James' Random Walk Problem: Description, Solution, Simulation, and Visualization

M.M. Dalkilic, PhD
Bloomington, IN, USA February 13, 2025

Introduction and Motivation

As described in the club by James, random walk questions are common for quant interviews. Random walks are an important class of models that provide insight into how random processes behave over time. For this problem, we describe a simple random walk of two independent processes seeking to answering a question about their relationship. There exists a fair amount of technical material on the subject; for this exposition, we'll be limiting ourselves to examination with the fewest mathematical skills required, but additionally building a simulation to find whether our approach appears sound. To our delight the simulation is consistent with the theory.

Problem

> Two drunks both start at a position *zero* just outside the bar wanting to return to their respective homes. We envision this as something like this: $\cdots \leftrightarrow \boxed{\text{-1}} \leftrightarrow \boxed{0} \leftrightarrow \boxed{1} \leftrightarrow \cdots$. Their inebriation causes them each to randomly take single steps to the left or right with the same equal probability, *i.e.*, left 50% of the time and right 50% of the time independently both of their respective previous steps and one another. After each has completed six total steps, what is the probability that they have returned to the starting point zero?

When there are only two outcomes, but several times, we can leverage the binomial:

$$\binom{n}{m} = \frac{n!}{m!(n-m)!} \tag{1}$$

$$(a+b)^n = \sum_{i=0}^{n} \binom{n}{n-i} a^{n-i} b^i \tag{2}$$

The coefficient gives us the count of permutations of a, b. As a probability model, we'd include the two probabilties of a, b occurring; however, since they're equal, we can focus solely on the coefficients. For this problem, $n = 6$. If we let $a = \mathsf{L}, b = \mathsf{R}$ for moving left and right, respectively we find for six steps:

LLLLLL	1	(3)
LLLLLR	6	(4)
LLLLRR	15	(5)
LLLRRR	20	(6)
LLRRRR	15	(7)
LRRRRR	6	(8)
RRRRRR	1	(9)

For example, there are six permutations where there is only one step to the right:

LLLLLR, LLLLRL, LLLRLL, LLRLLL, LRLLLL, RLLLLL

The total number of different permutations is $2(1) + 2(6) + 2(15) + 20 = 64$. Continuing our examination when when the drunk takes only a single step right, he'll finish at -4 (using a number line starting at zero). Fig. 1 shows all the possible final locations after six steps using a number line starting at zero.

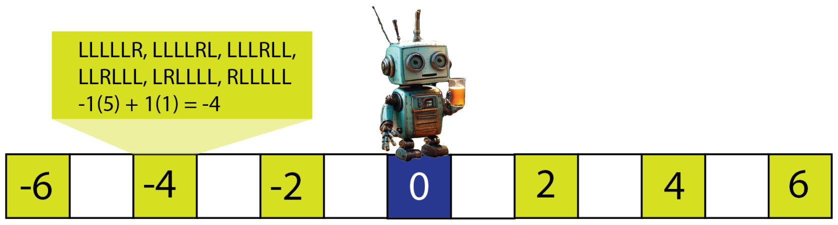

Figure 1: A drunk (robot) starting a zero. Each color square represents a possible final destination after six steps. We let left $\mathsf{L} = -1$ and $\mathsf{R} = 1$. There are six different ways to end at -4 which is $\binom{6}{1} = \frac{6!}{1!(6-1)!}$.

The probabilites can be found by dividing the coeficient values by the sum.

$$\mathsf{LLLLLL} \qquad \frac{1}{64} \tag{10}$$
$$\mathsf{LLLLLR} \qquad 6/64 \tag{11}$$
$$\mathsf{LLLLRR} \qquad 15/64 \tag{12}$$
$$\mathsf{LLLRRR} \qquad 20/64 \tag{13}$$
$$\mathsf{LLRRRR} \qquad 15/64 \tag{14}$$
$$\mathsf{LRRRRR} \qquad 6/64 \tag{15}$$
$$\mathsf{RRRRRR} \qquad 1/64 \tag{16}$$

Let's see code that generates these values:

```python
import math

n = 6 #stepps
coefs = [math.comb(n,i) for i in range(n+1)]

for i in range(n+1):
    movement = 'L'*(n-i) + 'R'*i
    location = -1*(n-i) + 1*i
    print(movement, location, coefs[i],coefs[i]/sum(coefs))
```

produces:

```
LLLLLL -6 1 0.015625
LLLLLR -4 6 0.09375
LLLLRR -2 15 0.234375
LLLRRR 0 20 0.3125
LLRRRR 2 15 0.234375
LRRRRR 4 6 0.09375
RRRRRR 6 1 0.015625
```

A drunk, by himself, will have about a 31% chance to return to where he started after six steps, but about a 10% chance to land at -4 or 4. Because there are an even number of steps, it's impossible to land on an odd number. We'd be claiming that $2k = 2k + 1$ for some k which means we'd be saying that $0 = 1$.

Simulation and Visualization of One Dimension

In this section we describe how we can simulate the distribution. First, observe that after six steps, there are only seven values: -6,-4,-2,0,2,4,6. Our simulation will generate these values from a random, uniform process using -1,1 in place of L, R. We sum these six values resulting in one of the final destinations. Here's the pertinent code:

```
#input n steps left: -1, right 1 equiprobable
#output sum
def get_drunken_walk(n):
    return sum([1 if rn.randint(0,1) else -1 for _ ↩
       in range(n)])
```

If `n = 6`, then we'll have a summation of six terms using -1,1. We generate 1,000,000 walks keeping track of how each combination. Here's the relevant code:

```
#simulations
history = {}
simulations = 1000000
simulated_history = {}
for i in range(simulations):
    walk = get_drunken_walk(n)
    if walk in history.keys():
        history[walk] += 1
    else:
        history[walk] = 1
```

The `simulated_history` will hold the fractions computed from `history`. Let's plot the two distributions using heat maps. This is a popular visualization tool to better understand relative counts that perhaps are too difficult to understand from simply numbers. Figure 2 shows two heatmaps. On the left is our theoretical map (with probabilities to the left). On the right is the simulation. The values are nearly identical.

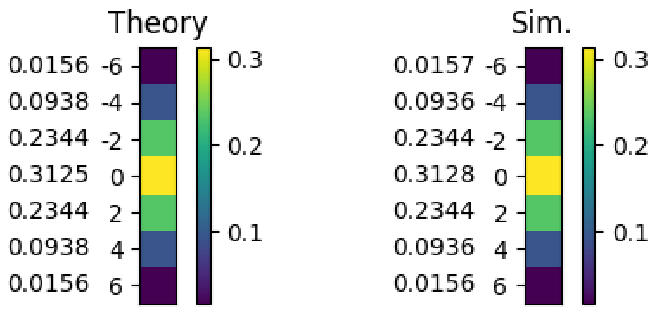

Figure 2: Heatmaps of the final destinations taking six steps randomly. The simulation is consistent with the theory.

Two Drunks

We now move to two drunks. We leverage the fact that they move independently of each other (in addition to moving independently at each step). Figure 3 shows how we'll visualize the two. We mark one drunk moving horizontally and the other vertically. This makes understanding simpler as well as providing an easier graphic to interpret. Another helpful technique is to

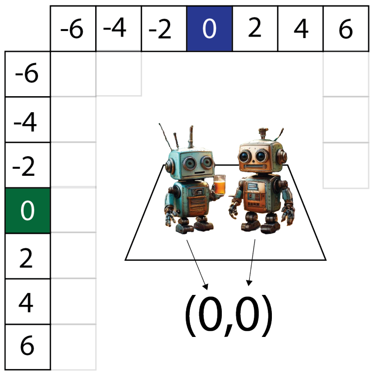

Figure 3: Both drunks (robots) starting a zero. Each color square represents a possible final destination after six steps. Since these are independent, we form the cross product to find the probablities.

introduce some notation (it's really just programming). How might we write the problem using probabilities? Let's use X, Y represent drunk 1 and drunk 2, respectively. Let's use our subscript to denote steps and superscript where the drunk ends up. Then, to start, we have X_0^0, Y_0^0. What we want to find is $P(X_6^0 \text{ and } Y_6^0)$. Since these processes are independent, we can rewrite this as:

$$\begin{aligned} P(X_6^0 \text{ and } Y_6^0) &= P(X_6^0)P(Y_6^0) && (17) \\ &= \frac{20}{64} \cdot \frac{20}{64} \approx 0.0977 && (18) \end{aligned}$$

To model two, independent drunks we can use the product for (i, j), $i, j \in \{-6, -4, -2, 0, 2, 4, 6\}$ as X_6^i, Y_6^j and, thus,

$$P(X_6^i \text{ and } Y_6^j) = P(X_6^i)P(Y_6^j) \quad (19)$$

Figure 3. shows two heat maps: the theoretical distribution and simulation.

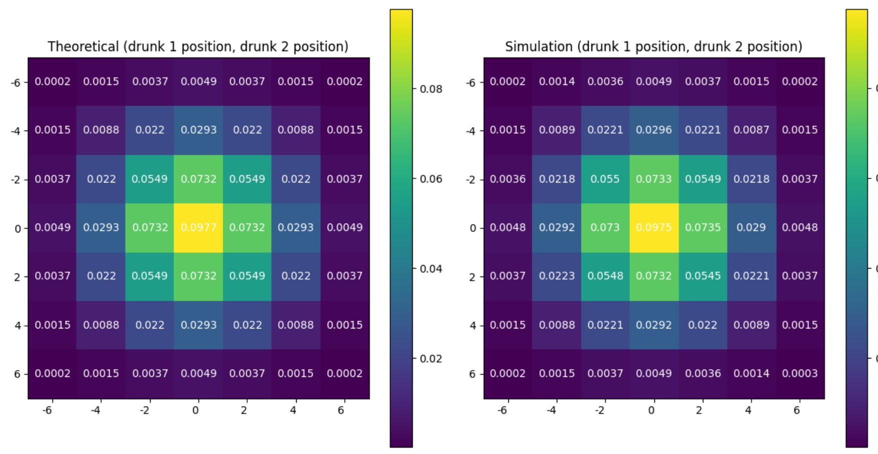

Figure 4: Each cell is indexed by the pair of final locations $i, j \in \{-6, -4, -2, 0, 2, 4, 6\}$, X_6^i, Y_6^j the entry is the probability of that event. For example, (0,0) means $X_6^0 \text{ and } Y_6^0$ in which both drunks, after six steps, found themselves at the place they began. This occurs about 10% of the time. (Left) Theoretical distribution of the possible final destinations of the two drunks. (Right) Simulation of the outcomes.

Each cell is indexed by the pair of final locations on the number line and contains the probability of that event. For example, (0,0) means both drunks, after six steps, found themselves at the place they began. If we sum across, say $j = 0$, we have all the probabilties where X stopped:

$$\begin{aligned}\sum_i P(X_6^i \text{ and } Y_6^0) &= 2(0.0049) + 2(0.0293) + 2(0.0732) + 0.0977 \quad (20)\\ &= 0.3125\end{aligned}$$

which is the marginal probability of Y_6^0 (see Fig. 2).

Questions for Students

1. Look for the arguments for and against random walks in finance. What is evidence that the model is useful? What are the criticisms?

2. The problem is usually generalized: what's the probability that, after six steps, both drunks end in the same spot when starting from the same spot (origin)?

   (a) How might we use our notation to describe this problem?

   (b) What would be a reasonable simple formula?

   (c) How might we use the tables to determine the value? On this account, there are closed forms (which require a bit of experience), but one of the simplest for this problem is $n = 6$:

$$\sum_{k=0}^{n} \binom{n}{k}^2 = \binom{2n}{n} \tag{21}$$

$$2^{-2(n)} \sum_{k=0}^{n} \binom{n}{k}^2 = \frac{\binom{2n}{n}}{2^{2n}} \tag{22}$$

$$\frac{\binom{2(6)}{6}}{2^{2(6)}} = \frac{924}{4096} \approx 0.223 \tag{23}$$

   We need Eq. 21 which allows us to simplify the summation. There will be a slight difference in the values drawn from the tables and the one above. Code confirms this:

```
import math
n = 6

def summation(n):
    return (2**(-2*n)) * sum([(math.comb(n,↩
       k))**2 for k in range(n+1)])

close_form = lambda n: math.comb(2*n,n)↩
    *(2**(-2*n))

print(summation(n), close_form(n))
```

producing:

```
0.2255859375 0.2255859375
```

3. What if drunk 1 stepped right with .6 of the time and left the remainder, while drunk 2 did the opposite?

   (a) What additions would we need in the theoretical and simulation?

   (b) What is the probablity that both end up where they started?

4. Another form of this problem is someone standing where he's k steps from a cliff ($k+1$ he falls to his death on the right) with any amount of space to the left. What's the probablity (if both steps are equally likely) for the fellow to live after taking $n \geq k$ steps? To simplify this problem, let's assume the cliff is five steps away on the right. If he takes six steps he'll fall. What is the probability he'll live after eight steps?

Simulation and Visualization Code

To simulate this problem we allow both drunks to take six steps 1,000,000 times, then find the relative frequence (which is simply dividing by 10^6). We then use heat maps to visualize the probabilities. Some code is written less succintly to show students what's happening. For example, line 60 would, for a real world solution, be written as a function depending on n. The first program generates the single simulation; the second is the full simulation.

Simulation and Visualization for One Drunk

```
 1  import random as rn
 2  import matplotlib.pyplot as plt
 3  import numpy as np
 4  import math
 5
 6  # MMDalkilic, Bloomington, IN, USA
 7  # 2/12/2025
 8
 9  # compare theoretical to simulation
10  # of a drunk taking  six steps, left or right with equal probability
11  # what are the probabilities of the final destinations>
12
13  #input n steps left: -1, right 1 equiprobable
14  #output sum
15  def get_drunken_walk(n):
16      return sum([1 if rn.randint(0,1) else -1 for _ in range(n)])
17
18
19  #hold theoretical distribution of one drunk
20  theo_dist_1_dimension = {}
21  n = 6 #number of steps
22  coef = [math.comb(n,i) for i in range(n+1)] # binomial coefficients
23
24  #build distribution for drunk 1
```

```
for i in range(n+1):
#    movement = 'L'*(n-i) + 'R'*i
     location = -1*(n-i) + 1*i
     theo_dist_1_dimension[location] = coef[i]/sum(coef)


#simulations
history = {}
simulations = 1000000
simulated_history = {}
for i in range(simulations):
    walk = get_drunken_walk(n)
    if walk in history.keys():
        history[walk] += 1
    else:
        history[walk] = 1

for k,v in history.items():
    simulated_history[k] = round(v/simulations,6)

#need labels as locations
drunk = [-6,-4, -2, 0, 2, 4, 6]

#convert label to array position
m = {-6:0, -4:1, -2:2, 0:3, 2:4, 4:5, 6:6}

#build heat maps
positions_theory = np.zeros((7,1))
positions_simulation = np.zeros((7,1))

for k,v in theo_dist_1_dimension.items():
    positions_theory[m[k]] = v

for k,v in simulated_history.items():
    positions_simulation[m[k]] = v

#create plot
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(5, 2))
```

```python

# Plot the first heatmap
heatmap1 = ax1.imshow(positions_theory)
ax1.set_title('Theory')
fig.colorbar(heatmap1, ax=ax1)

# Plot the second heatmap
heatmap2 = ax2.imshow(positions_simulation)
ax2.set_title('Sim.')
fig.colorbar(heatmap2, ax=ax2)

#Show ticks as final locations
ax1.set_xticks([])
ax1.set_yticks(range(len(drunk)), labels=drunk)

ax2.set_xticks([])
ax2.set_yticks(range(len(drunk)), labels=drunk)

#display probabilities next to ticks
for i in range(len(drunk)):
        text1 = ax1.text(-3, i,round(positions_theory[i][0],4),ha="center", va="center", color="black")

for i in range(len(drunk)):
        text2 = ax2.text(-3,i,round(positions_simulation[i][0],4),ha="center", va="center", color="black")

fig.tight_layout()
plt.show()
```

Simulation and Visualization for Two Drunks

```python
import random as rn
import matplotlib.pyplot as plt
import numpy as np
import math

#MMDalkilic, Bloomington, IN, USA
#2/12/2025


#compare theoretical to simulation
#two drunks start at origin
#take six steps, left or right with equal probability and independently
#what is the probability they end at the origin?

#input n steps left: -1, right 1 equiprobable
#output sum
def get_drunken_walk(n):
    return sum([1 if rn.randint(0,1) else -1 for _ in range(n)])


#hold theoretical distribution of one drunk
theo_dist_1_dimension = {}
n = 6 #number of steps
coef = [math.comb(n,i) for i in range(n+1)] #binomial coefficients

#build distribution for drunk 1
for i in range(n+1):
#   movement = 'L'*(n-i) + 'R'*i
    location = -1*(n-i) + 1*i
    theo_dist_1_dimension[location] = coef[i]/sum(coef)

#build distribution for both drunk 1, drunk 2
```

```
#these are independent P(drunk 1, drunk 2) = P(↩
   drunk 1)P(drunk 2)
theo_dist = {}
for k1,v1 in theo_dist_1_dimension.items():
    for k2, v2 in theo_dist_1_dimension.items():
        theo_dist[(k1,k2)] = round(v1*v2, 6)

#simulations
history = {}
simulations = 1000000
simulated_history = {}
for i in range(simulations):
    walk_1 = get_drunken_walk(n)
    walk_2 = get_drunken_walk(n)
    if (walk_1, walk_2) in history.keys():
        history[(walk_1, walk_2)] += 1
    else:
        history[(walk_1, walk_2)] = 1


for k,v in history.items():
    simulated_history[k] = round(v/simulations,6)


#need labels as locations
drunk_1 = [-6,-4, -2, 0, 2, 4, 6, ]
drunk_2 = drunk_1
#convert label to array position
m = {-6:0, -4:1, -2:2, 0:3, 2:4, 4:5, 6:6}

#build heat maps
positions1 = np.zeros((7, 7))
positions2 = np.zeros((7, 7))

for k,v in theo_dist.items():
    x,y = m[k[0]],m[k[1]]
    positions1[x][y] = v

for k,v in simulated_history.items():
    x,y = m[k[0]],m[k[1]]
```

```python
    positions2[x][y] = v

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 6))

# Plot the first heat map
heatmap1 = ax1.imshow(positions1)
ax1.set_title('Theoretical (drunk 1 position, drunk 2 position)')
fig.colorbar(heatmap1, ax=ax1)

# Plot the second heat map
heatmap2 = ax2.imshow(positions2)
ax2.set_title('Simulation (drunk 1 position, drunk 2 position)')
fig.colorbar(heatmap2, ax=ax2)

# Show all ticks and label them with the respective final location
ax1.set_xticks(range(len(drunk_1)), labels=drunk_1)
ax1.set_yticks(range(len(drunk_2)), labels=drunk_2)

ax2.set_xticks(range(len(drunk_1)), labels=drunk_1)
ax2.set_yticks(range(len(drunk_2)), labels=drunk_2)

# Loop over data dimensions and create text annotations.
for i in range(len(drunk_1)):
    for j in range(len(drunk_2)):
        text1 = ax1.text(j, i,round(positions1[i][j],4),ha="center", va="center", color="w")

for i in range(len(drunk_1)):
    for j in range(len(drunk_2)):
        text2 = ax2.text(j, i,round(positions2[i][j],4),ha="center", va="center", color="w")

fig.tight_layout()
```

```
plt.show()
```