**nLab 05 – Scientific simulations**

**COMP130 - Introduction to Computing**

**Dickinson College**

**Instructor: John MacCormick**

**An Empirical Scientific Question**

Suppose a person walks by choosing a new direction at random for every step they take. How far away from their starting point will they be after 100 steps? After 1000 steps? After 10,000 steps?

**Brownian motion**

Brownian motion is named after Robert Brown, a botanist, who in 1827 described particles of plant pollen moving around randomly in water and seemingly of their own volition. Over 75 years later, Albert Einstein hypothesized that these pollen particles were actually moving the way they did because they were being bounced around by collisions with water molecules.

**Random Walk Algorithms**

Computer simulations often model Brownian motion as a random walk using the following algorithm:

repeat some number of times:

 turn in a random direction

 take one step forward

Above is an example of a random walk using a Python Turtle object. The Turtle started its walk at the red dot and took 500 random steps, ending in the lower left, where the turtle icon is now. The blue line between the turtle and the red dot is the *straight-line distance* of its random walk. For such a simple process, there is an amazing variety of applications that use random walk algorithms. A few examples include:

* Models of the stock market are based on the [Random Walk Hypothesis,](https://en.wikipedia.org/wiki/Random_walk_hypothesis) which posits that stock prices in an efficient market can be modeled as a random walk.
* Biologists who study population genetics have modeled [Genetic Drift](https://en.wikipedia.org/wiki/Genetic_drift)using random walks.
* The [Who-To-Follow (WTF)](https://www.researchgate.net/publication/262318036_WTF_the_who_to_follow_service_at_Twitter) algorithm of the social media platform X uses random walks as one part of how it recommends users you might be interested in following.
* Astronomers use random walks to model [the movement of light photons in the sun](http://www.astronoo.com/en/articles/journey-of-the-photon.html) .

In this lab, we will use Python and Turtle to simulate *random walks* in order to figure out how many steps it takes a randomly moving body to move a certain straight-line distance from its starting point.

**Qu 1.** (Answer in responses document) Suppose we are told the length of each step in a random walk and the number of steps taken. Can we determine the straight-line distance of this random walk? Justify your answer in one or two sentences.

The random walk simulation program will be developed in numbered stages.

**Stage 1.** Create your program file, random\_walk.py. Do not use the graphics.py module for this lab. You must use the turtle module – so put import turtle at the top of your random\_walk.py file. Write a function do\_random\_walk(t). This function has a single parameter t, which is the turtle object that performs the random walk. The function should first reset the turtle, which returns it to the center of the screen and erases any path drawn previously. Then, it should perform a random walk of 50 steps with a step length of 10 pixels. Test your function to verify that it appears to be working correctly.

**Stage 2.** Generalize the do\_random\_walk function. Add parameters representing the number of steps and the step length. As always, test to verify that it is working before going on. This should be done at every stage of the development process, but will not be explicitly mentioned in the remainder of the lab.

**Stage 3.** Write a function print\_walk\_distance(t, num\_steps, step\_length). This function prints the straight-line distance of one random walk simulation, using the given turtle with the given number of steps and step length. You will need the distance method from the turtle module, which you should read about in the online documentation. When testing, try a few very long walks of 1000 steps or more. You can make these run more quickly by adding the following statement near the start of your top-level code: turtle.tracer(100, 0). Later on in the lab, continue to adjust the first argument of turtle.tracer to change the speed of turtle animation. Larger values make it run faster. See the supplementary study guide for a few more details on how to speed up turtle animations.

**Qu 2.** (Answer in responses document) Record the straight-line distances for five walks of 1000 steps, with step length 3. What is the average of these five distances and what is the largest deviation from the average? All calculations can be rounded to 2 decimal places.

**Stage 4.** Write a function print\_average\_distance(t, num\_steps, step\_length, num\_walks). This function prints the average straight-line distance of the given number of random walks.

**Qu 3.** (Answer in responses document) Record the average straight-line distances for five separate experiments. For each experiment, record the average straight-line distance of 50 random walks. As in the earlier experiments, each random walk has 1000 steps, with step length 3. What is the average of these five experiments, and what is the largest deviation from the average? All calculations can be rounded to 2 decimal places.

**Qu 4.** (Answer in responses document) In two or three sentences, compare the results of the previous two questions and suggest explanations for any differences or similarities.

**Qu 5.** (Answer in responses document) Spend some time experimenting with your code to answer the following question. If we double the number of steps in the random walk, what is the average effect on the straight-line distance of the random walk? It should be possible to form a reasonable hypothesis after a few minutes of experimentation with various calls to the print\_average\_distance function. Describe your results and your hypothesis in a paragraph of about 100 words.

**Qu 6.** (Answer in responses document) It can be shown mathematically that the straight-line distance of a random walk is proportional to the square root of the number of steps in the walk. Therefore, when we double the number of steps, the straight-line distance is expected to increase, on average, by a factor of $\sqrt{2}$, which is approximately 1.41. Is this in reasonable agreement with your answer to the previous question? If not, suggest at least one reason for the discrepancy. One or two sentences are sufficient to answer this question.

**Optional Stage 5.** Paste in the function graph\_average\_distances given at the end of this lab. You will also need the following import statement at the top of your file: import matplotlib.pyplot as plt. This requires the external library matplotlib, and the instructor can explain how to make this available. Execute the function and discuss the results. Does the graph have the expected shape? Record any thoughts in your responses document.

# Plot a graph of average straight-line distances of random walks for varying numbers

# of steps. Parameter t is the turtle object used for simulations.

def graph\_average\_distances(t):

 num\_data\_pts = 21 # number of data points to be graphed

 step\_len = 5 # step length for each random walk simulation

 num\_steps\_increase = 50 # Each time we go to a new data point,

 # how many extra steps do we include in the random walks?

 num\_trials = 25 # The number of random walks to be averaged for each data point

 x = [] # x-values of the data points

 y = [] # y-values of the data points

 for data\_pt in range(num\_data\_pts):

 num\_steps = data\_pt \* num\_steps\_increase

 # Compute average distance for num\_trials random walks

 trial\_dist = 0.0

 for \_ in range(num\_trials):

 do\_random\_walk(t, num\_steps, step\_len)

 dist = t.distance(0, 0)

 trial\_dist += dist

 avg\_dist = trial\_dist / num\_trials

 # Record the results for the current data point

 x.append(num\_steps)

 y.append(avg\_dist)

 plt.xlim([0, 1100]) # x-axis limits

 plt.ylim([0, 300]) # y-axis limits

 plt.title('random walk distances')

 plt.xlabel('number of steps')

 plt.ylabel('avg distance from starting point')

 plt.grid(True)

 plt.plot(x, y, 'r.') # plot with red dots

 plt.show()

**Acknowledgment.** This lab was originally authored by Grant Braught. It was adapted and edited by Lev Fruchter and John MacCormick.