# The Mathematical Analysis of Random Walk Systems

Ali Ashrafy, Benoy Drenzla, Tom Prickett
St Columba Anglican School

August 13, 2023

**Abstract**

The University of Melbourne Mathematics and Statistics Research Competition (MSRC) involves working through a preset series of problems, numbered 1 through 15. This report presents a solution and further elaboration on the 15th problem.

## 1 Problem 15

Problem 15 requires us to investigate the movement of a spider on a two-dimensional integer lattice, initiating its journey from the origin, $O$. The spider has an equal probability of 0.25 to move in each of the four cardinal directions: up, down, left, or right. We are particularly interested in determining the spider's escape time, $T$, which represents the number of steps the spider takes to reach a predefined boundary curve, $\Gamma$. This escape time, $T$, is a random variable with positive integer values, and the probabilities give its distribution

$$\{\mathbb{P}(T = n) : n = 1, 2, 3, ...\}.$$

Our primary task is to calculate the expected escape time, $\mathbb{E}[T]$:

$$E(T) = \sum_{n=1}^{\infty} n \times P(T = n) \tag{1}$$

which represents the average number of steps the spider would take to reach the boundary. This problem is conceptualised within the framework of random walks in probability theory, where an agent traverses through a grid based on random processes.

### 1.1 Problem 15 A

Problem 15 A, the primary task, is to compute the expected escape time of the spider for the figure below:

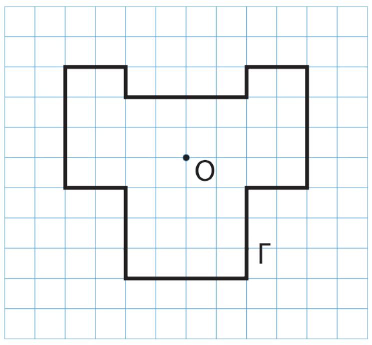

Figure 1: Boundary Curve Γ.

### 1.2 Problem 15 B

Given a fixed length for a boundary curve, how should you design a trap such that the spider's expected escape time from the starting point is:

1. Maximized (it takes the spider the longest possible time to escape)?
2. Minimized (the spider escapes in the shortest possible time)?

Essentially, problem 15 B is to give insights into shapes for Γ that either prolong or shorten the spider's journey to escape.

### 1.3 Problem 15 C

Is it possible to construct two regions that are geometrically different (namely, they are not rotations/ reflections of each other) but induce the same expected escape time?

### 1.4 Problem 15 D

Does the distribution of the escape time uniquely define the shape of the trap?

- If yes, can the boundary curve Γ be determined solely from the distribution of T?

## 2 Describing Random Walks

### 2.1 Introduction to Random Walks

A random walk is one of the fundamental concepts in probability theory and statistical physics. In summary, a random walk is a sequence of steps determined by a random mechanism. In the context of our question, the spider's journey across the two-dimensional integer lattice can be conceptualized as a random walk. At every point in its path, the spider's next move — up, down, left, or right — has an equal probability of 0.25.

The two-dimensional lattice random walk, as represented by our spider's movements, is a classic example of a discrete-time stochastic (or random) process. Each step in the spider's journey corresponds to a random variable, resulting in a sequence of random variables that describe its path.

Understanding the properties and behaviours of random walks is one way of analysing the expected escape time, $T$, for the spider to reach the boundary curve, $\Gamma$. In this report, we investigate the relationship between the spider's escape time and the parameters of the boundary; including size and shape.

By delving deep into the nature of random walks, we aim to gain insights into the spider's average time to escape, $\mathbb{E}[T]$, and the probabilistic dynamics governing its journey.

## 2.2 Markov Chains

To understand the spider's movements and calculate the time to escape, $\mathbb{E}[T]$, Markov chains are used. Markov Chains are mathematical concepts that not only have many applications in statistics, biology, economics, and physics but also enable an understanding of stochastic processes. They have been shown to describe the probabilities of tending towards certain states given the initial position.

Let's see an example of the use-case of a Markov Chain. Take a very simple example of the weather; it can exist in 2 states: sunny, or rainy, and can transition between these with determined probabilities. This behaviour can be illustrated using a graph where each node represents the state the weather can be in, that is, sunny or rainy, and the arrows represent the probability of moving from one state to another.

Note that a node can also link to itself, represented by a circular arrow leading to itself.

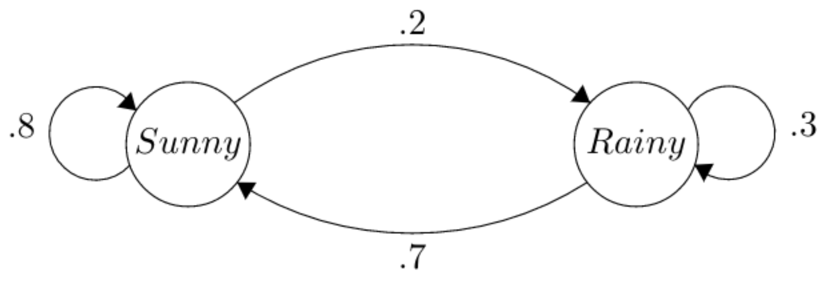

Figure 2: An example of a Markov chain diagram.

The transition probabilities must adhere to two important properties:

- The Markov Property: The future state depends only on the current state and not on the sequence of states that preceded it.

- The sum of probabilities of outgoing transitions from any state is always 1, ensuring a valid probability distribution.

### 2.2.1 Transition Matrices

An efficient way to represent Markov chains is through a transition matrix ($A$) where each element denotes the transition probability from one state (the rows) to another (the columns). Given a starting state, multiplying by the transition matrix gives the probability of moving from one state to the next. Doing this iteratively gives the probability of moving from the initial position (the rows) to the final state (the columns).

Formally, for a higher order transition matrix, when calculating for an $n$ amount of steps, you multiply the transition matrix by itself an $n$ amount of times.

Thus, we can find the probability of being at a certain state by iterating the transition matrix $A$ by the number of steps $n$ we find the probability of going from a state ($i$) to a second state ($j$), which can be expressed as the term $A_{i,j}$ which is a matrix with $i$ rows and $j$ columns. Take the previous example of the weather. Given that there are two states, it's possible to say that the transition matrix will have two potential initial states, and two potential final states, the resulting matrix will therefore be $2 \times 2$. By assuming that the state 1 is sunny, and state 2 is raining, we can represent these states as the numerical equivalent in rows and columns. That is, the first row is the *sunny* state and the second row is the *raining* state. Similarly, the first column is the *sunny* state and the second column is the *raining* state. Hence we can create a transition matrix based on Figure 2:

$$P(d) = \begin{bmatrix} 0.8 & 0.2 \\ 0.7 & 0.3 \end{bmatrix}$$

And by multiplying the matrix by itself $d$ amount of times (where $d$ is the number of days) we can calculate the probabilities of it being in state 1, or state 2 after $d$ days:

$$P(d) = \begin{bmatrix} 0.8 & 0.2 \\ 0.7 & 0.3 \end{bmatrix}^d$$

We can thus represent the probability of going from an initial state $i$ to the final state $j$ over a period of n steps by the expression:

$$P(n) = A^n$$

Which can be expanded as:

$$P(n) = \begin{bmatrix} P_{1,1} & P_{1,2} & \cdots & P_{1,j} \\ P_{2,1} & P_{2,2} & \cdots & P_{2,j} \\ \vdots & \vdots & \ddots & \vdots \\ P_{i,1} & P_{i,2} & \cdots & P_{i,j} \end{bmatrix}^n$$

## 2.3 Application of Markov Chains in Random Walks

The random walk, as described earlier, possesses the Markov property: the next state in the sequence (or the next step in the walk) is dependent solely on the current state and not on the sequence of events leading up to it. Hence, it is a prime candidate for representation as a Markov chain.

A random walk's transition matrix effectively captures the probabilities of moving from one state to another. For our two-dimensional spider scenario, the transition matrix would indicate the probabilities of moving up, down, left, or right from any given position. As time progresses and the spider makes more steps, the probabilities evolve and change according to the transition matrix.

One aspect of understanding random walks via Markov chains is the concept of 'null states'. In our spider scenario, the boundary curve, $\Gamma$, can be thought of as a *null state*. Once the spider reaches any of these points, it won't return to the inner lattice. It's at the end of its journey. Mathematically, a null state in a Markov chain is a state that, once entered, cannot be left, and does not affect any probabilities of any other state as the number of iterations increases. Thus each of the coordinates of the boundary curve, $\Gamma$, will have a total probability of moving from the state, or to itself, as zero.

Analysing the spider's random walk, considering the null boundary, will give insights into the expected time to a null state or, in the spider's case, the escape time. Understanding this expected time is vital for solving our primary problem.

Lastly, by leveraging Markov chains, we can answer several intriguing questions about the spider's journey, such as:

- What are the probabilities of the spider being at different positions after a certain number of steps?
- Which paths are most likely for the spider to take on its way to the boundary?
- How do these probabilities and paths change based on the spider's starting position within the lattice?
- By investigating the probability distribution of the spider solely, can we make any inferences regarding the starting position of the spider and the size and shape of the boundary?

### 2.3.1 Markov Chains to Solve Problem 15 A

By using the Markov chains and transition matrix properties to answer question 15 we can calculate the expected number of steps to escape through the use of the formula given. The boundary points are known and can therefore be labelled from $1 \leftarrow 111$, accounting for all of the points in the enclosed space of the boundary curve. Thus we need a transition matrix which is $111 \times 111$.

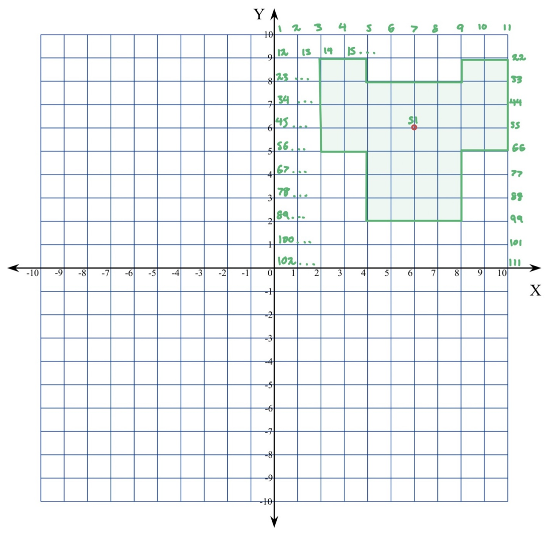

Figure 3: Boundary curve with labelled points.

$$P(n) = \begin{array}{c|cccc} & 1 & 2 & \cdots & 111 \\ \hline 1 & P_{1,1} & P_{1,2} & \cdots & P_{1,j} \\ 2 & P_{2,1} & P_{2,2} & \cdots & P_{2,j} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 111 & P_{i,1} & P_{i,2} & \cdots & P_{i,j} \end{array}^{\,n}$$

Figure 4: Transition matrix with rows and columns representing the total 111 initial states at which the spider can be $i$ (the rows) and the 111 final states at which the spider can be $j$ (columns).

$$\mathbb{E}(T) = \sum_{n=1}^{\infty} n \times P(T = n)$$

Which can further be expanded as:

$$\mathbb{E}(T) = \sum_{n=1}^{\infty} n \times \Big( \sum_{m_{initial}}^{m_{final}} \times A^{n}_{i,j} \Big) \tag{2}$$

Where $i$ is the original coordinate (represented by the $i$-th row on the transition matrix) and $m_{initial} \rightarrow m_{final}$, are the points represented by the boundary curve. That is $A^{n}_{i,m}$ is the point in the transition matrix which represents the probability of starting at point $i$ and finishing at a point $m$ which is on the boundary curve. As we are starting the spider at coordinates $(6, 6)$ or point 51 we can visually represent by Figure 5.

Ideally, a stationary distribution or equilibrium state is reached when the product remains unchanged over iterations. This is because the probability of the same states occurring as the number of iterations moves towards infinity typically tends towards zero.

$$P(n) = \left[\begin{array}{r|cc|c|c} & 1 & 2\ \dots & m & \dots\ 111 \\ 1 & P_{1,1} & P_{1,2} & & P_{1,j} \\ 2 & P_{2,1} & P_{2,2} & & P_{2,j} \\ \vdots & & & & \\ \hline 51 & & & P_{51,m} & \\ \hline \vdots & & & & \\ 111 & P_{i,1} & P_{i,2} & \cdots & P_{i,j} \end{array}\right]^{n}$$

Figure 5: Visual representation of the probability of being at the boundary point $m$ from the starting point 51.

Thus the expected number of steps to reach the boundary is represented by the sum of the probabilities of going from the original state $(O)$ to the boundary coordinates multiplied by the number of iterations, or steps that the spider has taken, $n$, as $n \to \infty$.

## 2.4 The Monte Carlo Method

The Monte Carlo method is a statistical technique that uses random sampling to solve problems that may be deterministic in nature. Although it is typically utilised in the evaluation of integrals, especially in high-dimensional spaces, optimisation and probability distribution, it can also be used for simulations.

Consider the following example of a typical use-case of a Monte Carlo methodology to approach a problem:

Suppose that you want to evaluate the integral of a function $f(x)$ over the interval $[a, b]$. The steps to do so are as follows:

1. Randomly sample points $x_i$ in the given interval $[a, b]$.
2. Evaluate $f(x_i)$ for each sampled point.
3. Taking the average of these evaluations will converge to $f(x)$ over $[a, b]$. Multiplying this by the length of the interval will result in an estimate of the integral.

## 2.5 Application of the Monte Carlo Method in Random Walks

Both the Markov Chains and Monte Carlo methods will be used in parallel to ensure the reliability of our results.

The Monte Carlo approach describes a method for simulating a random walk over very large $n$. It provides us with an estimate of the expected escape time for the given boundary to compare with the more analytical approach with Markov Chains (assuming our results for the simulation are correct).

Essentially, in each iteration up to an arbitrarily large $n$, a random walk is simulated until the spider reaches the boundary. The sum is incremented by the steps taken which is then divided by the $n$, the number of trials after all iterations are complete.

## 3 Solving Problem 15 A

Although it would be possible to calculate the expected value by hand multiplying the matrix, the computational complexities involved are large.

Approaching this programmatically has made it possible to both simplify the time needed to calculate the probability and ensure greater accuracy in the results.

The code is written in Python 3.10 and can be found in the appendices. There are 2 files, each of which use the different approaches; markov.py and montecarlo.py. Please note that markov.py will require an installation of the Python package Numpy for matrix multiplication. Further information can be found here.

#### 3.0.1 Logic and Methodology

1. **Transition Matrix Definition:** First we need to create a transition matrix (Represented in Figure 4), $A$, which is set up for the spider's movements. For an interior point $i$, $A_{i,j}$ is 0.25 if $j$ is one of the *N, E, S, W* (not diagonal) neighbors of $i$. Else the probability will be set to zero. Further, the boundary curves will be set to null states or escape states, and thus all probabilities within this will be zero.

2. **Matrix Power Computation:** The $n$-th power of the matrix, $A^n$, provides the probabilities of reaching each lattice point from any initial point after $n$ steps (shown in Figure 5).

3. **Iterative Calculation:** An iterative approach is employed. Starting from the initial state, we sum the probabilities of being at each subsequent boundary point represented in the matrix (Shown in Equation 2 and Figure 5). Iterations will then stop when the probability of not escaping becomes negligible in 32-bit single-precision floating points (explained in more detail below). This will be our best estimate for the expected value.

4. **Expected Escape Time Calculation:** Using the recursive relationship, the expected time is calculated using the sum of going from the initial state $O$ denoted by the $i$-th row to the boundary curve points, denoted by the $j$-th column. By then substituting the number of iterations which have occurred (the number of steps the spider has taken) and the sum of the probabilities of being at the boundary and escaping, we can thus find $\mathbb{E}(T)$:

$$E(T) = \sum_{n=1}^{\infty} n \times P(T = n)$$

5. **Computational Approach:** Given the complexities of matrix operations, we created a program to manage the matrix powers, boundary adjustments, and the expected escape time.

## 3.1 Results for Problem 15 A

Thus by following the methodology as shown above, we can calculate the expected escape time for the given shape $\Gamma$ which converges towards:

$$\mathbb{E}(T) = 7.520354455020274 \text{ steps}$$

### 3.1.1 The Result

At the beginning of solving the problem, it was thought that the expected number of steps will be close to the shortest path to the furthest point on the boundary curve. In the case of Problem 15 A, this is 6. As the final result was approximately 7.52, the result is 20 percent off. There is no clearly observable relationship, however, it would be interesting to compare the results of expected time compared to the shortest path to the furthest point over multiple shapes.

### 3.1.2 Explanation of Result

This value is reached after exactly $n = 195$ iterations. It does not appear to change after this as the new values are too small to be tracked by the 32-bit single precision floating point numbers that Python and most other languages use as their default storage method for real numbers. If it interests you, you may read more about it here.

We can verify this number by running a Monte Carlo simulation. With the given boundaries, the Monte Carlo method produces:

$$E(T) \approx 7.52 \text{ steps}$$

Which is nearly exactly the same as the value obtained with the Markov method. Thus, the reliability of our results can now be supported, and the method used is valid.

It is noteworthy that the Markov Chain method will return a more precise value, and should be favoured over the Monte Carlo, which has served its purpose as a demonstrator for the Markov method's validity.

Additionally, the Monte Carlo method is only an approximation as the values change each time it runs as it is a truly random walk simulation. However, it does still converge to around 7.52 as seen before.

# 4 Solving Problem 15 B

To address this problem we assume that Γ **must be an enclosed shape** - that is, one such that the spider cannot exit the confinement of the curve without first hitting the boundary of the curve itself.

We must also assume that **the curve must be placed on a discrete grid**, where each edge must be placed parallel to either of the $x$ and $y$ axes.

## 4.1 Maximising Escape Time for a Fixed Boundary Perimeter

Maximising the escape time means attempting to find a shape with the largest expected value for the spider's escape with a fixed perimeter.

Thus to maximise the escape time we must design the trap such that it is the farthest from the origin in all possible directions. Further, it is necessary to ensure all boundary parts remain at an approximately equal maximum distance from the origin. This ensures that the spider requires more steps on average to encounter any part of the boundary.

A circle centred at the origin serves as an intuitive choice for such a design, but this would counter the restrictions put in place, namely, *the curve must be placed on a discrete grid.* Thus on a discrete grid, a "circle" would be represented by the set of grid points that are closest to the continuous circle's boundary. This might look more like a diamond or a square depending on the grid resolution and the circle's size.

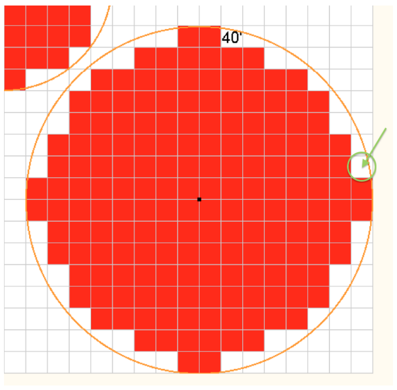

Figure 6: Circle superimposed onto a discrete grid.

## 4.2 Minimising the Escape Time

To optimize the spider's escape time within a lattice grid, we consider the spatial relationship between the boundary of the trap and the spider's starting position, or origin. The foundational principle is to minimize the time it takes, on average, for the spider to reach the boundary. We explore this in detail below:

### 4.2.1 Distance Considerations

- **Average and Minimum Distance to Boundary:** The quickest way to ensure a reduced escape time is to minimize both the average and the minimum distances from all possible starting positions within the boundary to the boundary itself. When considering a specific starting point, such as the origin, the boundary should be as proximate as possible. This ensures that the spider frequently reaches the boundary in a shorter amount of steps, thereby reducing $\mathbb{E}[T]$, the average escape time.

### 4.2.2 Probability Considerations

- **Escape Probability for Adjacent Points:** Beyond the distance to the boundary, the spider's probability of escape when adjacent to the boundary is crucial. Ideally, the boundary's design should maximise the likelihood of the spider moving onto the boundary in its subsequent move.

### 4.2.3 Geometric Implications

- **Ideal Boundary Shape:** A boundary shape that embodies the aforementioned principles would envelop the origin tightly. Given our lattice structure, a square shape encapsulating the origin is optimal. It ensures the shortest distance to the boundary from any point within.
- **Constraints and Modifications:** If we are operating under constraints, such as a predetermined perimeter, then the boundary might require modifications. For instance, to maintain a set perimeter, the square can have extensions or "arms" protruding from it. While these arms might not significantly influence the escape time from the origin, they allow us to meet the given boundary length constraint.

In summary, to minimize $\mathbb{E}[T]$, our boundary design should focus on proximity to the origin and ensure that the spider's trajectory towards and along the boundary is as unhindered as possible.

Star shapes or cross shapes with *arms* protruding towards the origin may be the first choices coming to mind. The depth and count of these protrusions can be tweaked to match the fixed boundary length while ensuring parts of the boundary remain proximal to the origin.

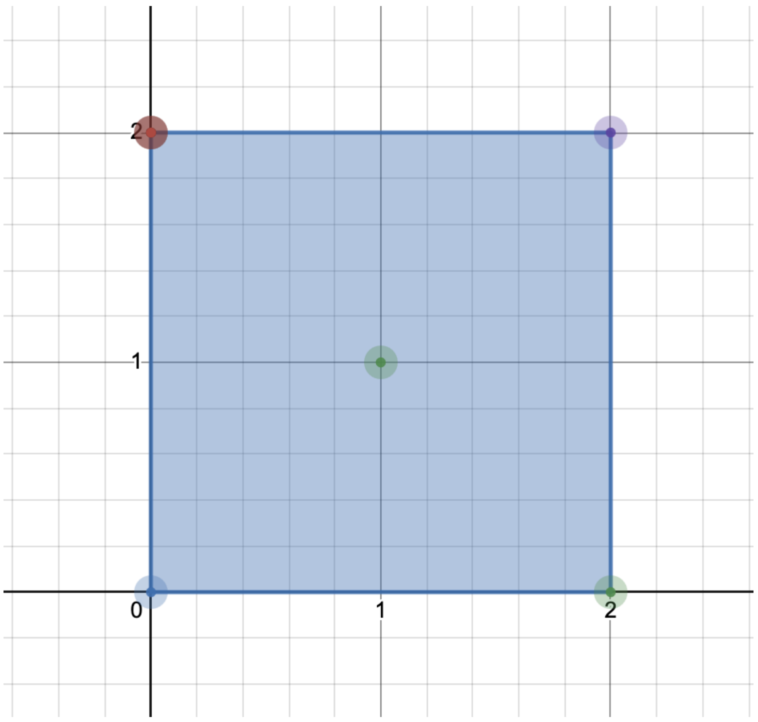

Figure 7: A 2x2 rectangle, where the chance of escaping is 1, being most efficient with perimeter.

## 4.3 Testing Hypotheses

### 4.3.1 Maximum Escape Time

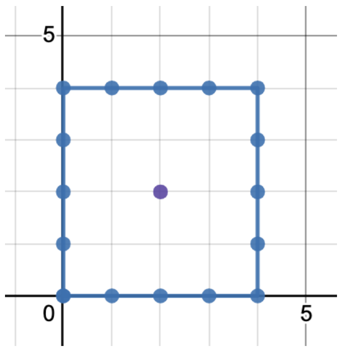

Figure 8: A 4x4 rectangle, where the chance of escaping is the least.

$$\mathbb{E}(T) = 4.5$$

After 119 iterations of $n$.

### 4.3.2 Minimum Escape Time

$$\mathbb{E}(T) = 1$$

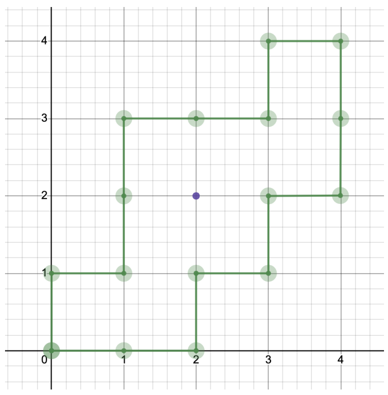

Figure 9: A series of squares that surround the origin.

After 1 iteration of $n$.

### 4.3.3 Summarising Findings

By testing multiple shapes with the same perimeter, our postulates have been shown true.

## 5 Solving Problem 15 C

Although investigated, were unable to come up with a viable solution for this problem in the given time frame. We did find the expected values of a few shapes that intuitively seem similar for the spider to escape from geometrically distinct (i.e. shapes that have no relation by rotating or reflecting).

A square's expected value was found. Then, a bulge was created on one side of the square and an indentation of equal size on the other. The expected escape times were different and this is the extent of our scope on this question.

## 6 Solving Problem 15 D

To approach this problem, a new program was developed that would graph the distribution of $n$ i.e each probability of hitting the boundary for each $n$ steps.

Without employing more complex techniques, some basic inferences about the boundary curve can be made.

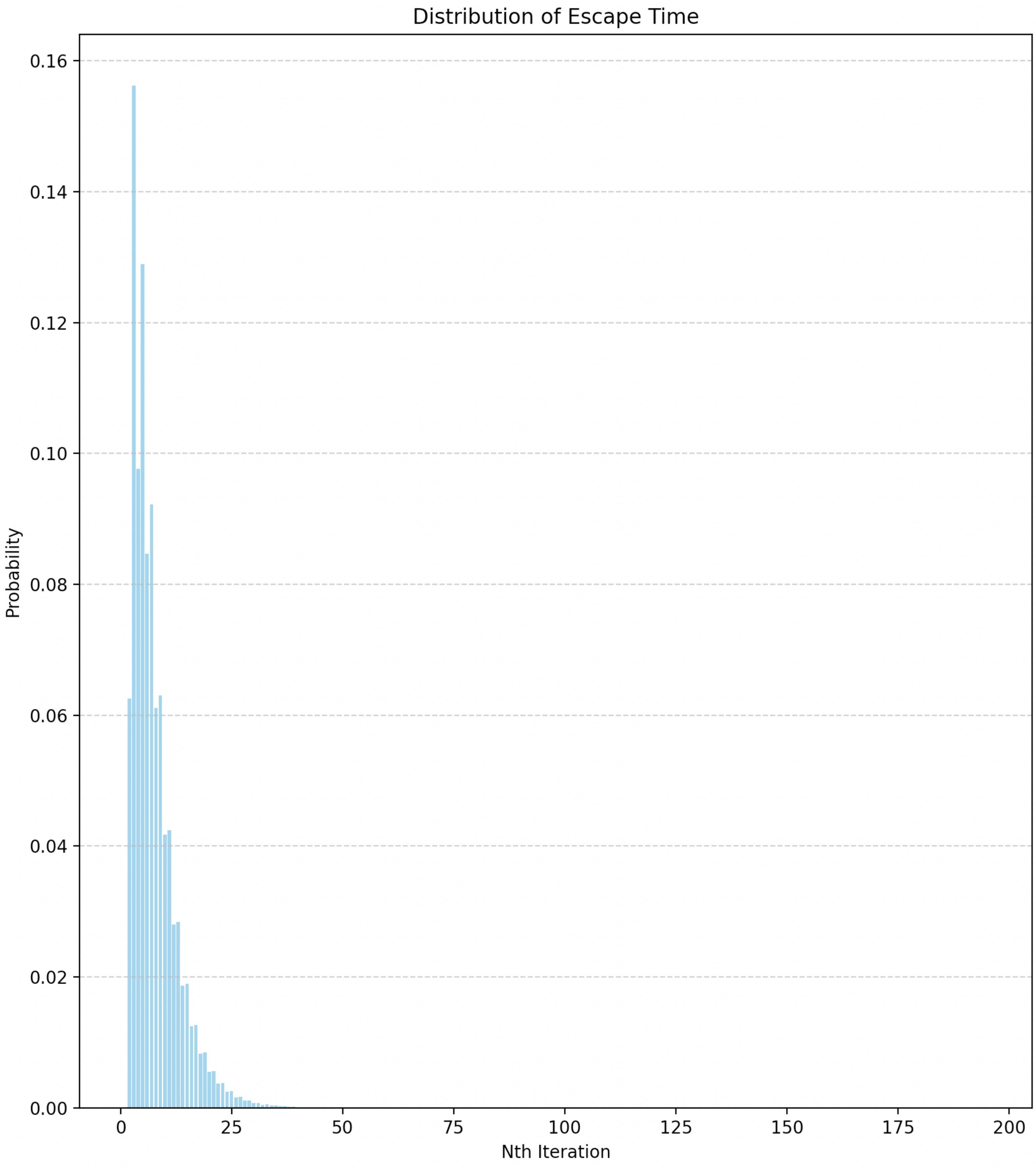

Figure 10: An example distribution of the boundary from Problem 15A.

### 6.0.1 The Closest Point

The first $n$ where probabilities are non-zero means that the spider took $n$ steps to arrive at the boundary. From this, we can conclude that the first $n$ with a non-zero probability is the distance from the origin to the closest boundary point. For example, for a given $\Gamma$, if probabilities are non-zero starting at $n = 3$, then we know that the boundary is at least 3 units away from the origin in any straight direction.

### 6.0.2 Peak of Distribution

A sharp peak at an early iteration indicates that the spider frequently escapes the region quickly. This could mean the spider starts close to the boundary, which is further backed up by 6.0.1.

### 6.0.3 Spread of Distribution

A narrow distribution (with probabilities concentrated around a specific iteration) might suggest a symmetric or regular boundary shape. The spider likely has a consistent experience trying to escape, regardless of the direction it chooses. A wide distribution indicates variability in the spider's escape time. This could be due to irregularities or asymmetries in the boundary.

### 6.0.4 Multiple Peaks

If the distribution has multiple peaks, it could hint at multiple typical escape scenarios. For instance, the spider might frequently escape quickly when heading in one direction (first peak) but take longer when heading in another direction due to a different boundary configuration (second peak).

### 6.0.5 Initial Probabilities

If a non-zero probability of escape exists at $n = 1$, this means that there is a boundary point directly reachable by moving NESW one step. This similarly repeats for $n = 2$, etc. as aforementioned in 6.0.1.

### 6.0.6 Skewness of Distribution

A distribution skewed to the right indicates the spider often takes longer than the average time to escape, which might suggest a larger boundary or a starting point far from the boundary. A distribution skewed to the left suggests the spider frequently escapes fast, meaning a small boundary.

#### 6.0.7 Obtaining a Transition Matrix

Since the transition matrices involve clearly outline the points of the boundary, that is, the rows that have all 0s for their columns, obtaining the transition matrix at any point would allow us to easily determine the exact shape of the original boundary.

A complicated, yet possible method of doing so, would be optimisation, and treating the difference between some arbitrary transition matrix and the probability distribution as a loss. Minimising this loss through gradient descent or another optimisation method would obtain the original matrix or something close to it. However, this is outside of the time frame of the project.

# 7 Summary and References

## 7.1 Summary and Conclusion

Throughout this paper, we have given a solution to Problem 15 which revolves around analysing the movement of a spider on a two-dimensional integer lattice, starting from an origin point, $O$. The spider's movement is governed by random decisions, where it has an equal chance (probability of 0.25) to move in any of the four cardinal directions: up, down, left, or right. This study's essence lies in understanding the spider's escape time, symbolised as $\mathbb{E}(T)$, denoting the number of steps it takes to reach a boundary curve $\Gamma$.

The spider's journey on the lattice honours the Markov property as each step relies solely on the present location and not its historical route. Representing this movement, a transition matrix can be created with the probabilities for each possible move. Importantly, when the spider reaches the boundary curve, $\Gamma$, it can be regarded as encountering a *null state*, from which it doesn't return.

**Approach and Methodology:** We utilised a programmatic approach to solve Problem 15 A due to the computational complexities involved in manually handling the matrix.

Key steps involved:

- Creating a transition matrix, $A$, for spider movements with boundary curves as null states.
- Computation of the matrix's $n$-th power, $A^n$, representing probabilities after $n$ steps.
- Iteratively summing probabilities to determine the escape likelihood until the escape probability becomes negligible.
- Calculating the expected escape time, $\mathbb{E}(T)$, using a recursive relationship.

**Results:** Our methodology yielded an expected escape time of $\mathbb{E}(T) = 7.520354455020274$ steps for the given shape $\Gamma$. This was reached after $n = 195$ iterations and was consistent across runs. The result slightly deviates from the initial assumption of it being near the shortest path to the furthest boundary point, which is 6.

**Verification and Reliability:** Verification using a Monte Carlo simulation produced a closely aligned result of $E(T) \approx 7.52$ steps, thereby supporting the accuracy of our approach. Nevertheless, the Markov Chain method's precision is superior, and it should be the method of choice over the Monte Carlo, which primarily demonstrated the Markov method's reliability.

**Moving Forward** Through reflecting upon the methodologies and outcomes of this study, it is evident that there is significant potential to expand upon this question.

1. **Advanced Computational Models:** While the Markov Chain and Monte Carlo methods have proven effective, leveraging neural networks and other machine learning models could refine our predictions, especially in more intricate lattice structures or when introducing further constraints as well as providing more efficient means of computation.
2. **Dynamic Boundaries:** This study has treated the boundary curve, $\Gamma$, as a static entity. We might investigate how dynamically evolving boundaries, possibly due to external factors, affect the spider's escape time.
3. **Multi-agent Dynamics:** Introducing multiple spiders or agents, each with its movement rules, could present a fascinating multi-agent system study. Interactions between agents and competition to reach the boundary first would provide a deeper understanding of escape dynamics.
4. **Three-Dimensional Exploration:** The current study revolves around a two-dimensional lattice. Extending the model into the third dimension might unravel intriguing dynamics, providing an even more holistic view of the spider's behaviour.
5. **Impact of External Factors:** Introducing factors such as pheromones which might influence the spider's decision-making could provide a biologically more accurate representation and lead to unique insights into movement behaviour.
6. **Iterative Refinements:** Our current solution to Problem 15 A is elegant and while the expected escape time has been identified for the current shape $\Gamma$, it would be interesting to test various other geometric confines, exploring how different structures affect escape dynamics.
7. **Boundary Shape Optimisation:** Delving deeper into the geometric shapes of boundaries to ascertain which forms yield the quickest or slowest escape times might unveil a series of optimal shapes for different scenarios.

Our research offers an approach to understanding a spider's movement in a confined lattice space. However, there is a vast amount of further improvement and research which could be done to deepen an understanding of escape dynamics and behavioural analysis.

**Solving Problem 15 B: Spider Escape Dynamics** Problem 15 asks us to find boundary shapes that maximise and minimise $\mathbb{E}(T)$.

- It was assumed that Γ is an enclosed shape on a discrete grid.
- For maximizing escape time, a grid-represented circle (looking like a diamond or square) is optimal.
- For minimising escape time, boundaries close to the origin, such as squares, are effective. Star or cross shapes with arms directed towards the origin can efficiently balance perimeter constraints and proximity to the origin.

**Solving Problem 15 C: Exploring Geometric Variations** While we were unable to find a solution to Problem 15 C, we were able to theoretically discern differing escape times from geometrically distinct shapes, such as bulges and indentations.

**Solving Problem 15 D: Distribution Insights** Although we were unable to find a distinct solution we did note some observations:

- The closest point to the origin can be identified by the first non-zero probability iteration.
- Peaks in the distribution may indicate escape scenarios, while the spread can hint at boundary regularity.
- The skewness of the distribution can give insights into the size and proximity of the boundary relative to the spider's starting position.

We also theorised that optimisation techniques, though complex, could potentially be used to derive the original boundary shape from a given probability distribution, though this was beyond our project's time frame.

**Final Remarks:**

While we have made significant progress in understanding the spider's escape dynamics and the influence of boundary configurations, 2 problems remain largely unsolved. In future investigations, it would be possible to refine our methodologies and delve deeper into these intriguing questions.

## 7.2 References and Links

### 7.2.1 Code

`https://github.com/alislaboratory/MSRC2023`

## References

[1] Normalized Nerd (2021) *Markov Chains Clearly Explained!* `https://www.youtube.com/watch?v=i3AkTO9HLXo&t=383s`

[2] S. Popov (2021) *Two-Dimensional Random Walk*

[3] Wikipedia (2023) *Stochastic Matrix* https://en.wikipedia.org/wiki/Stochastic_matrix

[4] Wikipedia (2023) *Markov Chain* https://en.wikipedia.org/wiki/Markov_chain

[5] Wikipedia (2023) *Stochastic Process* https://en.wikipedia.org/wiki/Stochastic_process

[6] Wikipedia (2023) *Stochastic Process* https://en.wikipedia.org/wiki/Markov_property

[7] Wikipedia (2023) *Discrete-time Markov Chain* https://en.wikipedia.org/wiki/Discrete-time_Markov_chain

[8] Spitzer, F. (1976). Principles of Random Walk, Springer-Verlag.

[9] K. Pearson (1905) *The Problem of the Random Walk*

[10] K. L. Chung (1975) *Elementary Probability Theory with Stochastic Processes*

[11] Bookdown *Chapter 10 - Markov Chains* https://bookdown.org/probability/beta/markov-chains.html

## 8 Code

### 8.1 Markov Chain Approach

```
// markov.py
### WRITTEN BY ALI ASHRAFY 2023
### MATHEMATICS AND STATISTICS RESEARCH COMPETITION
# 13 Aug 2023

### This code is the Markov Chain approach to random walks. Our paper can be
    found at the GitHub:
# https://github.com/alislaboratory/MSRC2023

import numpy as np
from pprint import pprint
import matplotlib.pyplot as plt


def plot_distribution(n_distribution):
    # Extracting keys and values
    iterations = list(n_distribution.keys())
    probabilities = list(n_distribution.values())

    # Plotting
    plt.figure(figsize=(10, 6))
    plt.bar(iterations, probabilities, color='skyblue')
    plt.xlabel('Nth Iteration')
    plt.ylabel('Probability')
    plt.title('Distribution of Escape Time')
    plt.grid(axis='y', linestyle='--', alpha=0.7)
    plt.tight_layout()
    plt.show()



def create_grid(n, m): # create a n rows x m columns grid
    grid = []
    for rows in range(n):
        row = []
        for columns in range(m):
            row.append((columns, rows))

        grid.append(row)

    return grid


# note: we are using 0,0 as top left - can convert later
# note: always make the grid a lot bigger than it needs to be because it is
    technically an infinite grid.


def create_transition_matrix(
    n, m, grid
):  # Create the initial starting transition matrix
    # initially storing as a 2D dictionary to allow indexing by tuples
    # then convert to numpy for matrixing
    matrix = {}
    temp_array = []
    for a in range(n):
        for b in range(m):
            temp_array.append(grid[a][b])

    for i in range(n * m):
        temp_row = {}
        current_from = temp_array[i]
        x = current_from[0]
        y = current_from[1]
        for j in range(n * m):
            current_to = temp_array[j]
            temp_row[current_to] = 0
            # no edge cases on the edge of the grid need to be taken into the
                account as the grid will always be bigger than required
                (infinite lattice grid)
            if current_to in [
                (x, y - 1),
                (x + 1, y),
                (x, y + 1),
                (x - 1, y),
            ]:  # is it immediately surrounding the ting
                temp_row[current_to] = 0.25

        matrix[current_from] = temp_row

    # now convert to matrix
    # pprint(matrix)
    new_matrix = np.zeros(shape=(m * n, m * n))
    for i in range(n * m):
        for j in range(n * m):
            new_matrix[i][j] = matrix[temp_array[i]][temp_array[j]]

    return new_matrix


# create stuffs


def np_to_dict(matrix, grid): # Unused at the moment
    dicmatrix = {}
    coords = []
    # generate coords
    for i in grid:
        for j in i:
            coords.append(j)

    # generate dictionary
    for i in range(len(coords)):
        temp_row = {}
        for j in range(len(coords)):
            temp_row[coords[j]] = matrix[i][j]

        dicmatrix[coords[i]] = temp_row

    return dicmatrix

gamma = [ # The coordinates for 15a gamma
    (6, 8),
    (7, 8),
    (8, 8),
    (8, 9),
    (9, 9),
    (10, 9),
    (10, 8),
    (10, 7),
    (10, 6),
    (10, 5),
    (9, 5),
    (8, 5),
    (8, 4),
    (8, 3),
    (8, 2),
    (7, 2),
    (6, 2),
    (5, 2),
    (4, 2),
    (4, 3),
    (4, 4),
    (4, 5),
    (3, 5),
    (2, 5),
    (2, 6),
    (2, 7),
    (2, 8),
    (2, 9),
    (3, 9),
    (4, 9),
    (4, 8),
    (5, 8),
]
grid_size = (12,12)
start_pos = (6,6)


# gamma = [ (1, 1), # The minimisation
#     (2, 1),
#     (3, 1),
#     (3, 2),
#     (3, 3),
#     (4, 3),
#     (4, 4),
#     (3, 4),
#     (2, 3),
#     (1, 3),
#     (1, 2),
#     (0, 1),
#     (0, 0),
#     (1, 0)]
# grid_size =(6,6)
# start_pos = (2,2)

# box 6x6
# gamma =[(1,4),
#(7,4),
#(7,7),
(6,1),
(7,2),
(7,6),
(1,2),
(1,7),
(1,1),
(1,6),
(7,5),
(7,3),
(2,1),
(3,1),
(4,1),
(5,1),
(7,1),
(6,7),
(5,7),
(4,7),
(3,7),
(2,7),
(2,1),
(1,5),
(1,3)]
# grid_size = (8,8)
# start_pos = (4,4)
# 10.715236686390538

# box with bulge and indentation 6x6
# gamma = [(7,7),
(6,1),
(7,2),
(7,6),
(1,2),
(1,7),
(1,1),
(1,6),
(7,5),
(7,3),
(2,1),
(3,1),
(4,1),
(5,1),
(7,1),
(6,3),
(6,4),
(6,5),
(6,7),
(5,7),
(4,7),
(3,7),
(2,7),
(0,5),
(0,4),
(0,3),
(2,1),
```

```python
(1,5),
(1,3)]
# grid_size = (8,8)
# start_pos = (4,4)
# 8.790242996856136


# gamma = [ (i[0], i[1]+2) for i in gamma] # use this for offsetting
# print(gamma)


grid = create_grid(grid_size[0], grid_size[1])
matrix = create_transition_matrix(grid_size[0], grid_size[1], grid)
n_distribution = {}

summ = 0
n = 1
prev = matrix
last = -1
debug = True
for (
    coord
) in (
    gamma
):  # Set the boundary curve points to be absorbing i.e the spider stays there
    if it reaches it
    for i in range(len(matrix[coord[0] * len(grid) + coord[1]])):
        matrix[coord[0] * len(grid) + coord[1]][i] = 0

    matrix[coord[0] * len(grid) + coord[1]][coord[0] * len(grid) + coord[1]] =
        0

# TESTING #
# m = 1
# while m < 100:
#     prev = np.matmul(prev, matrix)
#     print(prev[6 * len(grid) + 6][0 * len(grid) + 0])

#     m += 1


while True: # Expected value is sum of n x P(escape time = n) to n = infinity
    currentprob = 0

    for coord in gamma:
        currentprob += prev[start_pos[0] * len(grid) + start_pos[1]][
            coord[0] * len(grid) + coord[1]
        ]  # Find each probability for each gamma coordinate

    summ += n * currentprob # Add to the expected value sum.
    # Debugging statements
    if debug:
        print(n)
        print(f"Sum: {summ}")
        print(f"Escape prob for {n} steps: {currentprob}")
        print()
    #

    if last == summ and n>grid_size[0]: # Break when last value = current i.e
        converged
        print(n)
        print("Yahoo!") # Yahoo when converged
        break

    last = summ
    n_distribution[n] = currentprob
    prev = np.matmul(prev, matrix) # Multiply the transition matrices
    n += 1 # Increment n



print(summ)
plot_distribution(n_distribution=n_distribution) # Plot distribution (comment
    out if not working on Problem D as it will cause trouble)
```

## 8.2 Monte Carlo Simulation

```
### WRITTEN BY ALI ASHRAFY 2023
### MATHEMATICS AND STATISTICS RESEARCH COMPETITION
# 13 Aug 2023

### This code is the Monte Carlo/simulated approach to random walks. Our
    paper can be found at the GitHub:
# https://github.com/alislaboratory/MSRC2023



import random



def monte_carlo_simulation(start, gamma, trials=1000000):
    total_steps = 0

    for _ in range(trials):
        x, y = start
        steps = 0
        while (x, y) not in gamma:
            direction = random.choice(["up", "down", "left", "right"])
            if direction == "up":
                y -= 1
            elif direction == "down":
                y += 1
            elif direction == "left":
                x -= 1
            elif direction == "right":
                x += 1
            steps += 1
        total_steps += steps

    return total_steps / trials


# Running the Monte Carlo simulation for the spider problem


start_position = (6, 6)
gamma = [ # The coordinates for boundary curve gamma
    (6, 8),
    (7, 8),
    (8, 8),
    (8, 9),
    (9, 9),
    (10, 9),
    (10, 8),
    (10, 7),
    (10, 6),
    (10, 5),
    (9, 5),
    (8, 5),
    (8, 4),
    (8, 3),
    (8, 2),
    (7, 2),
    (6, 2),
    (5, 2),
    (4, 2),
    (4, 3),
    (4, 4),
    (4, 5),
    (3, 5),
    (2, 5),
    (2, 6),
    (2, 7),
    (2, 8),
    (2, 9),
    (3, 9),
    (4, 9),
    (4, 8),
    (5, 8),
]
monte_carlo_expected_time = monte_carlo_simulation(start_position, gamma)
print(monte_carlo_expected_time)
```