Key concepts

- Random walkers can be configured using pseudorandom number generators. These can then be used to mimic real life natural occurrences such as Brownian motion, diffusion limited aggregation (DLA), and biological macromolecules.
- Simulating random phenomena require parameter conditions which limit the chaos of the system. These organize the walkers' motion which lead to distinguishable descriptions for the patterns. For DLA, this is seen as tree or branch-like patterns which stem from the initial seed. For Brownian motion, erratic movement and diffusion from an initial central point. Lastly, for the self- avoiding random walker, the generation of polymer-like macromolecule shapes.
- Monte Carlo simulations deal with approximating measurements using a randomized sample size. They are especially useful when finding values that require a sheer amount of individual measurements. By the law of large numbers, we're able get highly accurate approximations by increasing the sample size whilst maintaining a realistic number of calculations. Measurements are done based on correlation time, and the total time to do so is based on the equilibrium time.

Essential code snippets

```
# Random sampling
r = np.random.randint(1, 5)
if (r == 1) and (not a[x+1][y]):
    x += 1
    steps += 1
elif (r == 2) and (not a[x-1][y]):
    x -= 1
    steps += 1
elif (r == 3) and (not a[x][y+1]):
    y += 1
    steps += 1
elif (r == 4) and (not a[x][y-1]):
    y -= 1
    steps += 1
```

The heart of all these random walker applications is the pseudorandom number generator and the initially allowed motion of the object. This is the simplest snippet which denotes right, left, up, and down movement as the basic possible actions. The lattice/domain as foundation along with the conditions added afterwards will be paramount in setting the stage for the simulation of the given phenomenon.

The next notable part is the part of code which decides what the object will do for each time step. For DLA, this is shown as conditions wherein the object will stop depending on a set variable 'stickiness' and whether there is an adjacent particle that has already attached to the formed aggregation. This is simpler for the self-avoiding random walker which focuses on cases of self-trapping and avoiding points it has been to. For Brownian motion, the conditions describe diffusion due to collisions between the particles and the water molecules.

```
if np.any(adjacent): # If at least one of the positions
    # If the stickiness is 100%, the particle will be s
    first_hit = adjacent.nonzero()[0][0]
    x = steps[first_hit, 0]
    y = steps[first_hit, 1]
    current_radius = (x - L)**2 + (y - L)**2
    # If not, we should perform only single steps at th
    # does not collide with existing particles. At each
    # probability of sticking. If the particle goes out
    # cluster, we forget about it and introduce a new
    if (stickiness < 1):
        is_attached = (np.random.rand(1) < stickiness)
        inside = (current_radius < (max_radius + 2)**2)
        while ((not is_attached) and inside):
            while True:
```

```
# Generating random numbers to genereate random lattice sites
rand_list_index_1 = np.random.randint(N, size=N ** 2)
rand_list_index_2 = np.random.randint(N, size=N ** 2)
# Generating N ** 2 random numbers between 0 and 1
rand_numbers = np.random.rand(N**2)
for l in range(N ** 2):
    i = rand_list_index_1[l]
    k = rand_list_index_2[l]
    check, delta_E = Attempt_flip(S, beta, i, k, N, rand_numbe
    if check:
        delta_m = (-2) * S[i][k] / (N ** 2)
        S[i][k] *= (-1)
    else:
```

The last part I found most important is the generation of the random lattice sites which serve as the samples. This process along with the updating of spins is central to how we apply the Metropolis algorithm in simulating the Ising Model. These lead to expectation values of the lattice's properties.

Pitfalls

- Dealing with properly storing data in arrays and properly representing them in figures/graphs.
- General code semantics and translating one's thought process into code for simulations. As well as understanding the concepts in the first place. The Ising model is relatively complex.