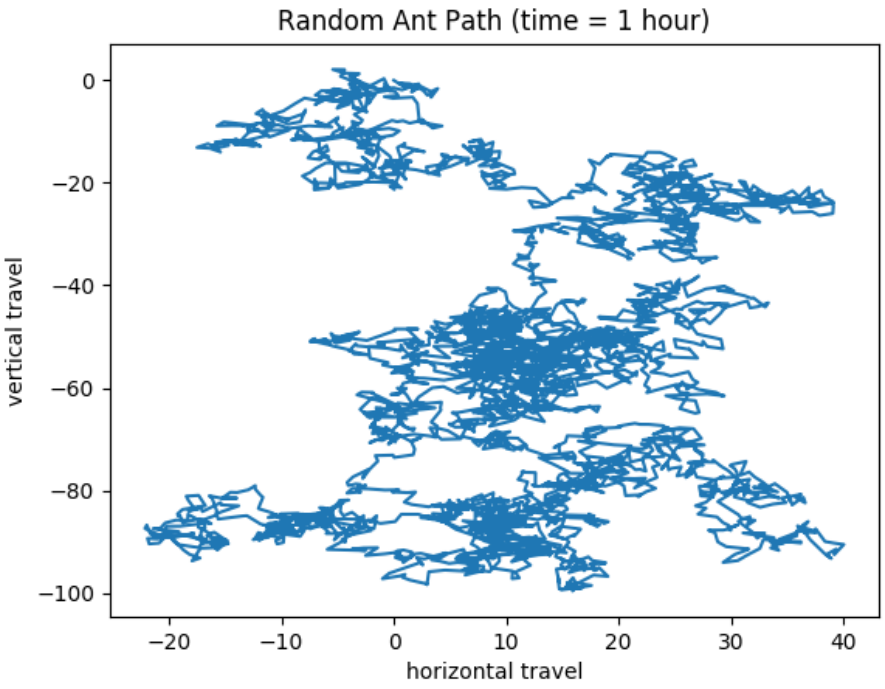
Problem #1a

```
#1a

def NorNum(sd):
    return np.random.normal(0, sd)

def randomAntPath(time):
    x = [0] * time
    y = [0] * time
    for i in range(1, time):
        xCh = NorNum(1)
        yCh = NorNum(1)
        y[i] = y[i - 1] + yCh
        x[i] = x[i - 1] + xCh
    return (x, y)

both = randomAntPath(3600)
x = both[0]
y = both[1]
plt.plot(x, y)
plt.title('Random Ant Path (time = 1 hour)')
plt.show()
```



(assume in the above plot that the ant begins at the origin (0, 0) and the axes represent horizontal and vertical travel)

Problem #1b

```
#1b

def distance(point1, point2):
    d = (point2[0] - point1[0]) ** 2
    d = d + ((point2[1] - point1[1]) ** 2)
    d = math.sqrt(d)
    return d

def antPathwithCheck(time, startPoint):
    x = [0] * time
    y = [0] * time
    x[0] = startPoint[0]
    y[0] = startPoint[1]
    for i in range(1, time):
        xCh = NorNum(1)
        yCh = NorNum(1)
        y[i] = y[i - 1] + yCh
        x[i] = x[i - 1] + xCh
        if (distance((x[i],y[i]), (0, 0))) <= 5:
            return 1
    return 0

def runSimulations(trials, time):
    successes = 0
    for i in range(0, trials):
        endpoint = randomAntPath(time)
        endpoint = (endpoint[0][time - 1], endpoint[1][time - 1])
        successes += antPathwithCheck(time, endpoint)
    return (successes / trials)

print(runSimulations(10000, 3600)) # return ~0.142 probability
```

When I run 10,000 simulations of the ant trying to randomly find its way back home, it consistently returns that the ant has around a 14.2% chance of finding its nest. I had to run 10,000 simulations in order to ensure the probability I was getting wasn't being affected by outliers (such as simulations in which the ant finds its nest surprisingly quickly, just by chance) and returned an accurate probability for the ant to find its nest. Obviously, as one could have guessed, this method for finding the nest is not a very efficient one, and the low probability of 14.2% reflects that. In this scenario, all the times when the ant does successfully find its nest are purely due to chance, and over the course of the hour that the ant walks, this chance that the ant happens to find its nest becomes less and less dependable.

Problem #1c

```python
#1c

def closestDistance(time, startPoint):
    x = [0] * time
    y = [0] * time
    x[0] = startPoint[0]
    y[0] = startPoint[1]
    dist = distance((x[0], y[0]), (0, 0))
    for i in range(1, time):
        xCh = NorNum(1)
        yCh = NorNum(1)
        y[i] = y[i - 1] + yCh
        x[i] = x[i - 1] + xCh
        currDist = distance((x[i],y[i]), (0, 0))
        if (currDist < dist):
            dist = currDist
    return dist

def runDistSimulations(trials, time):
    total = 0
    for i in range(0, trials):
        endpoint = randomAntPath(time)
        endpoint = (endpoint[0][time - 1], endpoint[1][time - 1])
        total += closestDistance(time, endpoint)
    return (total / trials)

print(runDistSimulations(10000, 3600)) # returns ~44.7 mm
```

When I run 10,000 simulations, the average distance the ant ends up from the nest after walking randomly on its return journey comes out to around 44.7 mm.

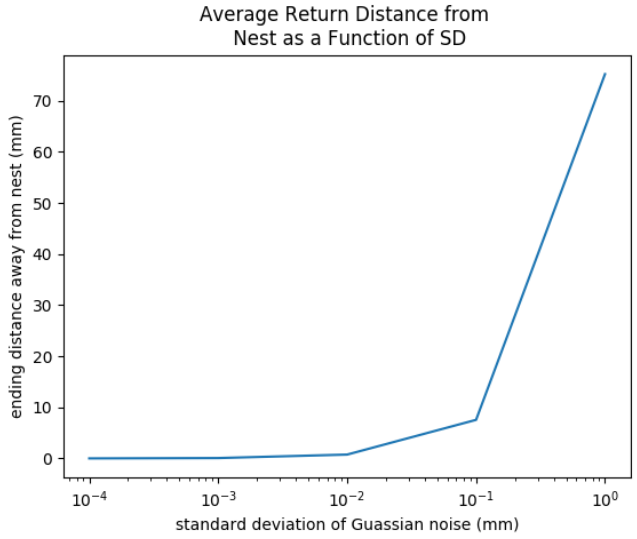
Problem #2

```python
#2

def walkAndRemember(sd, time):
    x = [0] * time
    y = [0] * time
    bigX = 0
    bigY = 0
    for i in range(1, time):
        xCh = NorNum(1)
        yCh = NorNum(1)
        y[i] = y[i - 1] + yCh
        x[i] = x[i - 1] + xCh
        bigX += (xCh + NorNum(sd))
        bigY += (yCh + NorNum(sd))
    return [bigX, bigY, (x[time - 1], y[time - 1])]

def runIntegSimulations(sd, time, trials):
    amountOff = 0
    for i in range(0, trials):
        walkData = walkAndRemember(sd, time)
        endpointX, endpointY = walkData[2][0], walkData[2][1]
        returnX, returnY = endpointX - walkData[0], endpointY - walkData[1]
        amountOff += distance((0, 0), (returnX, returnY))
    return (amountOff / trials)

x = [0.0001, 0.001, 0.01, 0.1, 1.0]
y = [runIntegSimulations(i, 3600, 10000) for i in x]
for i in range(0, len(x)):
    print("SD: ", x[i], " Dist from nest: ", y[i])
plt.title('Average Return Distance from \n Nest as a Function of SD')
plt.xlabel('standard deviation of Guassian noise (mm)')
plt.ylabel('ending distance away from nest (mm)')
plt.semilogx(x, y)
plt.show()
```




```
SD:  0.0001  Dist from nest:  0.0074886522090233144
SD:  0.001  Dist from nest:  0.07503221283716797
SD:  0.01  Dist from nest:  0.7554202383847325
SD:  0.1  Dist from nest:  7.55436644126971
SD:  1.0  Dist from nest:  75.23843910557167
```

(recorded average distance with respect to simulated standard deviation values)

Problem #3a

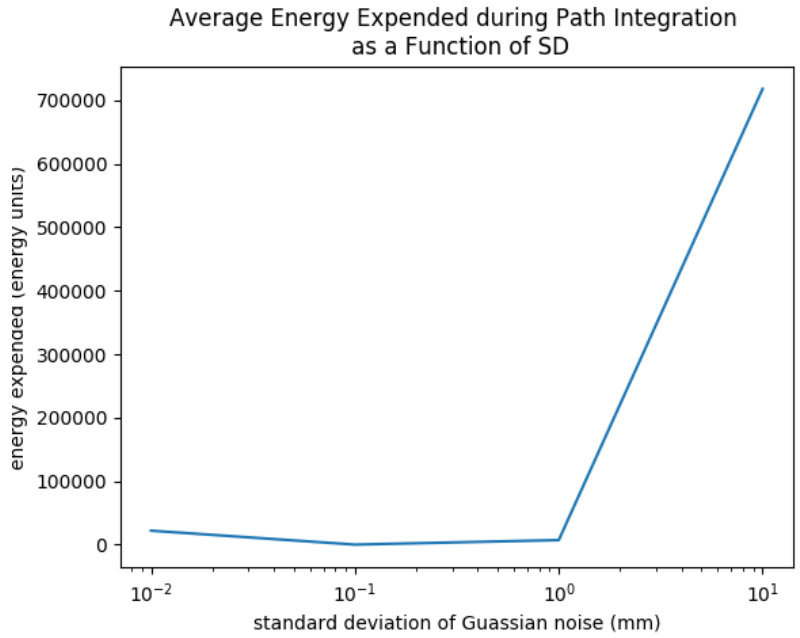
```python
#3a

def outboundEnergy(sd):
    return math.exp(0.1/sd)

def inboundEnergy(xChange, yChange, endpoint):
    returnPoint = (endpoint[0] - xChange, endpoint[1] - yChange)
    dist = distance((0, 0), returnPoint)
    return dist ** 2

def runEnergySimulations(sd, time, trials):
    energy = 0
    for i in range(0, trials):
        energy += outboundEnergy(sd)
        walkData = walkAndRemember(sd, time)
        energy += inboundEnergy(walkData[0], walkData[1], walkData[2])
    return (energy / trials)

x = [0.01, 0.03, 0.05, 0.07, 0.1, 2.0, 3.0]
y = [runEnergySimulations(i, 3600, 1000) for i in x]
plt.title('Average Energy Expended during Path Integration \n as a Function of SD')
plt.xlabel('standard deviation of Guassian noise (mm)')
plt.ylabel('energy expended (energy units')
plt.plot(x, y)
plt.show()
```




```
SD:  0.01  Energy:  22027.20310717755
SD:  0.1  Energy:  74.43853505160338
SD:  1.0  Energy:  7193.938816927759
SD:  10  Energy:  718218.1209458319
```

(recorded average energy used with respect to simulated standard deviation values)

Problem #3b

In talking about the evolutionary significance of how much energy an ant spends during its travels, we consider tradeoffs. The amount of energy the ant expends is the sum of the energy the ant spends outbound and inbound on its trip. The outbound energy is dependent on the standard deviation used, and as it gets smaller that energy usage grows bigger. The inbound energy is dependent on how far the ant is from the ant, which will be smaller with a smaller standard deviation. So there are two options here essentially: spend more energy on the outbound trip and end up closer to the nest (with a smaller energy exertion on the inbound trip), or spend less energy on the outbound trip and end up further from the nest (with a bigger energy exertion on the inbound trip).

From our graph, we can see a clear minimum as to where we get the least energy expenditure (around 0.1 for standard deviation). However, we can notice that decreasing the standard deviation by degrees of 10 has much less impaction energy expended than increasing the standard deviation by degrees of 10. Evolutionarily, this makes sense, since decreasing our standard deviation ensures a higher likelihood that the ant finds it nest. An ant not being able to find its nest after a journey to find food would not be beneficial for that ant's survival, and so allowing the ant to expend a little more energy for the outward journey ensure the ant has to spend much less energy on the journey back (and therefore finds its nest more easily). In other words, this model (in 3a) shows the evolutionary advantage of having a smaller standard deviation for the Guassian noise as the ant travels outwards, which does increase the amount of energy it expends outbound, but the tradeoff for this is massively beneficial in that it allows the ant to return to its nest much more easily with less energy exertion.