Random Walk Project

MATH 242 • Spring 2026

Due: Monday, May 4

(Following the due date and initial grading, there will be an opportunity to revise and resubmit for a higher grade.)

Consider the following model of two-dimensional random walk that is not confined to integer-valued coordinates: start at the origin, and choose a random unit vector (in any direction) for each step. That is, to determine a step of the walk, first choose an angle θ (uniformly) from the interval $[0, 2\pi)$. The step of the walk will then be $(\cos(\theta), \sin(\theta))$.

Your Tasks

1. Implement the random walk. Make a few plots of sample random walks to show that your implementation works. Then answer the following two questions:

   (a) What is the average distance of the random walk from the origin after n steps? Make a plot showing how the average distance of the random walk from the origin depends on n, and find a function that roughly fits your plot.

   (b) Since the walk is not on a grid, it's very unlikely that it will return *exactly* to $(0, 0)$. However, it may return to within a circle of radius $\frac{1}{2}$ around the origin. What proportion of n-step random walks return at least once to some point within $\frac{1}{2}$ unit of the origin? How does this proportion depend on n? Investigate for various n and make a plot of your results. Do you think that *all* random walks will eventually return near the origin? Why or why not?

2. What happens if you constrain the random walk to the region $-5 \leq y \leq 5$? Modify your code to keep the y-coordinate of the walk between -5 and 5. Your random walk should not stop when it reaches one of the boundaries $y = 5$ or $y = -5$, nor should it jump back to the origin, but it should continue making unit steps within $-5 \leq y \leq 5$. Make a few plots of random walks to show that your modified implementation works as expected. Then answer questions (a) and (b) above for this constrained random walk.

3. For a grade of Excellent, investigate one other question about your random walk (either the unconstrained or constrained version) and state a conjecture based on your work.

Computational Notes

SageMath real-number types support high-precision arithmetic but are unfortunately too slow for this project. Instead, you should use Python floating-point numbers.

To generate random floating-point numbers, first import Python's random module like this:

```
import random as pyrandom
```

(By importing the module as `pyrandom` we don't overwrite the SageMath `random` function.) Then you can generate a random floating-point number between a and b as:

`pyrandom.uniform(`a`,`b`)`

To use the sine and cosine functions with floating-point numbers, first import Python's math module like this:

```
import math as pymath
```

Then you can call trig functions as `pymath.sin(`θ`)` and `pymath.cos(`θ`)`.

Grading Criteria

For projects in MATH 242, *communication* is as important as *computation*. You should turn in a well-organized notebook that clearly explains, using sentences and paragraphs, what you computed and what conclusions you can draw.

This project will be graded on the EMRN scale, as described in the syllabus. To receive a grade of *Meets Expectations*, your notebook should exhibit the following characteristics:

- You implement the random walk described above and answer the questions about its average distance from the origin and about its return to near the origin.
- You implement the constrained version of the random walk and answer the two questions for this random walk.
- Your code is appropriate for the given tasks and produces reasonable output.
- Your reasoning is explained using sentences, and your notebook is well-formatted and easy to read.
- No significant gaps or errors are present.

To receive a grade of *Excellent*, your notebook should further exhibit the following:

- In addition to answering questions (a) and (b) above, you investigate one of your own questions and state a conjecture supporte by your computations.
- Computational methodology demonstrates mastery of the computational techniques that we have studied in this course.
- Code is of high quality, demonstrating skillful use of programming constructs (e.g., variables, lists, functions, functions).
- Exposition is clear and precise, thoroughly explaining your methodology and reasoning. Discussion should things like assumptions made designing your methods, limitations of your computational techniques, and possible extensions for future study.
- The work must extend beyond the project requirements in a creative or insightful direction.