Random Walks

Elijan Jakob Mastnak
Student ID: 28181157

October 2020

Contents

Assignment

Create a computer simulation of two-dimensional random walks and random flights. Start at the origin ($x = y = 0$), and determine the next position by randomly choosing an independent direction ϕ and step size l, i.e.

$$x_{n+1} = x_n + l\cos\phi \qquad \text{and} \qquad y_{n+1} = y_n + l\sin\phi$$

where ϕ is uniformly distributed on the interval $[0, 2\pi)$ and l is distributed according to $p(l) \propto l^{-\mu}$. Plot a few examples of walks and flights for 10, 100, 1000, and 10000 steps. From a large number of walks/flights, each with a large number of steps, determine the value of the exponent γ for a few chosen values of the parameter μ and use this result the characterize each walk/flight by its diffusion regime.

---

1 Theory

To jump right to the solution, see Section 2.

Random walks are a form of motion in which, over a process of many individual steps, a "walker" (e.g. a point particle) progresses from the origin to some final position while the parameters of each step (e.g. size, direction) randomly change over the course of the walk.

1.1 Normal Diffusion

Diffusion—net movement of particles from high to low concentration—can be modeled by treating each particle as a random walker. In *normal diffusion*, a particle's final position averaged over a large number of random walks, denoted by $\langle r(t)\rangle$, tends to stay at the origin, while variance of the final distance from the origin over time, denoted by $\sigma_r^2(t)$, grows linearly with walk time as:

$$\sigma_r^2(t) \equiv \left\langle r^2(t)\right\rangle - \left\langle r(t)\right\rangle^2 = 2Dt$$

The proportionality constant D is the *diffusion constant*. As long as the mean time between steps and mean squared step size are finite, the central limit theorem states that the resulting distribution of final distances from the origin follows a normal distribution.

1.2 Anomalous Diffusion

We now consider random walks with a possibility of very large step sizes, and parameterize the step length distribution's probability density function with the power distribution

$$p(l) \propto l^{-\mu} \tag{1}$$

where $\mu \in (1, 3)$. In this case, the distribution's second moment

$$\left\langle l^2\right\rangle = \int_0^\infty l^2 p(l)\,\mathrm{d}l$$

diverges, and the central limit theorem does not apply. In this case, the variance $\sigma_r^2(t)$ behaves as

$$\sigma_r^2(t) \sim t^\gamma \tag{2}$$

where the value of the exponent γ depends on the value of μ in Equation 1. Diffusion with a non-linear relationship between σ_r^2 and t is called *anomalous diffusion*. We will consider two types of anomalous diffusion: Lévy flights and Lévy walks.

1.2.1 Lévy Flights

Lévy flights are a subset of anomalous diffusion in which each step takes the same amount of time. Because the time of each step is fixed and step size, determined by Equation 1, can vary wildly, so too can the speed at which steps are taken. The flight's total time is directly proportional to the total number of steps.

Summary: Fix step time, draw random step size, adjust step speed to match step time.

For Lévy flights, the theoretically expected relationship between γ and μ is

$$\gamma(\mu) = \begin{cases} \frac{2}{\mu - 1} & \mu \in (1, 3) \quad \text{(super-diffusion)} \\ 1 & \mu > 3 \quad \text{(normal diffusion)} \end{cases} \tag{3}$$

The special case $\mu = 2$ gives $\gamma = 2$, which is called *ballistic diffusion.*

1.2.2 Lévy Walks

Lévy walks are a subset of anomalous diffusion in which each step is taken at the same fixed speed. Because the step speed is fixed and step size, determined by Equation 1, can vary wildly, so too do the times each step takes—the time each step takes is proportional to the step size. The time of the entire walk is proportional to the total distance traveled.

Summary: Fix step speed, draw random step size, adjust step time to match step speed.

For Lévy walks, the expected relationship between γ and μ is

$$\gamma(\mu) = \begin{cases} 2 & \mu \in (1, 2) \quad \text{(ballistic diffusion)} \\ 4 - \mu & \mu \in (2, 3) \quad \text{(super-diffusion)} \\ 1 & \mu > 3 \quad \text{(normal diffusion)} \end{cases} \tag{4}$$

For the special cases $\mu = 2$ and $\mu = 3$ we expect

$$\mu = 2 \implies \sigma_r^2(t) \sim \frac{t^2}{\ln t} \qquad \text{and} \qquad \mu = 3 \implies \sigma_r^2(t) \sim t \ln t$$

2 Solution

2.1 Random Number Generation

As suggested in the instructions, I modeled a random step in two dimensions using polar coordinates: an angle $\phi \in [0, 2\pi]$ controlling direction and a length $l \in (\epsilon, \infty)$ controlling step size. Here ϵ denotes a small quantity; I personally used $\epsilon = 10^{-5}$, but the exact value is not critical and only serves to set the problem's distance scale.

2.1.1 Sampling Angle

I sampled the angle ϕ uniformly from the interval $[0, 2\pi]$ with a simple call to a pseudo-random number generation function, in my case the NumPy Python library's `random.uniform` function. Mathematically, the relevant equation is

$$\phi = \rho_\phi \cdot 2\pi \tag{5}$$

where ρ_ϕ is a (pseudo) random number uniformly distributed on the interval $[0, 1]$. In practice, I just used `np.random.uniform(0, 2*np.pi)`, which achieves the same result.

2.1.2 Sampling Step Size

I sampled step length l according to the power distribution in Equation 1 from the interval (ϵ, ∞) using the generating function

$$l(\rho_l) = \epsilon \cdot (1 - \rho_l)^{\frac{1}{1-\mu}}$$

where ρ_l is pseudo-random number uniformly distributed on $[0, 1]$.

Derivation: Start with the inverse transformation sampling-motivated equation

$$\int_{l_{\min}}^{l} p(t)\,\mathrm{d}t = \rho_l \cdot \int_{l_{\min}}^{l_{\max}} p(t)\,\mathrm{d}t \qquad \text{or} \qquad \int_{l_{\min}}^{l} t^{-\mu}\,\mathrm{d}t = \rho_l \cdot \int_{l_{\min}}^{l_{\max}} t^{-\mu}\,\mathrm{d}t$$

Solving the integrals, inserting the integration limits and solving for l leads to

$$\frac{t^{1-\mu}}{1-\mu}\bigg|_{t=l_{\min}}^{l} = \frac{\rho_l \cdot t^{1-\mu}}{1-\mu}\bigg|_{t=l_{\min}}^{l_{\max}} \implies l = \left[(1-\rho_l)\cdot l_{\min}^{1-\mu} + \rho_l \cdot l_{\max}^{1-\mu}\right]^{\frac{1}{1-\mu}}$$

As $l_{\min} \to \epsilon$ and $l_{\max} \to \infty$, the $l_{\max}^{1-\mu}$ term grows negligibly small compared to $l_{\min}^{1-\mu}$, leaving

$$l = \left[(1-\rho_l)\cdot \epsilon^{1-\mu}\right]^{\frac{1}{1-\mu}} = \epsilon \cdot (1-\rho_l)^{\frac{1}{1-\mu}} \tag{6}$$

See this Desmos link for an interactive example of $l(\rho)$, showing the results of changing the parameters $l_{\min}$ and μ.

A simple Python function for generating ϕ and l might read

```python
import numpy as np
l_min, mu = 1e-5, 2.      # set minimum step size and mu
np.random.seed(42)        # seed the random number generator

def phi_l_sampling_example():
    phi = np.random.uniform(0, 2*np.pi)    # samples phi from (0, 2pi)
    rho = np.random.uniform(0, 1)          # random float on (0, 1)
    l = l_min * (1 - rho) ** (1/(1 - mu))  # generates power-distributed step size
    return phi, l
```

Important: For a proper random walk, ϕ and l must be *independent*, meaning the values of the random number ρ used to generate them must be drawn separately. I tried to highlight this with the notation ρ_ϕ and ρ_l (instead of using just ρ for both ϕ and l).

2.1.3 Random Step Algorithm

To generate a single random step, I sampled a random ϕ and l according to Equations 5 and 6 and increment the walker's x and y positions according to

$$x_{n+1} \to x_n + l\cos\phi \qquad \text{and} \qquad y_{n+1} \to y_n + l\sin\phi$$

Note that a walker's next position depends only the previous position but is otherwise memory-less. For bookkeeping purposes, I also increment walk time either by the fixed step time t_0 (for Lévy flights) or by l/v_0 (step length divided by fixed step speed; for Lévy walks) and increment step number n.

A simple code implementation—adaptable to either flights or walks—might read:

```python
t0, v0 = 1, 1   # step parameters for Levy flights and walks, respectively
def get_next_step_example(x, y, t, n): # input x, y, time and step number
    phi, l = phi_l_sampling_example()  # generate random phi and l
    x_step = l * np.cos(phi)           # calculate increments to x and y
    y_step = l * np.sin(phi)
    x = x + x_step                     # update x and y positions
    y = y + y_step
    if(flight): t += t0                # increment time by fixed step time
    if(walk): t += l / v0              # increment time by using fixed speed
    n += 1                             # increment step number
    return x, y, t, n
```

2.2 Simulating an Entire Flight and Walk

For the record, my walk parameters are $l_{\min} = 10^{-5}$, $v_0 = 1$ for walks and $t_0 = 1$ for flights, chosen for a one-to-one correspondence between elapsed flight time and step number.

Lévy Flight

Simulating a Lévy flight is simple: decide on the number of steps in the walk and generate that many steps using the random step algorithm above. For example:

```python
def get_levy_flight(n_steps):
    x, y, t, n = 0, 0, 0, 0  # initialize position, time and step number
    x_list, y_list, r_list, t_list = np.zeros(n_steps), np.zeros(n_steps),
    ↪  np.zeros(n_steps), np.zeros(n_steps)  # arrays to hold flight history
    for i in range(0, n_steps):
        x, y, t, n = get_next_flight_step(x, y, t, n)  # generate steps
        x_list[i] = x
        y_list[i] = y
        r_list[i] = np.sqrt(x ** 2 + y ** 2)
        t_list[i] = t
    return x_list, y_list, r_list, t_list
```

My (Perhaps Ill-Fated) Approach to Lévy Walks

I tried experimenting with a different approach here. Namely: setting a fixed total walk time rather than a fixed number of steps. Here is my thought process: as far as I understand, the eventual goal is to find the variance in r as a function of *time*, not as a function of step number. If I simulate walks for a fixed number of steps, the total time of each walk will vary, which complicates calculating variance across all walks at a given time. Simulating walks for a fixed time instead of a fixed number of steps simplifies this, explained further in Subsubsection 2.4.2. However—skipping ahead somewhat—this approach to Lévy walks may be responsible for the deviation of the fitted parameter γ from the theoretically expected value in the super-diffusive regime, shown in Figure 2.

Following is an example Python function for generating a Lévy walk:

```python
def get_levy_walk(walk_time):
    x, y, t, n = 0, 0, 0, 0  # initialize position, time and step number
    x_list, y_list, r_list, t_list = [], [], [], []  # to hold flight history
    while t < walk_time:  # simulate for given time, not given step number
        x, y, t, n = get_next_walk_step(x, y, t, n)  # generate steps
        x_list.append(x)
        y_list.append(y)
```

```
        r_list.append(np.sqrt(x ** 2 + y ** 2))
        t_list.append(t)
    return x_list, y_list, r_list, t_list, n
```

2.3 Plots of Random Walk and Flight

To avoid cluttering the main report with too many figures—LaTeX is somewhat finicky with float placement—I moved the plots of 2D random walks and flights to Appendix A, where you will find plots of both Lévy flights and walks in the ballistic, super-diffusive and normal regimes for 10, 100, 1000 and 10000 steps. Appendix B shows analogous plots in three dimensions.

2.4 MAD Calculation

This section quantifies the spread in the walkers' distance from the origin over time. Although mean squared displacement σ_r^2 is standard in the literature, I used median absolute deviation (MAD), which proves better suited to the large outlying values of r.

2.4.1 MAD Calculation for Lévy Flights

Here is the procedure I used to calculate MAD for Lévy flights:

- Simulate N_{flight} flights, each with N_{step} steps. For each flight, call the flight simulation function in Section 2.2, which returns distance from origin r at each step. Because steps correspond one-to-one with time, this data is equivalent to distance from the origin at each time t. I used $N_{\text{flight}} = 10^4$ and $N_{\text{step}} = 10^3$.
- Create a distances matrix $\mathbf{R}$ with dimensions $(N_{\text{steps}} \times N_{\text{walks}})$ whose columns are the $r(t)$ lists for each walk.
- Calculate MAD of the $\mathbf{R}$ matrix across rows using `stats.median_abs_deviation(R, axis=1)`. This gives the flight's MAD as a function of time. In Python:

```
def flight_simulation_example(n_walks, n_steps):
    r_matrix = np.zeros((n_steps, n_walks))  # matrix of distances from origin
    for i in range(0, n_walks):
        _, _, r, _, _ = get_levy_flight(n_steps)  # list of x, y, r, time and n for
        ↪  each walk. Only r is needed here
        r_matrix[:, i] = r  # sets the walk r as the r_matrix's ith column
    r_mad = stats.median_abs_deviation(r_matrix, axis=1)  # takes MAD across rows
    t = np.linspace(1, t0*n_steps, n_steps)  # take advantage of fixed step time
    return t, r_mad
```

2.4.2 MAD Calculation for Lévy Walks

Here is the somewhat involved procedure I used to calculate MAD for Lévy walks:

- Simulate N_{walk} walks each of fixed duration t_{walk}. For each walk, call the flight simulation function in Section 2.2, which returns distance from origin r versus time.

  As for Lévy flights, I used $N_{\text{walk}} = 10^4$. Because of the varying step number in each walk, I could not fix a single value for N_{step}, but instead chose t_{walk} so that the median number of steps per walk was about 10^3, as for Lévy flights.

- Calculate median number of steps N_{med} per walk (I used median instead of mean to mitigate the effect of outlying step numbers).

  Create a distance matrix $\mathbf{R}$ with N_{med} rows and N_{walk} columns and a time list vector $\boldsymbol{t}_{\text{med}} \in \mathbb{R}^{N_{\text{med}}}$ with N_{med} entries spaced uniformly from 0 to t_{walk}.

- For each walk, loop through each time in $\boldsymbol{t}_{\text{med}}$, and assign the walker's position r at that time to the corresponding entry in $\mathbf{R}$. This step takes advantage of the fact that $\boldsymbol{t}_{\text{med}}$ and each simulated walk have the same time duration (even though the number of steps vary) and both grow monotonically from the initial to the final time.

  The result is a distance matrix $\mathbf{R}$ with a uniform number of entries per column for each walk characterizing the walkers' distance from the origin over time.

- At this point, proceed as for flights: calculate MAD of the $\mathbf{R}$ matrix across rows to find the walk's MAD as a function of time.

I'm leaving out the code—which is either tedious array index manipulation or identical to the flight case—since I don't think it adds any extra value to the above explanation.

2.5 Determining Gamma with Linear Regression

The parameter γ and the $\text{MAD}_r(t)$ data are connected by the proportionality chain

$$\text{MAD}_r^2(t) \propto \sigma_r^2(t) \propto t^\gamma \implies \text{MAD}_r^2(t) = \alpha t^\gamma, \quad \alpha \in \mathbb{R}$$

This power relationship begs for log linearization! Taking the logarithm of both sides gives

$$2\ln\big[\text{MAD}_r(t)\big] = \gamma \ln(t) + \ln(\alpha),$$

meaning the power γ is the slope of the best-fit line passing through the quantity $2\ln\big[\text{MAD}_r(t)\big]$ plotted on the ordinate axis versus $\ln(t)$ plotted on the abscissa. To find γ, I calculated the $\text{MAD}_r(t)$ for various values of the parameter μ as described in Subsection 2.4 for both flights and walks, linearized the data as described above, and used the `SciPy` library's `curve_fit` function to perform the linear regression. `curve_fit` also returns a covariance matrix whose diagonal entries give an estimate of the error on the fit parameters. Subsection 2.6 explains how I estimated the error of each $2\ln\big[\text{MAD}_r(t)\big]$ data point.

Figures 1 and 2 show the fitted values of γ, including an estimate of error, for three regimes of Lévy flights and walks, respectively.

Discussion of Results

- I believe the simulation of Lévy flights and subsequent "extraction" of γ is satisfactory. The fitted values of γ closely agree with the theoretically expected values—to within less than one percent for $\mu = 1.5$ and 2.0 and a somewhat larger four percent error for $\mu = 3.5$ in the normal diffusion regime—and agree with the theoretical values within the realm of the regression error.

- Meanwhile, the Lévy walks are somewhat lacking. On the one hand, in the ballistic and normal regimes ($\mu = 1.5$ and 3.5) the fitted values of γ more or less agree with the theoretically expected values within the range of regression error. On the other hand, (even though the discrepancy is not drastic) the fitted value of γ differs appreciably from the theoretically predicted value in the super-diffusive $\mu = 2.5$ case. I imagine the problem lies in my approach to handling the varying-step nature of Lévy walks when calculating $\text{MAD}_r(t)$, discussed in Subsubsection 2.4.2.

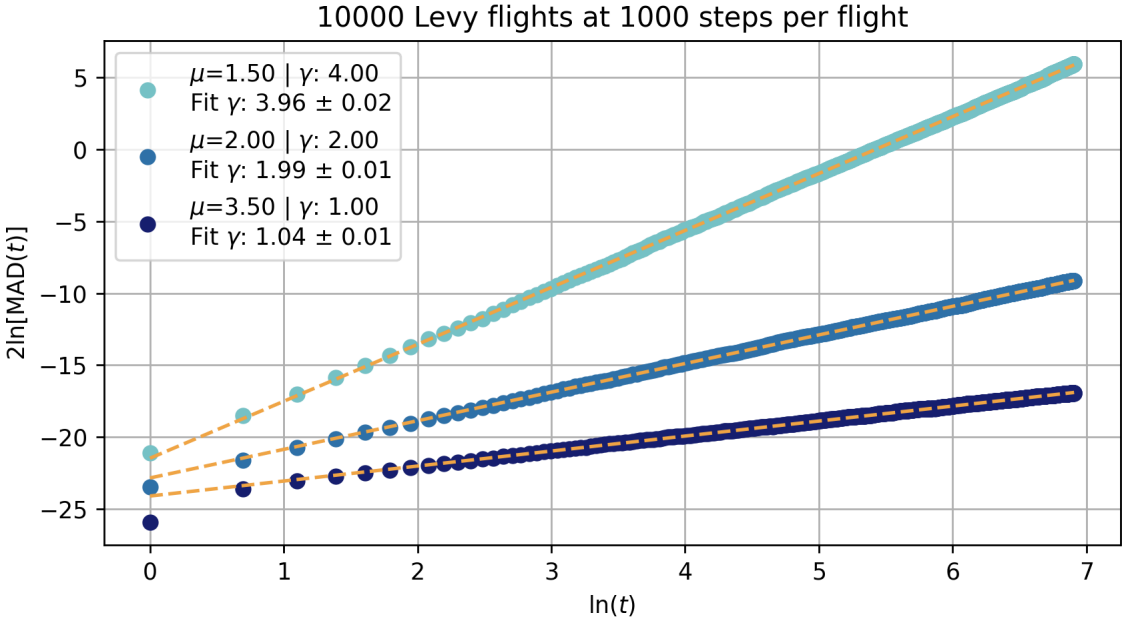


Figure 1: Finding the parameter γ for three regimes of Lévy flights using linear regression. The legend compares the theoretically expected and fitted value for each μ. In order of increasing μ, the flights fall in the super-diffusive, ballistic, and normal regimes. Error is estimated using the regression's corresponding covariance matrix.

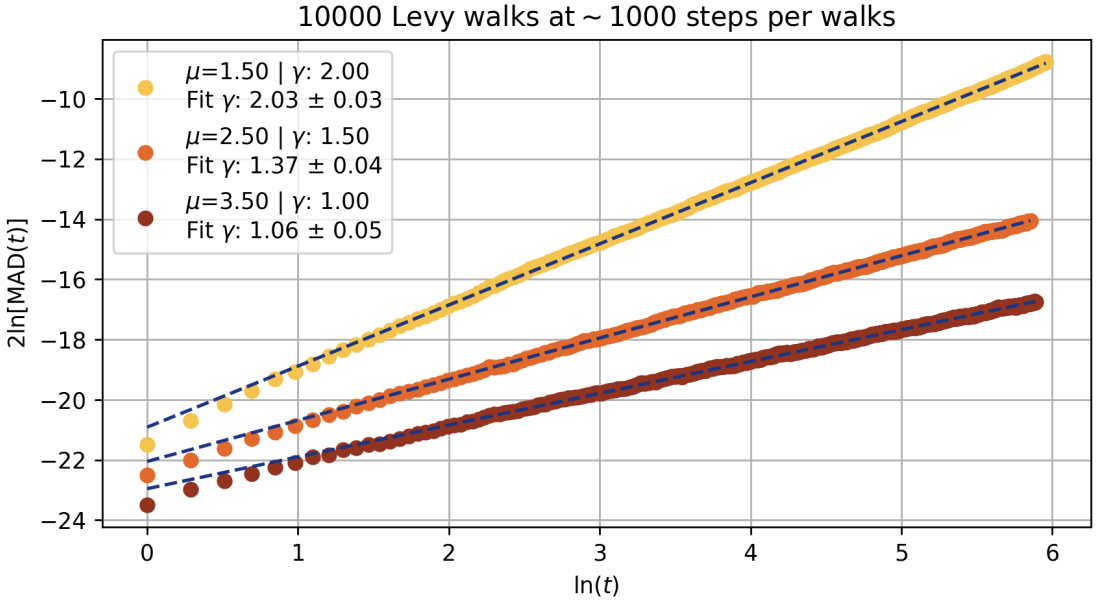


Figure 2: Finding the parameterγ for three regimes of Lévy walks using linear regression. The legend compares the theoretically expected and fitted value for each μ. In order of increasing μ, the walks fall in the ballistic, super-diffusive, and normal regimes. Error is estimated using the regression's corresponding covariance matrix. Note that time values have been shifted so all three graphs are next to each other, which gives a better visual presentation without changing the lines' slopes.

2.6 Estimating Error of Individual Data Points

This section explains how I estimated the error on each $2\ln\left[\mathrm{MAD}_r(t)\right]$ data point when performing linear regression. Note that there's probably a better way to do this—the approach feels a bit "hacky", but is at least an honest attempt if nothing else.
In essence, I assumed the quantity $2\ln[\mathrm{MAD}_r(t)]$ behaved as a random variable distributed about a central value, so that its standard deviation provided an estimate of its error. To realize this idea:

- For each value of μ, I ran 50 sets of 100 walks at either 1000 steps per walk (for Lévy flights) or at a fixed time corresponding to about 1000 median steps per walk (for Lévy walks). Of course, I changed the random seed at each run. I calculated the $\mathrm{MAD}_r(t)$ values of each run as described in Subsections 2.4.1 and 2.4.2.

- I found $2\ln[\mathrm{MAD}_r(t)]$ from $\mathrm{MAD}_r(t)$ and then calculated the sample standard deviation of the 50 resulting $2\ln[\mathrm{MAD}_r(t)]$ values at each point in time. The choice of 50 runs is somewhat arbitrary; I just needed a large enough number for the concept of standard deviation to become meaningful.

- I used the sample standard deviation s as an estimate of the standard error σ at each time t. These values gave an estimate of the error of $2\ln[\mathrm{MAD}_r(t)]$ as a function of time. In this way each point is assigned an individual error, rather than just assuming a blanket error for all points.

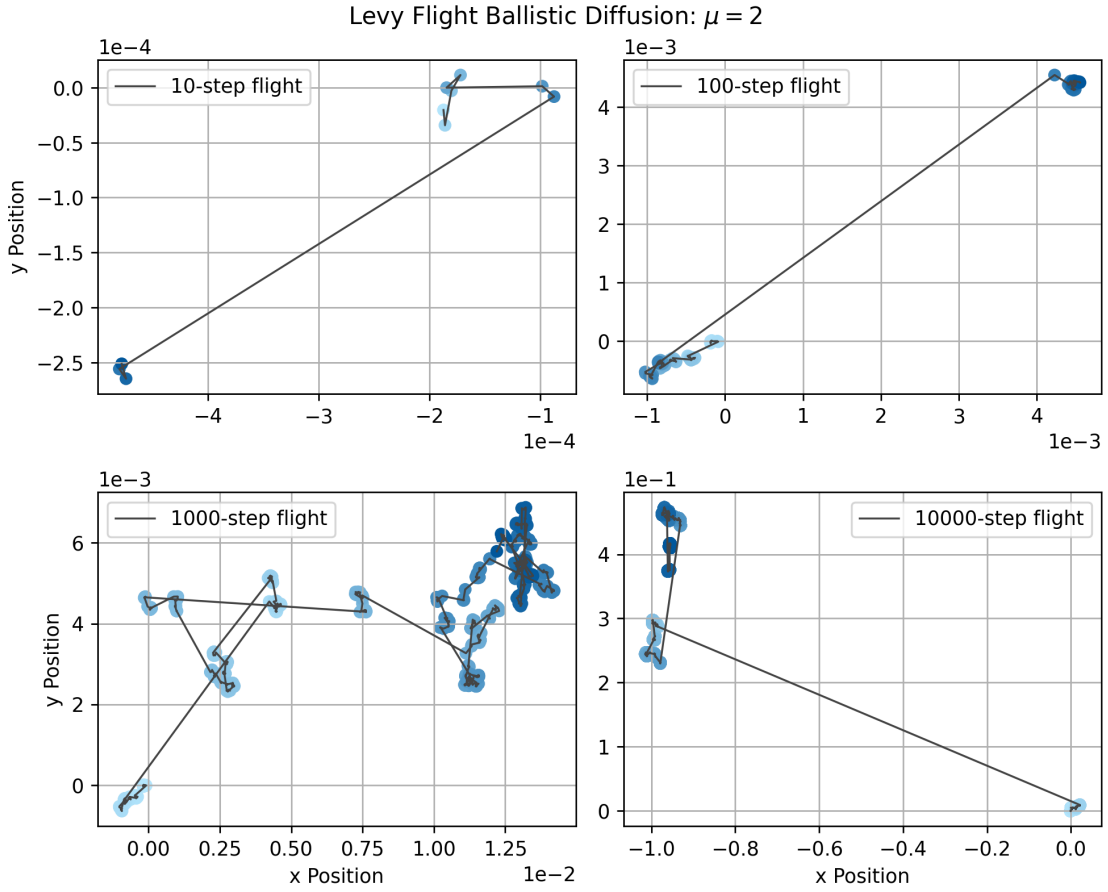
Given the large number of runs, I simulated only 100 walks per set when estimating error (instead of 10000 when estimating γ) because of finite computing power. I then assumed error scaled by a factor of $\sqrt{\frac{100}{10000}}$ in accordance with square root statistics when scaling up from 100 to 10000 walks per simulation, since more walks corresponds to more samples.

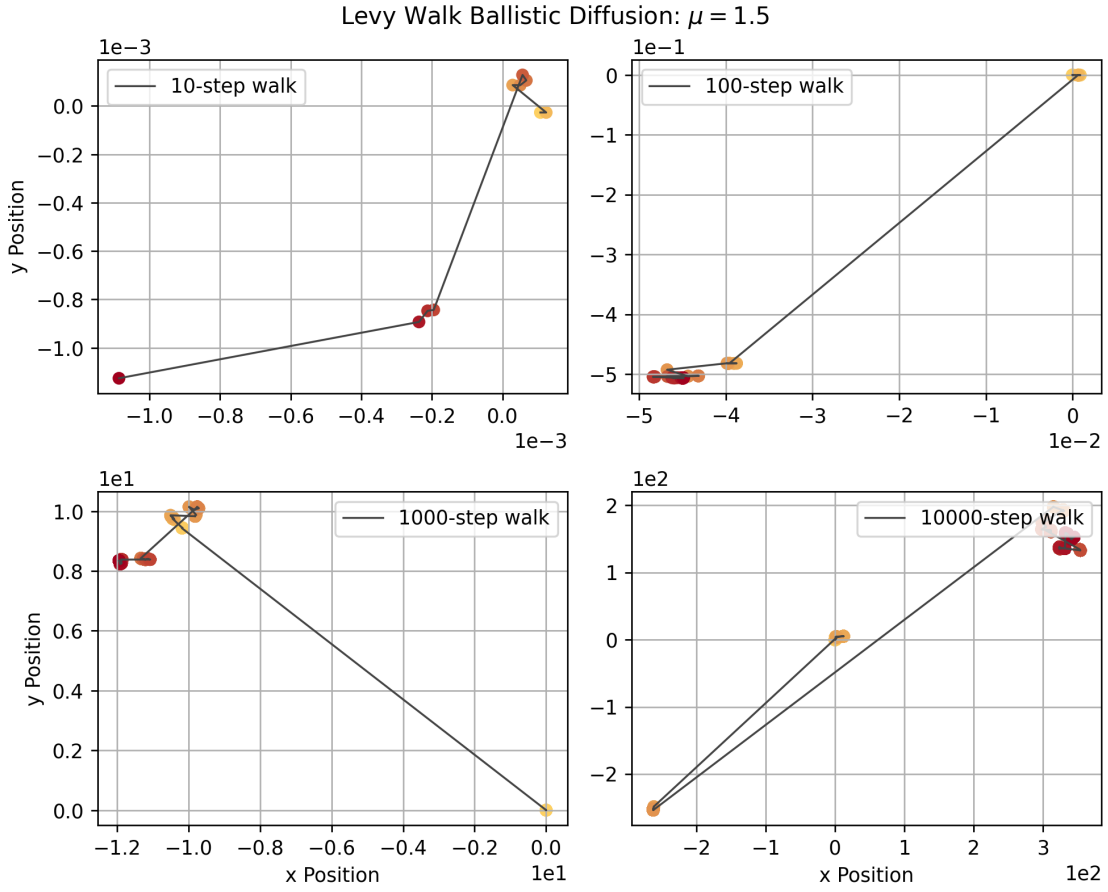
A Plots of Random Walks in 2D

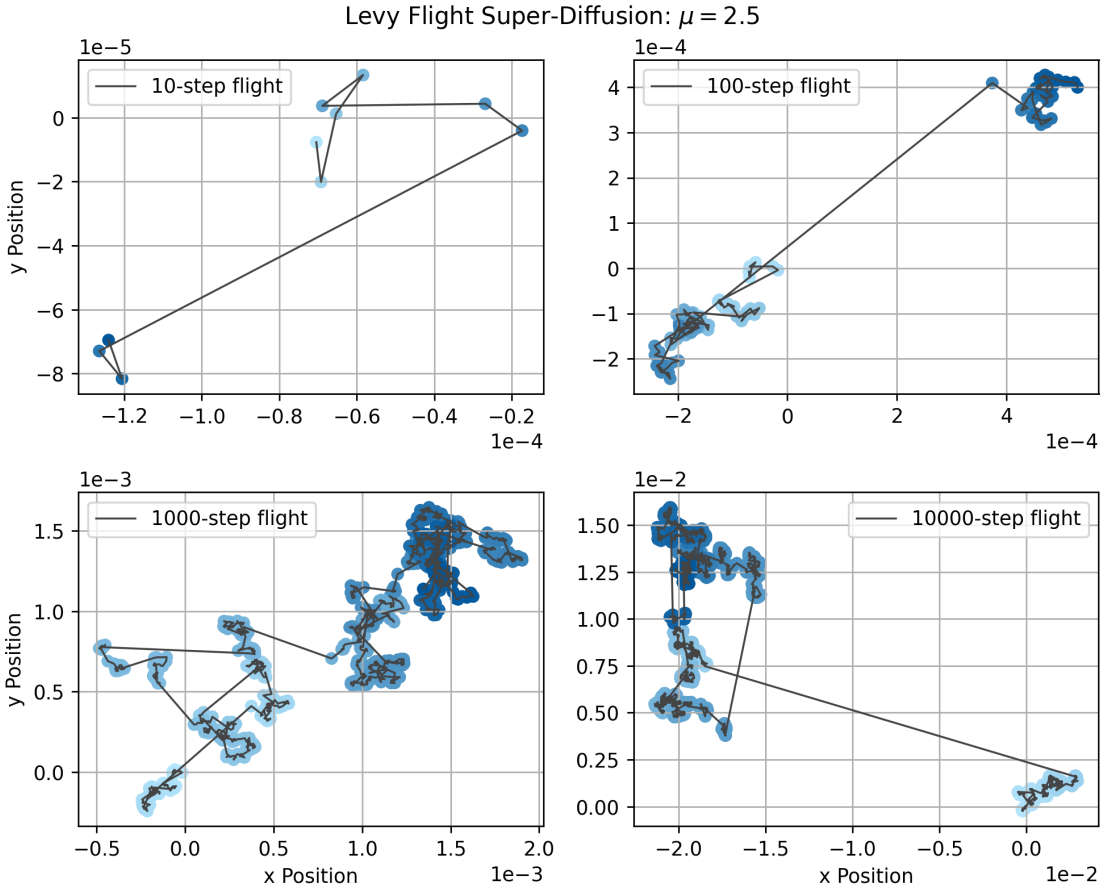
This section includes plots of Lévy flights and walks in the ballistic, super-diffusive and normal regimes for 10, 100, 1000 and 10000 steps. The progression in color gradient from light to dark corresponds to increasing step number and is intended to help visualize the walk's time evolution.
Two fairly obvious observations:

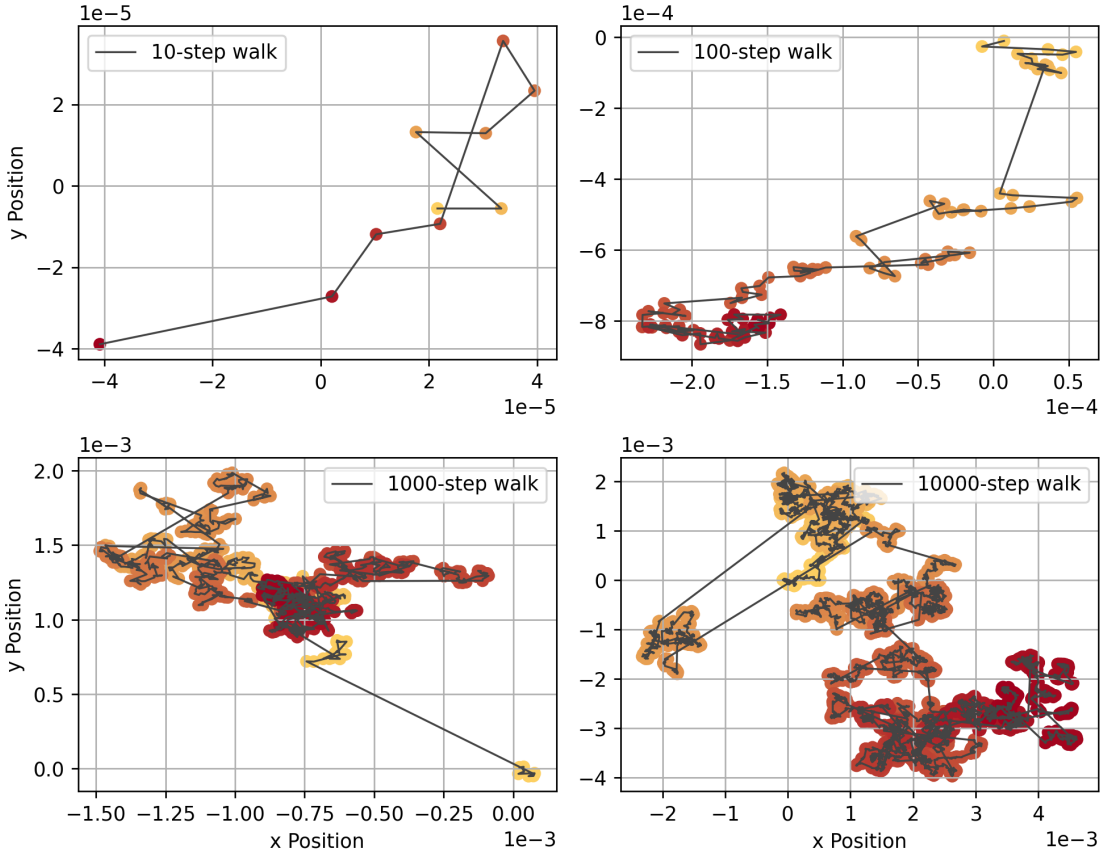
1. The walks in the anomalous diffusion regime are dominated by the large steps, while the walks in the normal diffusion regime are more "balanced".

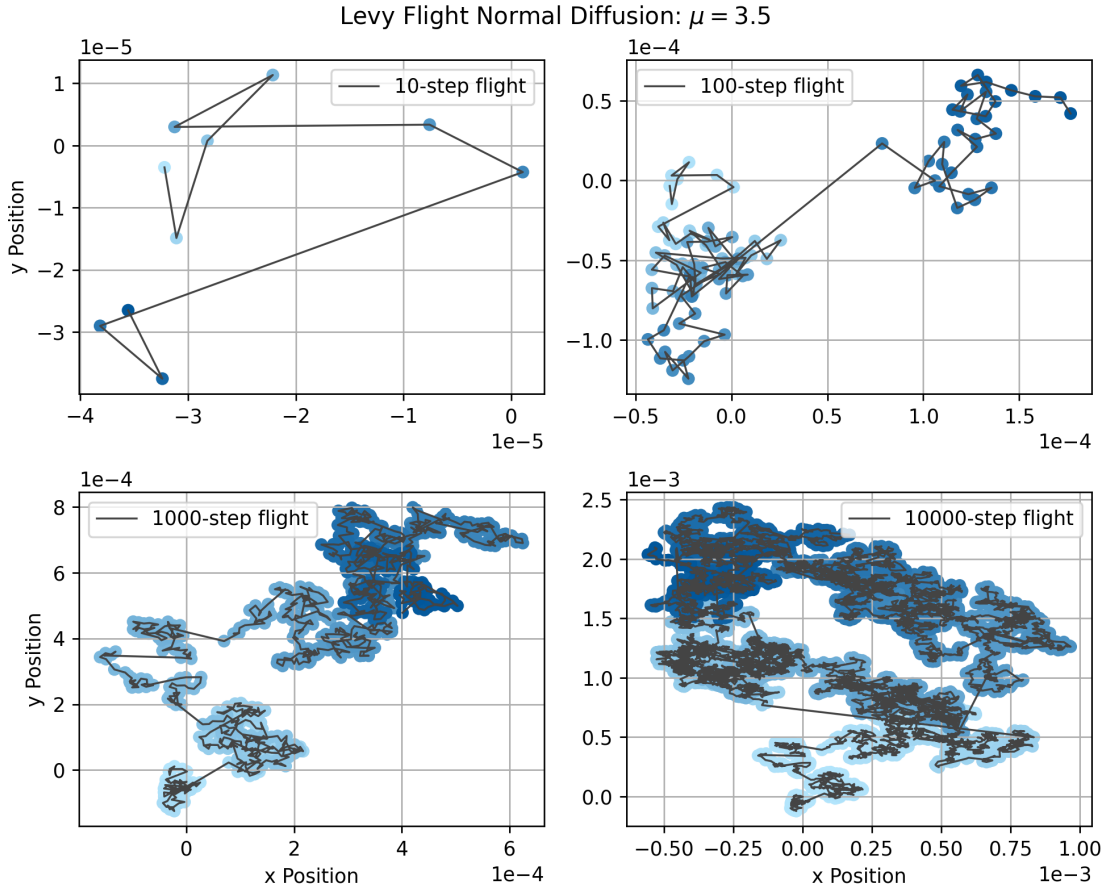
2. The plots exhibit a fractal-like nature as the number of steps increases. This trend is particularly visible in the anomalous diffusion cases.

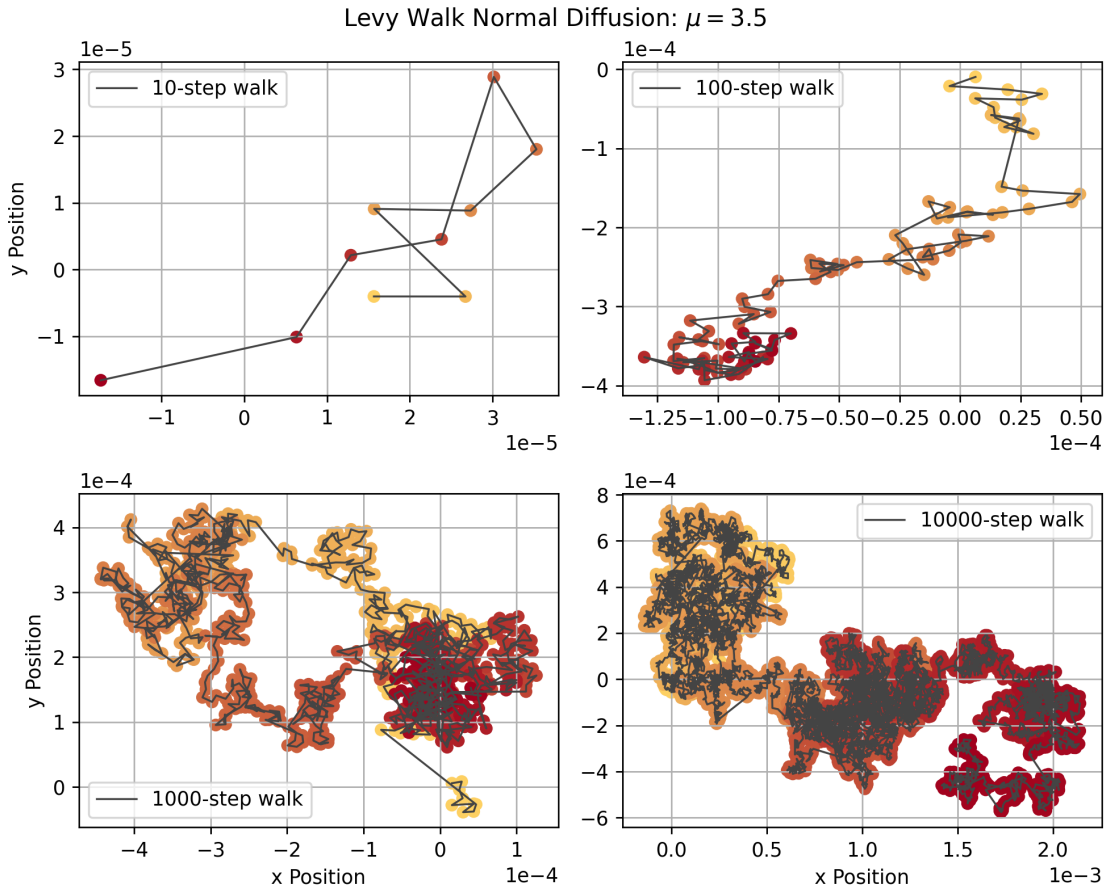

Levy Flight Ballistic Diffusion: $\mu = 2$
10-step flight
100-step flight
1000-step flight
10000-step flight
y Position
x Position
1e−4
1e−3
1e−2
1e−1


Levy Walk Ballistic Diffusion: $\mu = 1.5$
10-step walk
100-step walk
1000-step walk
10000-step walk
y Position
x Position
1e−3
1e−1
1e−2
1e1
1e2


Levy Flight Super-Diffusion: $\mu = 2.5$
10-step flight
100-step flight
1000-step flight
10000-step flight
y Position
x Position
1e−5
1e−4
1e−3
1e−2

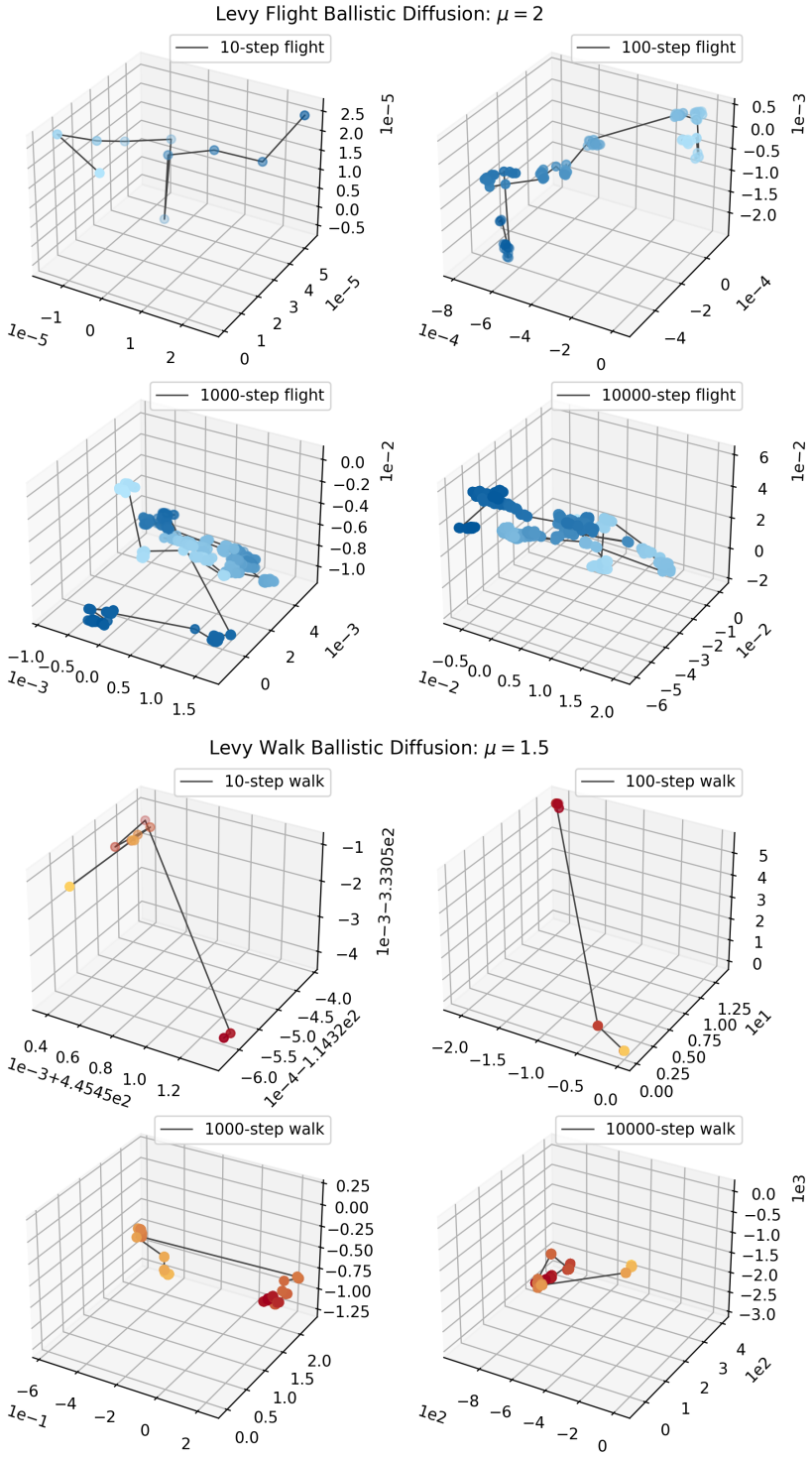

Levy Walk Super-Diffusion: $\mu = 2.5$
10-step walk
100-step walk
1000-step walk
10000-step walk
y Position
x Position
1e−5
1e−4
1e−3

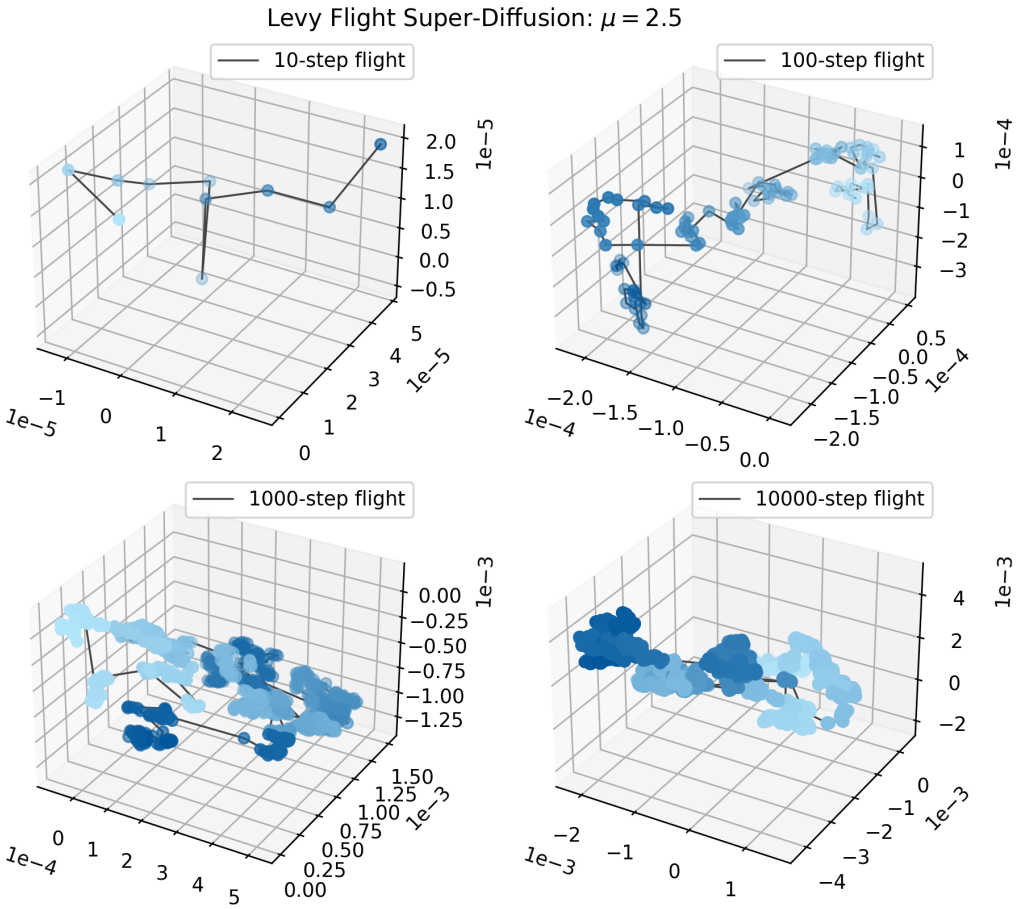

Levy Flight Normal Diffusion: $\mu = 3.5$
10-step flight
100-step flight
1000-step flight
10000-step flight
y Position
x Position
1e−5
1e−4
1e−3

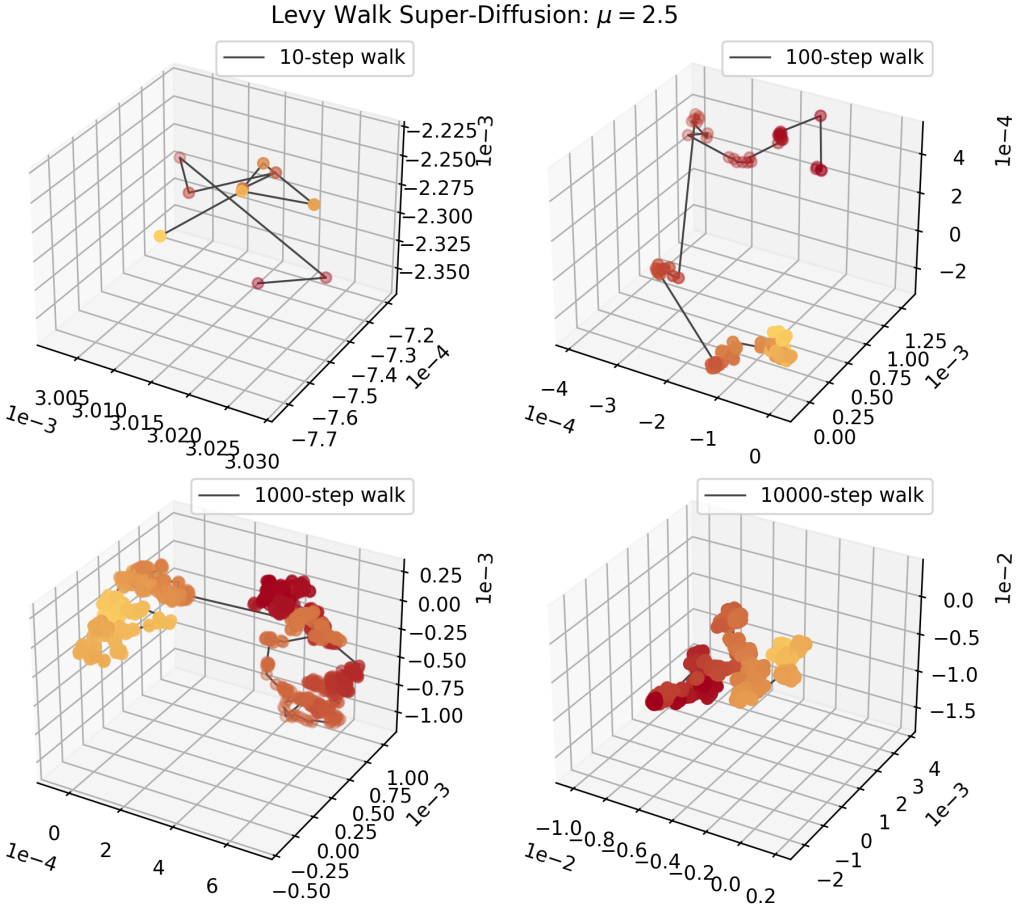

Levy Walk Normal Diffusion: $\mu = 3.5$
10-step walk
100-step walk
1000-step walk
10000-step walk
y Position
x Position
1e−5
1e−4
1e−3

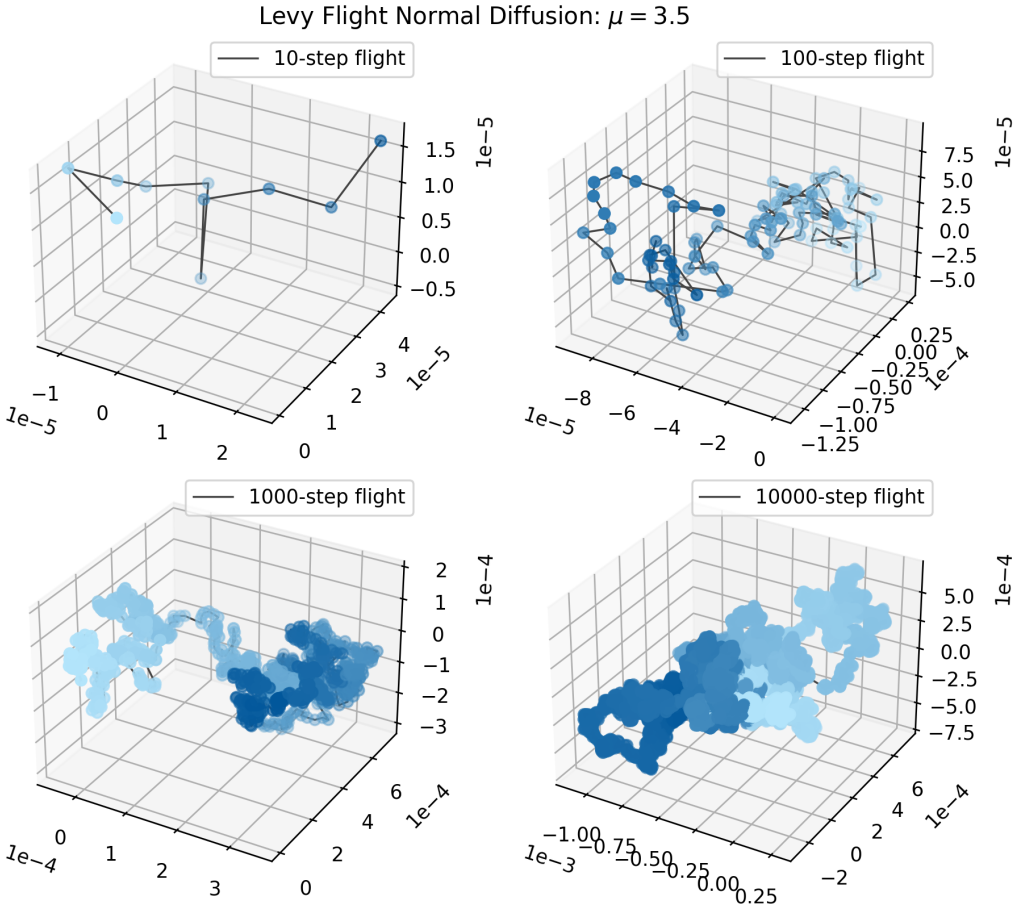
B Plots of Random Walks in 3D

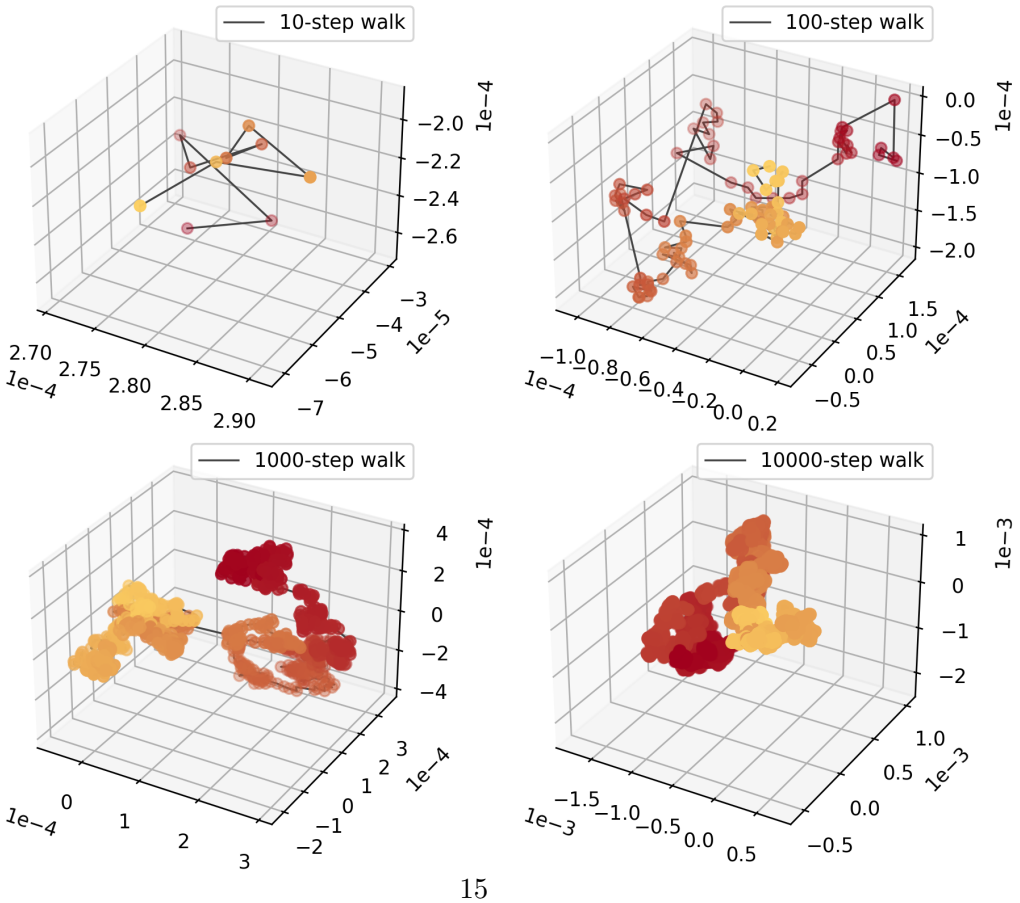
This section is included just for fun—basically an excuse to play around with `matplotlib`'s 3D plotting capabilities. I generated the random steps as described in e.g. Subsubsection 2.1.3, but scaled to three-dimensional spherical coordinates by including a polar angle θ uniformly distributed on $[0, \pi]$.




Levy Flight Super-Diffusion: $\mu = 2.5$
10-step flight
100-step flight
1000-step flight
10000-step flight


Levy Walk Super-Diffusion: $\mu = 2.5$
10-step walk
100-step walk
1000-step walk
10000-step walk


Levy Flight Normal Diffusion: $\mu = 3.5$
10-step flight
100-step flight
1000-step flight
10000-step flight


Levy Walk Normal Diffusion: $\mu = 3.5$
10-step walk
100-step walk
1000-step walk
10000-step walk