The Discrete and Continuous Probability Functions of Random Walkers

Introduction:

A random walk is the movement of a particle in space by a random mechanism. In this investigation, these randomly moving particles and their positions will be referred to as random walkers. For example, the number of heads of a coin flipped repeatedly may be seen as a random walk in which the value either remains constant or increases by one with each move and whose movements have equal probability. A biased random walk is one in which all movements do not have equal probability. The movement of bacteria in a solution follows a biased random walk in which the probability of traveling in the direction of high nutrient concentrations is greater. The probability mass and distribution functions of a random walk may be of use for modeling random walks in the various fields (physics, biology, economics etc.) where random walks appear.

Aim and Rationale

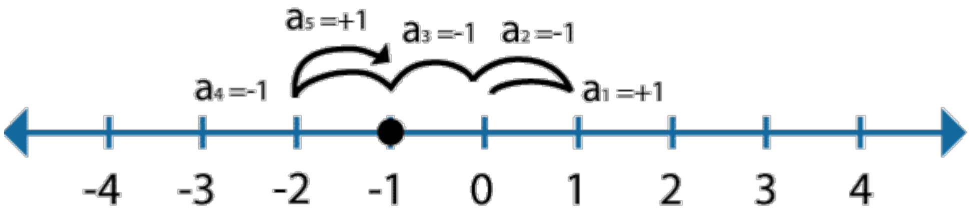
The aim of this experiment is to answer the research question, "What is the probability that a walker which moves randomly will be at a point after a number of steps?" As a variety of possibilities for movement exist, the analysis of this question will be separated into two parts. In part 1, the probability mass function of the discrete position of a walker whose movements are a fixed length will be explored in one and multiple dimensions. In part 2, the probability distribution function of a walker whose movement follows its own probability distribution function will be explored in one and multiple dimensions.

I first chose to investigate random walks because of my interest in biology and was quickly absorbed by the complexity that lies behind the seemingly simple problem. Breaking the problem into discrete and continuous movement then into one and multidimensional movement required approaches utilizing probability, simulation, and calculus. Data and calculations provided by computer programs, coded by the author, simulating random walkers, are used when necessary or beneficial to reinforce and supplement mathematically derived conclusions. The code for these programs are provided in the appendices.

Part 1: Discrete Movement:

Discrete One-Dimensional Movement:

This investigation will begin by exploring the probability distribution of the position of a random walker after *n* steps of constant length in one of two opposite directions. The position of the walker after n steps may be represented on a number line for which one movement direction is considered positive or right and whose units are the length of a single step. The picture below depicts the path of a particular random walker that has taken 5 random 1 unit long steps.



Let *f(x, n)* be the probability mass function that a walker's position, *X*, is *x* after *n* steps. If the probability of a movement right remains constant, at *p*, the number of right movements after *n* steps is a binomially distributed variable.

$$r \sim B(n,p)$$

and

$$P(R = r) = \binom{n}{r} p^r p^{n-r}$$

To reach a position *x* after *n* movements, a walker will have moved right *r* times and left *l* times. Observe,

$$n = r + l \text{ and } x = r - l$$

$$l = n - r$$

$$x = 2r - n$$

$$r = \frac{x+n}{2}$$

For a position x to be possible, ie. $f(x,n) \neq 0,\ r \in \mathbb{Z}$. Thus, $f(x,n) = 0$ where $\frac{x+n}{2} \notin \mathbb{Z}$. In other words, $f(x,n) = 0$ where $x \bmod 2 = n \bmod 2$. Thus,

$$f(x,n) = P\left(R = \frac{x+n}{2}\right) = \binom{n}{\frac{x+n}{2}} p^{\frac{x+n}{2}} p^{\frac{x-n}{2}}, \qquad \frac{x+n}{2} \in \mathbb{Z}$$

Discrete Multi-Dimensional Movement:

Independent Movement Across Dimensions:

For simplification, one may begin multi-dimensional analysis by exploring the probability mass function of walkers whose movement lies in only two perpendicular dimensions, x_1 and x_2. Examine the situation in which, during each step, the walker moves either up or down in each dimension independently. Let the position of a point in space be represented as the sequence $\{x_1,\ x_2\}$, with the probabilities of moving in the positive direction $\{p_1,\ p_2\}$. The probability mass function after one move is shown below.

x_1

		$1-p_1$	0	p_1
	p_2	$(1-p_1)$ $*\ p_2$	0	$p_1 *\ p_2$
x_2	0	0	0	0
	$1-p_2$	$(1-p_1)$ $* (1-p_2)$	0	$(1-p_2)$ $*\ p_1$

As seen in the previous section, the probability of $x_n = A$ after *n* steps is $f(A,n)$. As the x_1 position and x_2 position are independent events,

$$P(x_1 = A\ \cap\ x_2 = B) = P(x_1 = A\,) *\ P(\,x_2 = B) = f_1(A,n) * f_2(B,n)$$

Thus,

$$P(x_1 = A\ \cap\ x_2 = B) = \binom{n}{\frac{A+n}{2}} p_1^{\frac{A+n}{2}} p_1^{\frac{A-n}{2}} * \binom{n}{\frac{B+n}{2}} p_2^{\frac{B+n}{2}} p_2^{\frac{B-n}{2}}$$

For d dimensions, let the position of a point in space be represented as the sequence $\{x_1,\ x_2, \ldots,\ x_d\}$ and let the probabilities of moving in the positive direction in each dimension be $\{p_1,\ p_2\ \ \ldots,\ p_d\}$. Just as before, the probabilities of movement in every dimension are independent. Thus,

$$P(position =\ \{x_1,\ x_2, \ldots,\ x_d\}) = \prod_{i=1}^{d} f_i(x_i, n) = \prod_{i=1}^{d} \binom{n}{\frac{x_i + n}{2}} p_i^{\frac{x_i+n}{2}} p_i^{\frac{x_i-n}{2}}$$

Part 2: Continuous Movement :

Continuous One-Dimensional Movement:

One may next investigate the probability distribution of the position of a random walker after n steps of magnitude less than or equal to one unit. The probability distribution function of the x position of the walker after n steps shall be $f_n(x)$. After one step, the piecewise function $f_1(x)$ may be defined as:

$$f_1(x) = \begin{cases} c, & -1 \leq x \leq 1 \\ 0, & x < -1 \cup x > 1 \end{cases}$$

The area under a pdf for a given domain is the probability that the value will lie within that domain. As the total probability must equal 1, the area under $f_1(x)$ from -∞ to ∞ must be 1. As $f_1(x)$ is only nonzero within $-1 \leq x \leq 1$, the area of the rectangle with height *c* and length 1– $(-1) = 2$ must be 1.

$$2\ *\ c\ =\ 1$$

$$c\ =\ \frac{1}{2}$$

$$f_1(x) = \begin{cases} \frac{1}{2}, & -1 \leq x \leq 1 \\ 0, & x < -1 \cup x > 1 \end{cases}$$

The probability that a walker is at a point α after *n* steps is the infinite sum of the probabilities for all points *x* after *n-1* steps, $f_{n-1}(x)$ times the respective probability of moving from *x* to α $f_1(\alpha - x)$.

$$f_n(\alpha) = \int_{-\infty}^{\infty} f_{n-1}(x) * f_1(\alpha - x)\, dx$$

As,

$$f_n(\alpha - x) = \begin{cases} \frac{1}{2}, & \alpha - 1 \leq x \leq \alpha + 1 \\ 0, & x < \alpha - 1 \cup x > \alpha + 1 \end{cases}$$

The integral may be separated into

$$f_n(\alpha) = \int_{-\infty}^{\alpha-1} 0 * \, f_{n-1}(x)\,dx + \int_{\alpha-1}^{\alpha+1} \frac{1}{2}\, f_{n-1}(x)\,dx + \int_{\alpha+1}^{\infty} 0 * \, f_{n-1}(x)\,dx$$

Thus,

$$f_n(\alpha) = \frac{1}{2}\int_{\alpha-1}^{\alpha+1} f_{n-1}(x)\,dx$$

Thus, $f_n(\alpha)$ will be determined by the sections of $f_{n-1}(x)$ within the domain $\alpha - 1 \le x \le \alpha + 1$.

$f_2(\alpha)$ may, then, be derived like such:

$$f_2(\alpha) = \begin{cases} \dfrac{1}{2}\left(\displaystyle\int_{\alpha-1}^{1} \frac{1}{2} dx + \int_{1}^{\alpha+1} 0\,dx\right), & 0 \le \alpha \le 2 \\ \dfrac{1}{2}\left(\displaystyle\int_{\alpha-1}^{-1} 0\,dx + \int_{-1}^{\alpha+1} \frac{1}{2} dx\right), & -2 \le \alpha \le 0 \\ 0, \quad \alpha < -2 \cup \alpha > 2 \end{cases}$$

$$= \begin{cases} \dfrac{1}{4}(2-\alpha), & 0 \le \alpha \le 2 \\ \dfrac{1}{4}(2+\alpha), & -2 \le \alpha \le 0 \\ 0, \quad \alpha < -2 \cup \alpha > 2 \end{cases}$$

Similarly,

$$f_3(\alpha) = \begin{cases} \dfrac{1}{2}\left(\displaystyle\int_{\alpha-1}^{2} \frac{1}{4}(2-x)\,dx + \int_{2}^{\alpha+1} 0\,dx\right), & 1 \le \alpha \le 3 \\ \dfrac{1}{2}\left(\displaystyle\int_{\alpha-1}^{0} \frac{1}{4}(2+x)\,dx + \int_{0}^{\alpha+1} \frac{1}{4}(2-x)\,dx\right), & -1 \le \alpha \le 1 \\ \dfrac{1}{2}\left(\displaystyle\int_{\alpha-1}^{-2} 0\,dx + \int_{-2}^{\alpha+1} \frac{1}{4}(2+x)\,dx\right), & -3 \le \alpha \le -1 \\ 0, \quad \alpha < -3 \cup \alpha > 3 \end{cases}$$

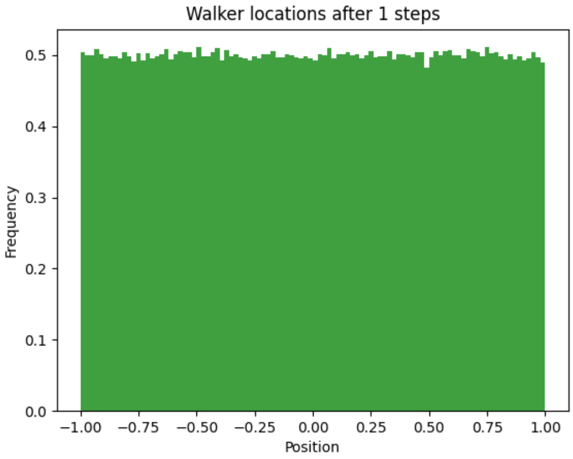
$$f_3(\alpha) = \begin{cases} \frac{1}{16}(3-\alpha)^2, & 1 \leq \alpha \leq 3 \\ \frac{1}{8}(3+\alpha), & -1 \leq \alpha \leq 1 \\ \frac{1}{16}(3+\alpha)^2, & -3 \leq \alpha \leq -1 \\ 0, & \alpha < -3 \cup \alpha > 3 \end{cases}$$

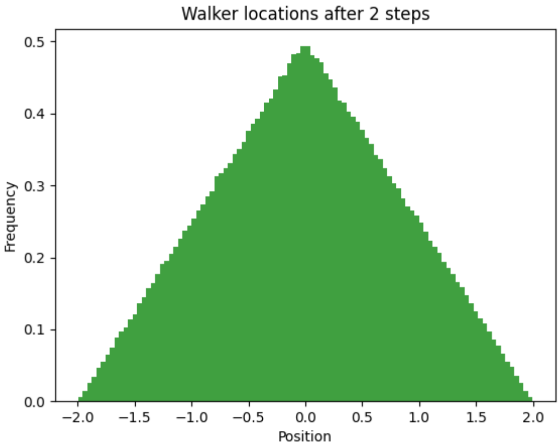
A program was written in Python3 to automate $f_n(\alpha) \rightarrow f_{n+1}(\alpha)$ and confirm observed patterns. The polynomial sections of $f_n(\alpha)$ up to n = 5 and the source code may be found in **APPENDIX A**. Through iterating this process, one observes the following patterns:

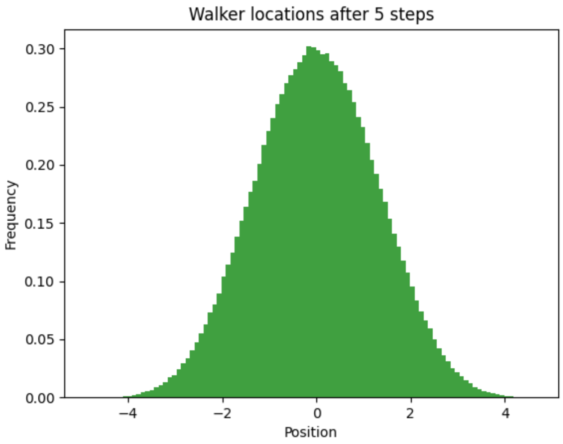
1. $f_n(x)$ is a piecewise function composed of exactly *(n – 1)* non-zero, 2 unit long sections. We will label $s_n^m(x)$ be the *m*th non-zero section, starting with 1, counting from the right.
2. As the probability density function is symmetrical about the origin, if $s_n^m(x)$ is a section of the function, $s_n^m(-x)$ is the section $s_n^{n-1-m}(x)$ of the function.
3. $s_n^1(x) = \frac{1}{2^n(n-1)!}(n+x)^{n-1}$
4. All coefficients of $s_n^m(x)$, when multiplied by $2^n(n-1)!$, are integers. One may thus predict that $\frac{1}{2^n(n-1)!}$ is a multiplicative factor of the general formula for $s_n^m(x)$.
5. The leading coefficients of the sections of $f_n(x)$, when multiplied by $2^n(n-1)!$, are the binomial coefficients. In other words, the leading coefficient of $s_n^m(x)$ is

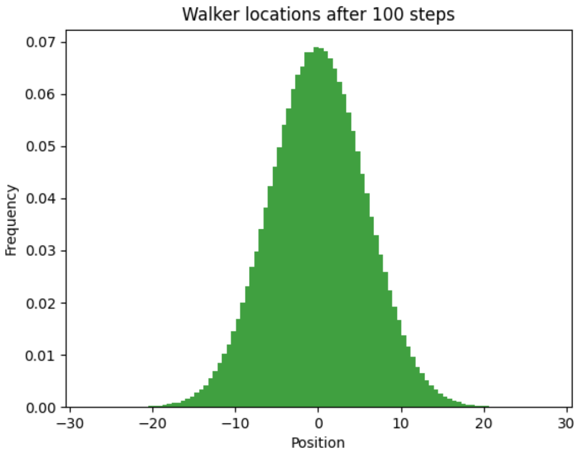
$$\binom{n}{m-1}$$

6. As n is increased, the probability distribution function, $f_n(x)$ approximates a normal distribution. A program was created with Python3 which simulates the movement of random walkers after n steps. The source code can be found in **Appendix B**. For 1000000 random walkers, after 1, 2, 5, and 100 steps, the following distributions were produced.









As $f_n(x)$ is symmetrical about the origin, 50 % of the distribution lies on either side of the origin, so the expected value of $f_n(x)$ will be 0. An interesting expansion upon this discovery is documented in the following section.

7. As n increases to infinity, the finite total probability is spread out over an increasing domain. This results in the curve flattening to 0. As *n* approaches infinity, the probability of the position lying between real numbers, *a* and *b* approaches 0.

$$\lim_{n\to\infty}\int_a^b f_n(x) = 0, \qquad \{a,b\} \in \mathbb{R}$$

This seventh observation provides a basis for the idea of diffusion. Diffusion is the phenomena by which an initially concentrated group of particles in a liquid evenly disperses. Molecules in a liquid follow a random walk called Brownian motion. Each movement of an individual molecule is randomized by collisions with neighboring molecules. As collisions occur at a constant rate, *n* will increase linearly over time. Thus, over time the distribution of particles will disperse evenly throughout a container.

Alternate probability distributions

It should be noted that the equation derived in the previous section,

$$f_n(\alpha) = \int_{-\infty}^{\infty} f_{n-1}(x) * f_1(\alpha - x)\, dx$$

holds true for any initial probability distribution function, $f_1(x)$. For example, one may apply this to the following probability distribution function which is linearly increasing from -1 to 1 and 0 elsewhere on its domain.

$$f_1(x) = \begin{cases} \frac{1}{2}(x+1), & -1 \leq x \leq 1 \\ 0, & x < -1 \cup x > 1 \end{cases}$$

So,

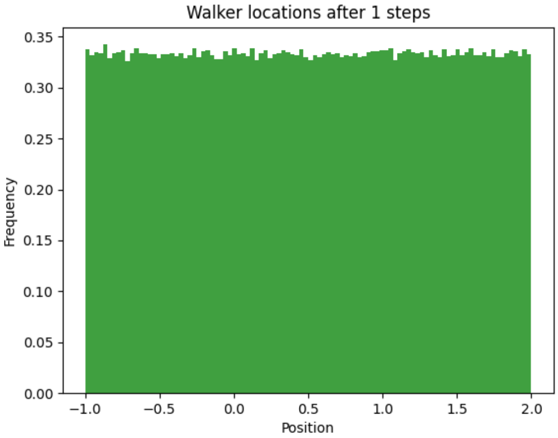
$$f_n(\alpha - x) = \begin{cases} \frac{1}{2}(\alpha - x + 1), & \alpha - 1 \leq x \leq \alpha + 1 \\ 0, & x < \alpha - 1 \cup x > \alpha + 1 \end{cases}$$

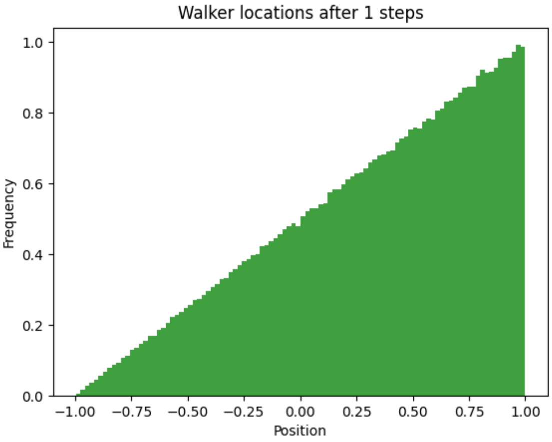
$$f_n(\alpha) = \int_{1-\alpha}^{1+\alpha} f_{n-1}(x) * \frac{1}{2}(\alpha - x + 1)\, dx$$

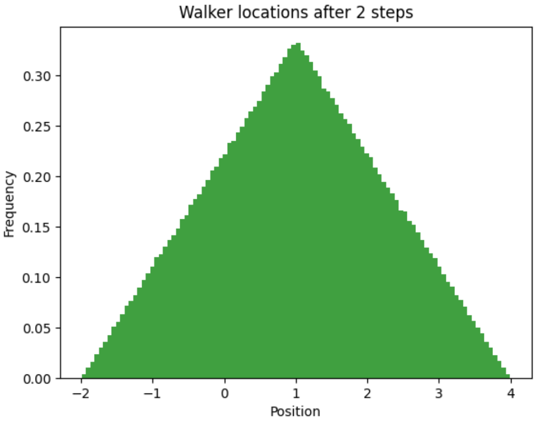
Although computationally intensive, this formula may be used to compute the probability distribution function of any such integrable $f_1(x)$. With a computer program like the one found in **Appendix A**, which quickly performs polynomial integration, one may easily derive the polynomial sections of $f_n(\alpha)$. To further explore alternate probability functions, the following two movement probability functions were selected for simulation.

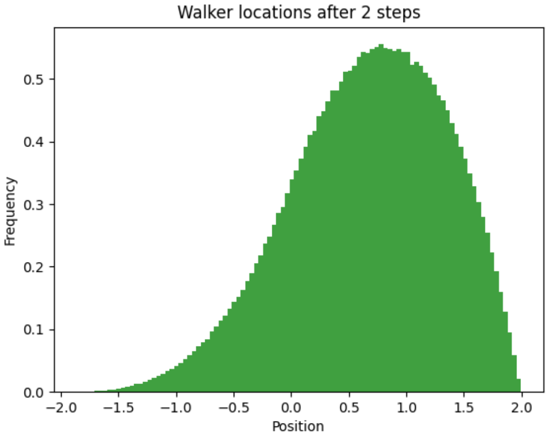
$$f_1(x) = \begin{cases} \frac{1}{3}, & -1 \leq x \leq 2 \\ 0, & x < -1 \cup x > 1 \end{cases} \qquad f_1(x) = \begin{cases} \frac{1}{2}(x+1), & -1 \leq x \leq 1 \\ 0, & x < -1 \cup x > 1 \end{cases}$$

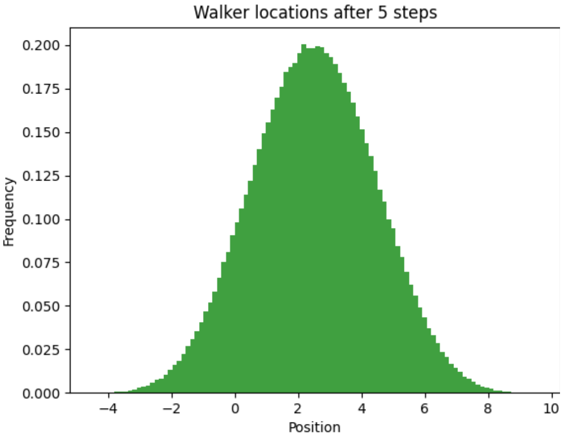
Distributions of 100000 walkers after 1, 2, 5, and 100 steps, for the two functions, are displayed below.

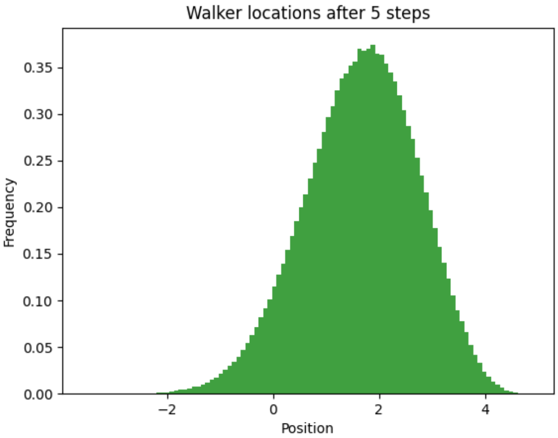


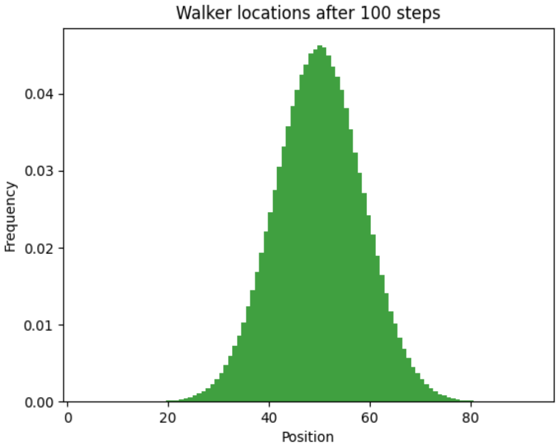


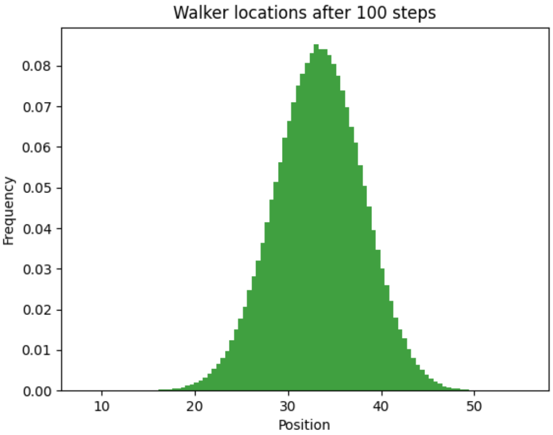








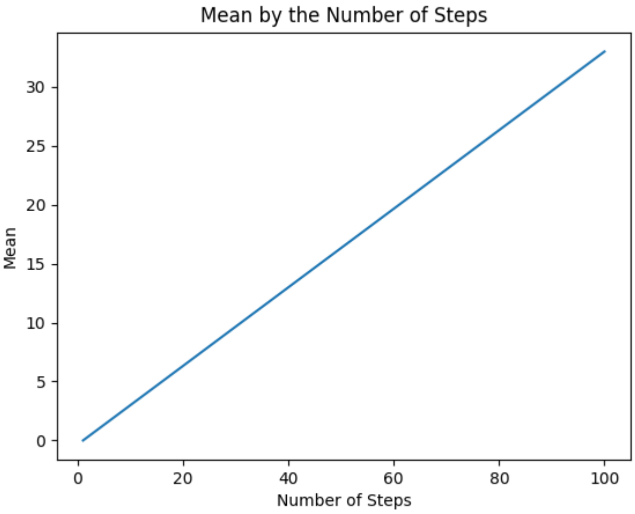




Two observations from the previous subsection appear to hold true for any alternate probability distribution. First, the probability distribution flattens due to an increasing domain. As *n* approaches infinity, the probability of the position lying between real numbers, *a* and *b* approaches 0.

$$\lim_{n\to\infty}\int_a^b f_n(x) = 0, \qquad \{a, b\} \in \mathbb{R}$$

Second, as *n* increases, the probability distribution function approximates to a normal distribution. To explore the nature of this phenomenon, the expected values of $f_n(x)$ were graphed according to n. When graphed, a third observation may be added; the means of the distributions increase linearly.



The expected value of the distribution $f_n(x)$, defined by $f_1(x) = \begin{cases} \frac{1}{2}, & -1 \leq x \leq 1 \\ 0, & x < -1 \cup x > 1 \end{cases}$, remained constant at 0 as n increased. From this, it may be posited that the expected value of $f_1(x)$ is a multiplicative factor of the slope of the line, mean by the number of steps. Further investigation is required to determine whether these conclusions hold for all movement probability density functions, $f_1(x)$, or a subset.

Continuous Multi-Dimensional Movement:

As in the case of discrete multi-dimensional movement, if movement within each dimension has independent probability, the probability of a walker being at a position $\{x_1,\ x_2, \ldots,\ x_d\}$ is the product of the probabilities that the walker is at x_i for each dimension.

$$P(position = \{x_1,\ x_2, \ldots,\ x_d\}) = \prod_{i=1}^{d} f_n(x_i)$$

Conclusion:

From Part 1 and Part 2, we determined equations for discrete random walker movement and documented close analysis of observations of probability distribution functions. The discrete probability mass function, $f(x,n)$, of the position of a walker

$$f(x,n) = \binom{n}{\frac{x+n}{2}} p^{\frac{x+n}{2}} p^{\frac{x-n}{2}}, \qquad \frac{x+n}{2} \in \mathbb{Z}$$

In multiple dimensions, the probability mass function is the product of the independent probability mass functions of each dimension.

$$P(position = \{x_1,\ x_2, \ldots,\ x_d\}) = \prod_{i=1}^{d} f_i(x_i, n) = \prod_{i=1}^{d} \binom{n}{\frac{x_i+n}{2}} p_i^{\frac{x_i+n}{2}} p_i^{\frac{x_i-n}{2}}$$

In Part 2, although no general formula was found for the continuous one-dimensional movement probability density function $f_n(x)$, a method was described, using the following equation, that allows an individual to derive $f_n(x)$ from any $f_1(x)$.

$$f_n(\alpha) = \int_{-\infty}^{\infty} f_{n-1}(x) * f_1(\alpha - x)\, dx$$

From $f_n(x)$, one may then calculate the probability that a walker lies within a defined region. Several key observations were made concerning the nature of $f_n(x)$ for any $f_1(x)$. First, a probabilistic mechanism for diffusion was found in the flattening of $f_n(x)$ as n increased. Second, it was observed that $f_n(x)$ approaches a normal curve as n increases. This observation may be useful in analyzing continuous time random walks in which the distance of a given step approaches 0 and *n* approaches ∞. The last observation is that the expected value of $f_n(x)$ increase linearly with *n*. To conclude Part 2, it was that in multiple dimensions, the probability density function is the product of the independent probability density functions of each dimension.

$$P(position = \{x_1, x_2, \ldots, x_d\}) = \prod_{i=1}^{d} f_n(x_i)$$

Future Research:

No general formula was found for the continuous one-dimensional movement probability density function $f_n(x)$ or a given section $s_n^m(x)$. This could be the subject of future research. In addition, future researchers might search for counterexamples or attempt a proof of the phenomena that $f_n(x)$ approximates a normal curve as n increases.

One might also explore the probability distribution function of walkers in various other situations. First, the movement of a random walker with constant mass and a velocity to which random acceleration vectors are discretely or continuously applied. Second, the probability distribution of a walker within a container. Third, the probability that two walkers which begin a distance *d* apart meet. Fourth, the probability distribution of a walker with increasing or decreasing movement size. Lastly, the discrete or continuous probability function of a walker on a spherical or warped plane.

Appendices :

Appendix A : Code for Integration

integration.py

Output: Coefficients of the polynomial sections in ascending order, multiplied by 2^n(n-1)!

```
n = 1
[1]



n = 2

[2, 1]

[2, -1]



n = 3

[9, 6, 1]

[6, 0, -2]

[9, -6, 1]



n = 4

[64, 48, 12, 1.0]

[32, 0, -12, -3.0]

[32, 0, -12.0, 3.0]

[64, -48, 12, -1.0]



n = 5

[625, 500, 150, 20, 1]
```

```
[220, -40, -120, -40, -4]

[230, 0, -60, 0, 6]

[220, 40, -120, 40, -4]

[625, -500, 150, -20, 1]
```

Code:

```
import matplotlib.pyplot as plt
from itertools import zip_longest

class polynomial:
    '''
    A class used to represent a polynomial

    ...

    Attributes
    ----------
    coeffs : list
        a list containing the polynomial coefficients.
        The ith item is the coefficient of x**i
    '''

    def __init__(self, coeffs):
        self.coeffs = coeffs

    def __add__(self, other):
        '''adds coefficents in each polynomial'''
        return polynomial([sum(x) for x in zip_longest(self.coeffs, other.coeffs, fillvalue=0)])

    def __rmul__(self, factor:float):
        return polynomial([factor * x for x in self.coeffs])

    def __getitem__(self, i:int):
        return self.coeffs[i]

    def __setitem__(self, key:int, value):
        self.coeffs[key] = value

    def __str__(self):
        return(str(self.coeffs))

    def apply(self, x):
        output = 0
        for i in range(len(self.coeffs)):
            output += self.coeffs[i] * (x ** i)
        return output

    def integral(self):
        '''Returns polynomial integral of polynomial'''
        integr = polynomial([0])
        for i in range(len(self.coeffs)):
            integr.coeffs.append(self.coeffs[i] / (i + 1))
        return integr


def binom_expansion(n:int):
    """Binomial expansion in n^2 time"""
    coeffs = [0, 1, 0]
    for _ in range(n):
        # Creates next level of Pascal's triangle
        coeffs = [0] + [coeffs[i] + coeffs[i+ 1] for i in range(len(coeffs) - 1)] + [0]
    return coeffs[1:-1]


def factorial(x:int):
    if x == 0:
        return 1
    n = 1
    for i in range(1, x + 1):
        n *= i
    return n


def double_integration(f1:polynomial, f2:polynomial, flip_pt:int):
    integr_f1 = f1.integral()
    integr_f2 = f2.integral()
    const = integr_f1.apply(flip_pt) - integr_f2.apply(flip_pt)

    for i in range(len(integr_f1.coeffs)):  # Recalculates integr_f1(A) to = integr_f1(A-1)
        coeff = integr_f1[i]
        integr_f1[i] = 0
        for k in range(i + 1):
            sign = (-1) ** (i - k) # f1 integral uses (A-1), so odd powers of (-1) are * -1
            integr_f1[k] += coeff * binom_expansions[i][k] * sign

    for i in range(len(integr_f2.coeffs)): # Recalculates integr_f1(A) to = integr_f1(A-1)
        coeff = integr_f2[i]
        integr_f2[i] = 0
        for k in range(i + 1):
            integr_f2[k] += coeff * binom_expansions[i][k]
    integr_sum = -1 * integr_f1 + integr_f2
    integr_sum[0] += const
    return integr_sum


def iterate_set(poly_set:list, n:int):
    '''Iterates the polynomials for the probability function of step n + 1

    Arguments
    -----
    poly_set : list
        list of polynomials
    n : int
        Number of steps
    '''
    poly_set = [polynomial([0])] + poly_set + [polynomial([0])] # f(x) = 0 borders on either
side
    next_set = []
    for i in range(len(poly_set) - 1):
        next_set.append(double_integration(poly_set[i], poly_set[i + 1], 2 * i - n))
    return next_set


if __name__ == "__main__":
    n_max = 100

    binom_expansions = [binom_expansion(n) for n in range(n_max + 1)]
    poly_tree = [[polynomial([1])]]
    for n in range(1, n_max):
        poly_tree.append(iterate_set(poly_tree[-1], n))

    print("\n")
    for i in range(len(poly_tree)):
        for item in poly_tree[i]:
            print(factorial(i) * item)
        print("\n")
```

Appendix B: Code for Simulation

main.py

Code:

```
import matplotlib.pyplot as plt
from numpy import random, std, pi, cos, sin
from graph import *


def rand_walker_1d(sample_size:int, num_steps:int):
    walkers = [0 for _ in range(sample_size)]           # List of walkers with location
    std_devs = []
    means = []

    for _ in range(num_steps):                          # Changes position, num_steps times
        std_devs.append(std(walkers))
        means.append(sum(walkers)/len(walkers))
        for i in range(len(walkers)):
            walkers[i] += ((random.rand() ** 0.5 * 2) - 1)     # Adds random number from -1 to 1
    return walkers, std_devs, means
```

```
def rand_walker_2d(sample_size:int, num_steps:int):
    """Simulates a population of walkers that move up
    to 1 unit in a random direction num_steps times
    """
    walkers = [[0,0] for _ in range(sample_size)]       # List of walkers with location
    std_devs = []
    for _ in range(num_steps):
        std_devs.append(std([dist(walker[0],walker[1]) for walker in walkers]))
        for i in range(len(walkers)):
            radius = random.rand()                  # Movement magnitude, [0, 1]
            theta = random.rand() * 2 * pi            # Movement angle, [0, two pi]
            walkers[i][0] += radius * cos(theta)       # Adds movement to x
            walkers[i][1] += radius * sin(theta)       # Adds movement to y
    return walkers, std_devs


if __name__ == "__main__":
    sample_size = 1000000
    num_steps = 100

    walkers, std_devs, means = rand_walker_1d(sample_size, num_steps)
    # scatter_walker(walkers)
    # print(std_devs)
    # std_by_steps(std_devs, num_steps, sample_size)
    # pos_hist(walkers=walkers, num_steps=num_steps, num_bins=100, dimensions=1,
by_radius=True, density=1)
    mean_by_steps(means, num_steps, sample_size)
```

graph.py

Code:

```
import matplotlib.pyplot as plt
from numpy import random, std, pi, cos, sin

def pos_hist(walkers:list, num_steps:int, num_bins:int, dimensions:int, by_radius = True,
density=1):
    if dimensions == 1:
        n, bins, patches = plt.hist(walkers, num_bins, density=density, facecolor='g', alpha=0.75)
        plt.title("Walker locations after " + str(num_steps) + " steps")
        plt.xlabel("Position")
        plt.ylabel("Frequency")
        plt.show()
    elif dimensions == 2:
        if by_radius:
            walkers_dist = [dist(walker[0],walker[1]) for walker in walkers]

            n, bins, patches = plt.hist(walkers_dist, num_bins, density=density, facecolor='g',
alpha=0.75)
            plt.title("Walker locations after " + str(num_steps) + " steps")
            plt.xlabel("Distance from Origin")
            plt.ylabel("Frequency")
            plt.show()
        else: # Scales the bin size by the inverse of the circumference of the circle with the
radius in the middle of the bin
            walkers_dist = [dist(walker[0],walker[1]) for walker in walkers]
            n, bins, patches = plt.hist(walkers_dist, num_bins, density=density)
            for i in range(len(n)):
                bins[i] = (bins[i] + bins[i + 1]) / 2
                n[i] = n[i] / (pi * (bins[i]) ** 2)
            plt.plot(bins[:-1], n, "b.")
            plt.show()


def dist(x:float, y:float) -> float:
    return (x ** 2 + y ** 2) ** 0.5


def std_by_steps(std_devs:list, num_steps:int, sample_size:int):
    plt.plot(std_devs)
    plt.title("Standard Deviation by the Number of Steps")
    plt.xlabel("Number of Steps")
    plt.ylabel("Standard Deviation")
    plt.text(4/6 * num_steps, 0.5, "n = " + str(sample_size))
    plt.show()

def mean_by_steps(means:list, num_steps:int, sample_size:int):
    plt.plot(means)
    plt.title("Mean by the Number of Steps")
    plt.xlabel("Number of Steps")
    plt.ylabel("Mean")
    plt.show()

def scatter_walker(walkers):
    x = [walker[0] for walker in walkers]
    y = [walker[1] for walker in walkers]
    plt.plot(x, y, "b.")
    plt.show()
```