Vamshi Jandhyala

Can You Reach the Edge of the Square?

✠

19 October 2025

A couple of nice geometric-probability puzzles from *Fiddler on the Proof*.

Random walk in a unit square

You start at the centre of the unit square and pick a random direction to move in, with all directions equally likely. You move along this chosen direction until you reach a point on the perimeter. On average, how far can you expect to have travelled?

Analytical solution

Consider a unit square with corners at $(0,0), (1,0), (1,1)$, and $(0,1)$. By symmetry, restrict attention to the triangle with vertices $(\frac{1}{2}, \frac{1}{2})$, $(1, \frac{1}{2})$, and $(1,1)$. Pick θ uniformly in $[0, \pi/4]$. The distance to the right edge of the triangle is

$$d(\theta) \;=\; \frac{1}{2\cos\theta}.$$

The expected distance is

$$\begin{aligned}
\mathbb{E}[d] \;&=\; \int_0^{\pi/4} \frac{1}{2\cos\theta} \cdot \frac{4}{\pi}\, d\theta \\
&=\; \frac{2}{\pi} \int_0^{\pi/4} \sec\theta \, d\theta \;=\; \frac{2}{\pi} \ln\left|\sec\theta + \tan\theta\right|_0^{\pi/4} \\
&=\; \frac{2}{\pi} \ln(\sqrt{2}+1) \;=\; \frac{1}{\pi} \ln(3+2\sqrt{2}) \;\approx\; \mathbf{0.5614}.
\end{aligned}$$

Monte Carlo simulation

The simulation gives 0.5611, very close to the analytic value.

```python
import numpy as np

def simulate(n=1_000_000):
    theta = np.random.uniform(0, np.pi / 4, n)
    return 0.5 / np.cos(theta)

print(f"Simulated: {simulate().mean():.6f}")
```

Random walk in a unit cube

You start at the centre of a unit cube. You pick a random direction, again uniformly, and move until you reach the surface of the cube. On average, how far have you travelled?

Analytical solution

The cube has corners at $(0,0,0)$ and $(1,1,1)$. Start at $(\frac{1}{2},\frac{1}{2},\frac{1}{2})$ and pick a direction *uniformly*, equivalent to picking a random point uniformly on the unit sphere. Parameterise by spherical coordinates (θ,φ) with $\theta \in [0,2\pi)$ azimuthal and $\varphi \in [0,\pi]$ polar from the z-axis.

Generating uniformly distributed points on a sphere

Let ν be a point on the unit sphere S. We want the probability density $f(\nu)$ to be constant for a uniform distribution: $f(\nu) = 1/(4\pi)$, since $\int_S f(\nu)\,dA = 1$ and $\int_S dA = 4\pi$. Representing points by (θ,φ) and noting $dA = \sin\varphi\,d\varphi\,d\theta$ gives

$$f(\theta,\varphi) \;=\; \frac{1}{4\pi}\sin\varphi.$$

Calculating the expected distance

By symmetry restrict to one fourth of one face. The distance from the centre to a point on the surface is

$$d(\theta,\varphi) \;=\; \frac{1}{2\cos\theta\sin\varphi}.$$

Therefore

$$\begin{aligned}\mathbb{E}[d] \;&=\; 24\cdot\frac{1}{4\pi}\int_0^{\pi/4}\int_{\arctan(\sec\theta)}^{\pi/2}\frac{\sin\varphi}{2\cos\theta\sin\varphi}\,d\varphi\,d\theta\\ &=\; \frac{3}{\pi}\int_0^{\pi/4}\Big(\frac{\pi}{2}-\arctan(\sec\theta)\Big)\sec\theta\,d\theta \;\approx\; \mathbf{0.610687}.\end{aligned}$$

Python code for numerical integration

```python
import numpy as np
from scipy import integrate

result1, err1 = integrate.dblquad(
    lambda y, x: 3 / (np.pi * np.cos(x)),
    0, np.pi / 4,
    lambda x: np.arctan(1 / np.cos(x)),
    lambda x: np.pi / 2,
)
print(f"Integral: {result1:.10f}")
```

Monte Carlo simulation

Sampling. Marginalise the joint distribution to get

$$f(\theta) \;=\; \frac{1}{2\pi}, \qquad f(\varphi) \;=\; \frac{\sin\varphi}{2}.$$

Sampling θ is straightforward. For φ use inverse transform sampling: with

$$F(\varphi) \;=\; \int_0^{\varphi} f(\hat{\varphi})\, d\hat{\varphi} \;=\; \tfrac{1}{2}(1-\cos\varphi),$$

draw $u \sim \mathcal{U}[0,1]$ and set $\varphi = F^{-1}(u) = \arccos(1-2u)$.

The Monte Carlo estimate is 0.6106855, very close to the analytical value.

Simulation in Python

```python
import numpy as np

def simulate_random_walk_3d(num_simulations=1_000_000):
    x0, y0, z0 = 0.5, 0.5, 0.5
    theta = np.random.uniform(0, 2 * np.pi, num_simulations)
    phi  = np.arccos(1 - 2 * np.random.random(num_simulations))

    dx = np.sin(phi) * np.cos(theta)
    dy = np.sin(phi) * np.sin(theta)
    dz = np.cos(phi)

    t_right = np.where(dx > 0, (1 - x0) / dx, np.inf)
    t_left  = np.where(dx < 0, -x0 / dx, np.inf)
    t_front = np.where(dy > 0, (1 - y0) / dy, np.inf)
    t_back  = np.where(dy < 0, -y0 / dy, np.inf)
    t_top   = np.where(dz > 0, (1 - z0) / dz, np.inf)
    t_bottom = np.where(dz < 0, -z0 / dz, np.inf)

    distances = np.minimum(
        np.minimum(np.minimum(t_right, t_left),
                   np.minimum(t_front, t_back)),
        np.minimum(t_top, t_bottom),
    )
    return distances

distances = simulate_random_walk_3d(100_000_000)
print(f"Simulated: {np.mean(distances):.6f}")
```

References

corysimon.github.io — uniform distribution on a sphere.