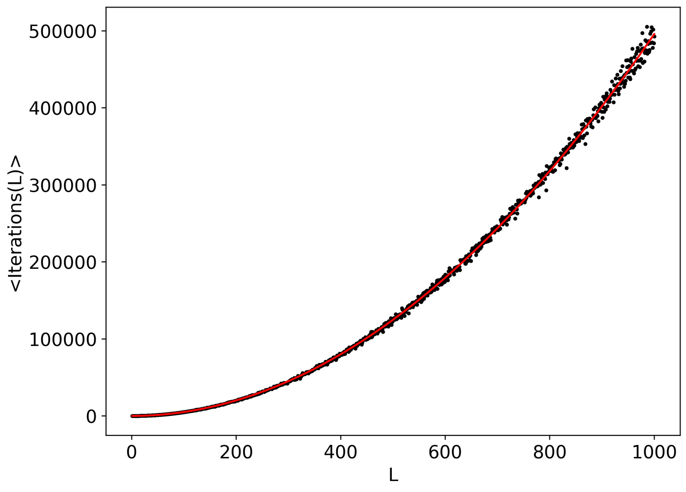

Fig. 1. Average number of iterations in 1D as a function of L.

1. Programming a D-dimensional random walk

The general approach was to use D indices stored in an array to track the position of the particle within each dimension. By first generating a random number to pick the dimension and then a second to move forwards or backwards I was able to simulate a multi-dimensional random walk. To track the position, each coordinate was converted into a unique index J using,

$$J = \sum_{j=0}^{D-1} x_j L^j, \tag{1}$$

where x_j is the index in a given dimension. The benefit of flattening the array, as opposed to tracking the position in a D-dimensional array, is that it's much easier to scale to higher dimensions while maintaining manageable dynamic memory allocation. Information regarding the particles position within the grid is stored inside a structure. Memory is then dynamically allocated for the position and tracker arrays, that get resized as either L or D changes. At each step of the random walk, the current coordinate is converted into J and the tracker array is checked at that index. If its value is 0 then it, along with a separate *count* variable, get incremented by 1. This process continues until the value of *count* is equal to the total number, $N = L^D$, of positions within the grid, signifying all positions have been visited.

2. 1D random walk

2.1. Seeding random()

random() will use a linear congruential algorithm,

$$X_{n+1} = (aX_n + c) \ mod \ m, \tag{2}$$

or a similar formula, so the same seed X_0 will naturally produce the same set of pseudo-random numbers. In later code I have chosen to use the current time as the seed.

2.2. Average iterations for $L = 10$

The average number of hops converges to 46.00 after 10^7 iterations

2.3. Average iterations with varying L

Figure 1 has been calculated using 10^5 iterations for the first 100 values of L and 10^3 for the remaining 900. From this we find the power law relationship describing the average number of iterations,

$$\bar{I} = (0.53 \pm 0.011)\, L^{(1.99\ \pm\ 0.0030)}, \tag{3}$$

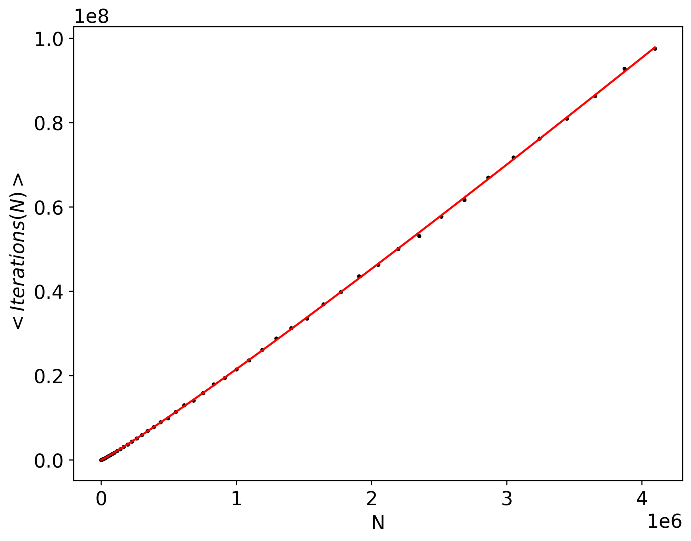

Fig. 2. $\bar{I} = (7.86 \pm 0.26)\, N^{(1.073 \pm 0.002)}$ for $0 < L < 160$, each at 100 iterations, in 3D.

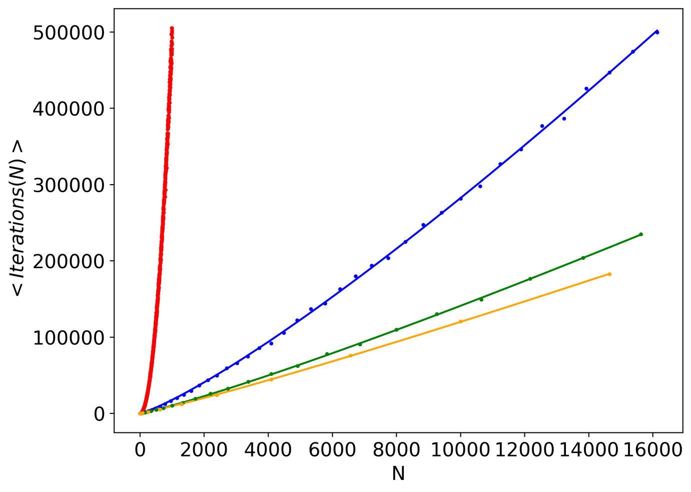

Fig. 3. Decreased scaling of $\bar{I}$ with N in 1D (red), 2D (blue), 3D (green), 4D (yellow).

where the error derives from SciPy curve-fit. To examine the accuracy of this equation, i simulated $L = 2000$ for 10^4 iterations, which returns $\bar{I} = 2.01 \times 10^6$, very similar to $\bar{I} = 1.97 \times 10^6$ obtained from this formula, suggesting suitable extrapolation. From the central limit theorem, given a sufficiently large number of iterations, one expects a normal distribution to appropriately describe the probability that the random walk is a certain distance from the start. This suggests that as L increases the probability of the particle ending up at the furthest position falls off in a nonlinear fashion. Therefore, a power law relationship appropriately describes the average number of iterations, $\bar{I}(L)$, to visit all positions.

3. Higher dimensional random walk

As $N = L^D$, in higher dimensions we no longer have a 1:1 mapping between N and L. Instead for the same N the grid is far more connected. A particles position is described by D indices, therefore one can imagine these as D independent particles each confined to a given dimension. In a random walk these 'particles' will move on average every D goes, each with a normal probability distribution about the origin. For a given N, the number of positions in 1D within the grid is $L = N^{1/D}$; thus, each particle is far more likely to explore the extremes of the space. Further, as D increases, the number of directions available to the particle also grows as $2 \times D$, meaning the particle has a lower probability of revisiting sites, and hence for a given N a fewer number of iterations to explore the available space.

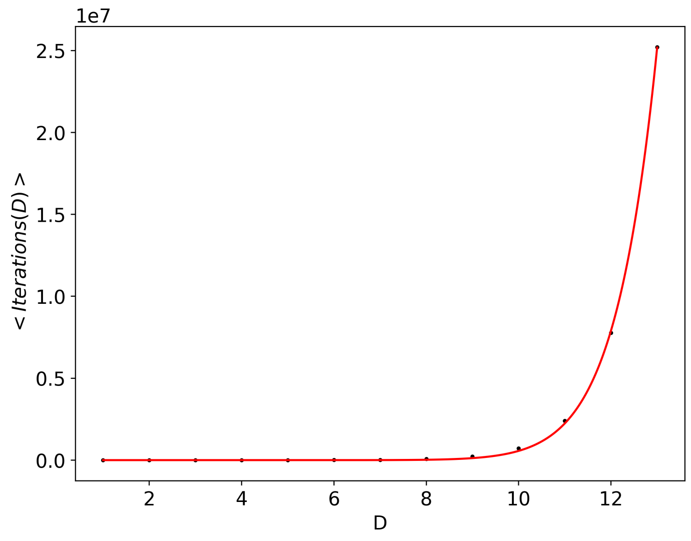

Fig. 4. $\bar{I}(D)$ for the first 13 dimensions, at $L = 3$ and 100 iterations.

Figure 2 shows the increased linearity of $\bar{I}(N)$ in 3D due to these effects, and Figure 3 shows $\bar{I}(N)$ becoming flatter as D varies from 1 to 4. Increasing D further, while holding $L = 3$ fixed, naturally reveals the relationship of $\bar{I}(D)$ depicted in Figure 4. This power-law increase follows from $N = L^D$.

4. Random walks on a network

4.1. Computational approach

Much of the same code can be reused here with the main changes being to the indexing and random number generation. We can label each component, i, from 1 to A, and nodes, j, within each component from 1 (the node connected to $i - 1$) to B (the node connected to $i + 1$). If $j \neq 1, B$ then the particle can move to any other maximumly connected node j within i, with probability $1/(B - 1)$. Thus, we can use a simple acceptation-rejection algorithm to generate a number between 1 and B until we get a number that is not the current position. For a reasonable size of B this has a favourable rejection probability of only $1/B$. If instead $j = 1, B$ then the particle may also move components. We can do almost the same as above but generate a random number x starting at 0. If $x > 0$ then we move to node $j = x$ within i, but if $x = 0$ then we move to the next component and j changes to 1, if B originally, or vice versa.

4.2. Limitations

If A or $B = 1$, then the simulation results are inaccurate, and while these could be solved with an *if* statement to catch these specific cases, I have chosen not to as these conditions are unnecessary for the reasons outlined below, and would clutter the code. If $A = 1$, the periodic boundary condition creates two edges between nodes 1 and B. However, this particular case can be solved analytically as $\bar{I}$ is equivalent to the expected number of outcomes needed to generate every random number between 1 and B, which can be solved easily using

$$\bar{I} = E(B) = B \times H_n, \tag{4}$$

where H_n is the sum of the reciprocals of the first B integers. If instead B equals 1, a 1D random walk is returned. These limitations are greatly outweighed by the memory and speed advantages of not having a matrix store which nodes are connected to one another.

4.3. Convergence to a 1D random walk

Setting $B = 2$ recovers an even 1D random walk, and with $A = 5$ we find $\bar{I} = 46.00$, the same as for $L = 10$.

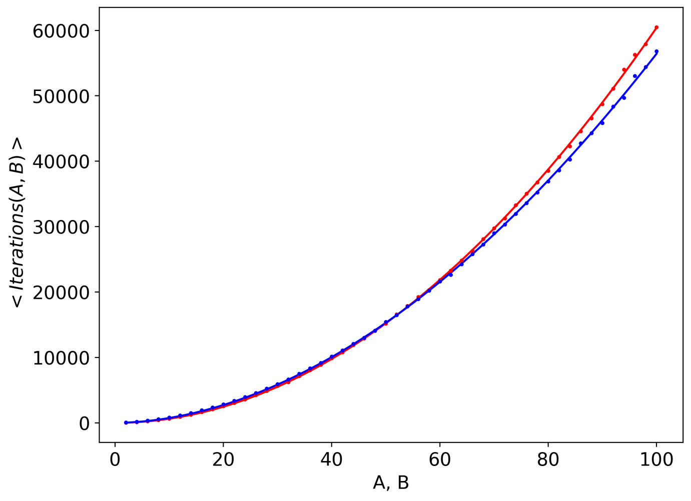

Fig. 5. $\bar{I}(A)$ (red) at fixed $B = 4$, and $\bar{I}(B)$ (blue) at fixed $A = 5$, both averaged over 10^5 iterations at each point.

4.4. Fixed and dynamic A, B

For fixed $A = 5$, $B = 4$ we find $\bar{I} = 167.54$, averaged for 10^7 iterations. Figure 5 shows A and B varying between 1 and 100 while the other variable is held fixed. Interestingly $\bar{I}(A) \propto A^{(1.99 \pm 0.004)}$ scales the same as for the 1D walk ($\propto L^{1.99}$). This remains true for increasing values of B too, but with an enlarging proportionality constant. This suggests that the network random walk is much like that of the 1D walk, except scaled by a larger pre-factor, as the particle can now get stuck in a component of B nodes at each position. As B increases the particle spends a greater period of time within each component, but scales at a lesser rate (for equated A, B) than $\bar{I}(A)$. This may be because, from Equation 4, $E(B) \propto B^{1.11}$, reducing the scaling factor.

4.5. Multi-particle diffusion

Physical diffusion problems involve many more than one particle, so to simulate this I created an array of ‘particle’ structures that can move independently. For n particles, one can then track the average number of times each component is visited for a given number of iterations. Figure 6 illustrates these results, depicting a normal distribution of the particles about the start position as discussed in section 3, offering a promising validation of the theory.

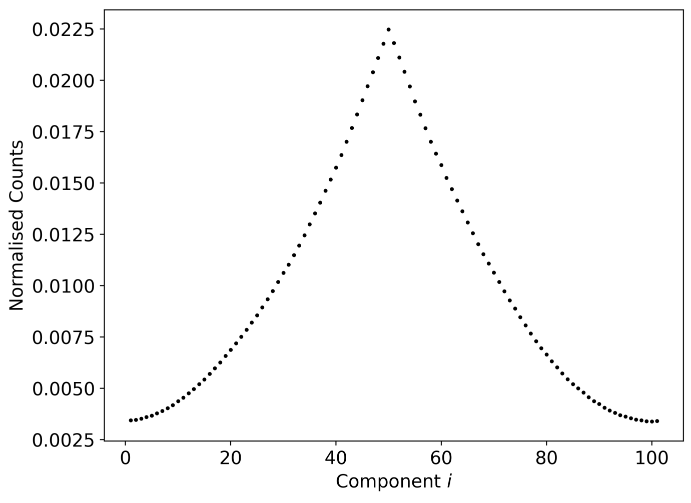

Fig. 6. Normalised distribution, centred at component 51 (start position), depicting the number of times a component, i, is visited for 5 particles and 10^4 iterations at $A = 100$.