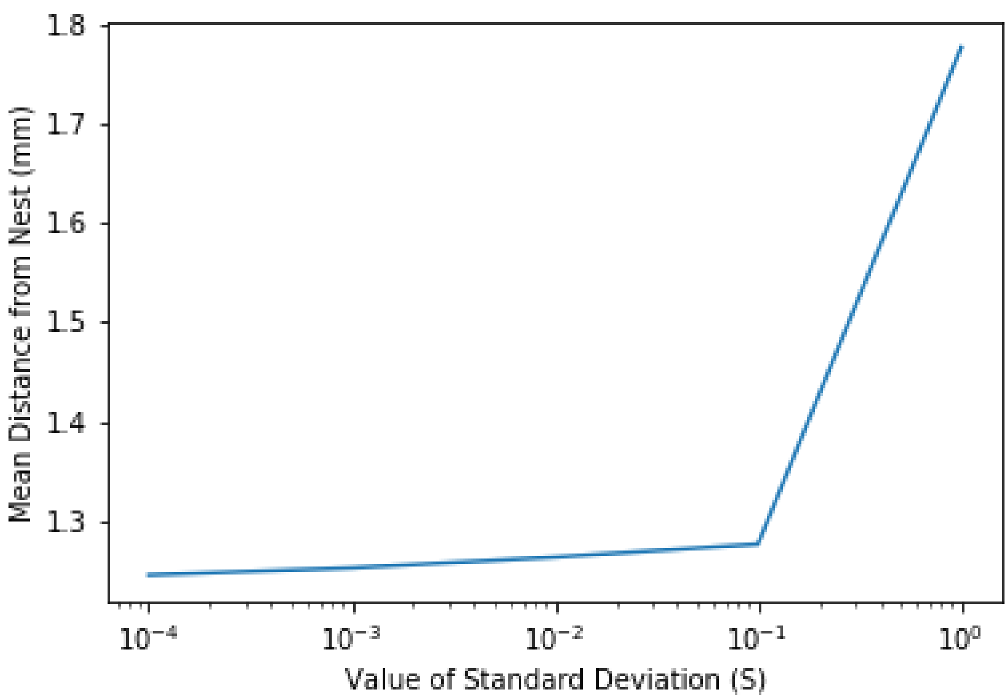
After a simulation of **1000 trials**, the ant comes a mean distance of only ~ **45mm** from the nest.

## 4 QUESTION 2

[280]:
```
def distance(x2, x1, y2, y1):
    return np.sqrt((x2 - x1)**2 + (y2 - y1)**2)



def wander_back_integrated(s):
    master = []
    for k in s:
        distances = []
        x_comp = [0]
        y_comp = [0]
        food = wander(0, 0)
        new_x, new_y = food[0], food[1]
        for i, j in zip(new_x, new_y):
            x_comp.append(i + np.random.normal(0, k))
            y_comp.append(j + np.random.normal(0, k))
        for w, x, y, z in zip(new_x, x_comp, new_y, y_comp):
            dist = distance(w, x, y, z)
            distances.append(dist)
        master.append(np.mean(distances))
    return master
```

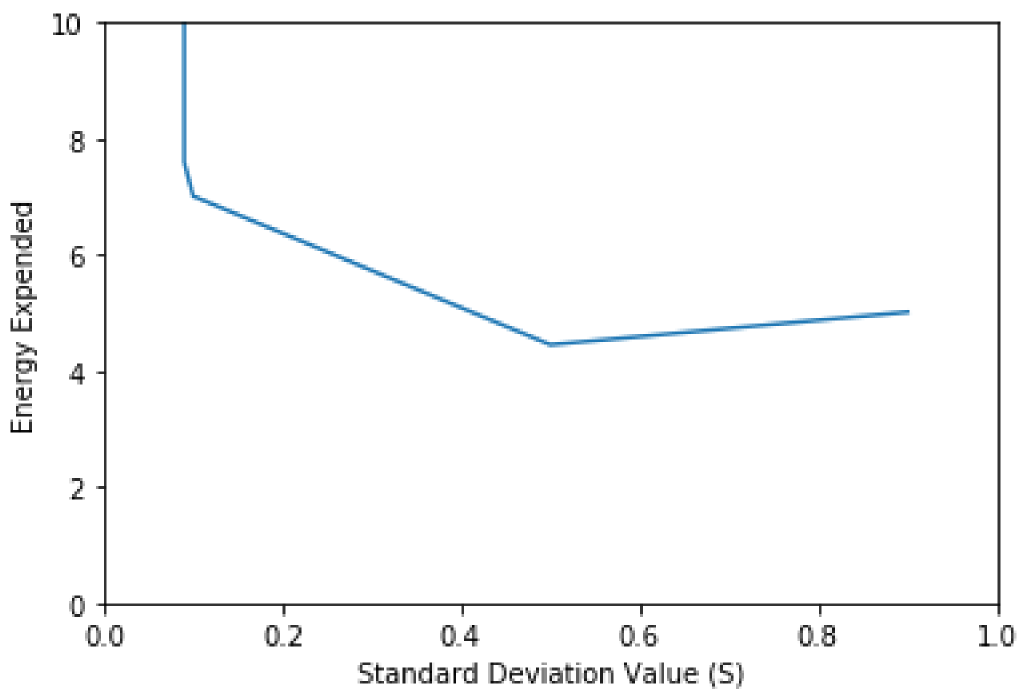
```
dat = [1, 0.1, 0.01, 0.001, 0.0001]
x = wander_back_integrated(dat)

plt.plot(dat, x)
plt.xscale("log")
plt.xlabel("Value of Standard Deviation (S)")
plt.ylabel("Mean Distance from Nest (mm)")
```

[280]: Text(0, 0.5, 'Mean Distance from Nest (mm)')



# 5 QUESTION 3A

[316]:
```
dat = [0.9, 0.5, 0.1, 0.09, 0.009, 0.0009]
integrator_cost = []
for x in dat:
    integrator_cost.append(2*(np.exp(0.1/x)))
x = wander_back_integrated(dat)
for i in range(0, len(x)):
    integrator_cost[i] += (x[i]**2)

plt.plot(dat, integrator_cost)
plt.ylim(0, 10)
plt.xlim(0, 1)
plt.xlabel("Standard Deviation Value (S)")
plt.ylabel("Energy Expended")
plt.show()
```




# 6 QUESTION 3B

The global minimum of the plot describes the optimization point in which an ant spends the least amount of energy to run an internal path integrator. Over several generations, this minimum was likely selected for within the population as some ants consumed too much energy achieving fine-precision navigation, and other ants consumed too little energy and were unable to navigate back to their nest.

```
[ ]:
```