Report 1: Single file diffusion with binding and unbinding
PHYC 6261

Taylor Dunn
B00688623

March 18, 2016

1 Model and Method

Our model is single file diffusion (SFD) of hard-core particles that cannot pass each other. Under these conditions, SFD typically exhibits subdiffusive behaviour characterized by mean-squared displacement $< \Delta x^2 > \sim t^{1/2}$ [1], in contrast with simple-diffusion (SD) scaling of $< \Delta x^2 > \sim t$.

It has been suggested that the luminal αTAT1 protein (radius of 3.5 nm [2]) may undergo SFD inside microtubules (inner radius of 7 mm [3]). An important consideration of this diffusive behaviour is the protein binding to sites inside the microtubule, by which it catalyzes the acetylation reaction.

To study the effects of transiently bound particles blocking diffusion, we use stochastic simulations of N particles in a ring geometry (or equivalently, a tube of finite length L with periodic boundary conditions). The ring is discretized into a lattice with L points, and particle positions and states are updated at each time step as follows:

1. If enough time has passed, bind/unbind a random free/bound particle
2. Choose a random particle.
3. If the particle is free, choose a random step direction (± 1).
4. If the new position is unoccupied, move the particle.
5. Update time by $\Delta t = 1/N$.

The choice of time step here ensures that at integer steps of time, each particle has, on average, attempted to move once. To implement the dynamics of binding and unbinding events, we introduce binding and unbinding rates K_{on} and K_{off}, respectively, and employ the Gillespie algorithm to determine the next event and when it occurs:

1. Compute the combined rate of the entire system:

$$\Gamma = N_f K_{\text{on}} + N_b K_{\text{off}} \tag{1}$$

2. Determine the time of the next event:

$$t_{\text{next}} = t - (1/\Gamma) \log [\text{RAN}(0, 1)] \tag{2}$$

3. Determine the probability of a binding event versus unbinding:

$$P_b = N_f K_{\text{on}}/\Gamma \tag{3}$$

For easier comparision of simulation parameters, we also introduce the occupancy number $p = N/L$ and the ratio $R = K_{\text{on}}/K_{\text{off}}$.

There are multiple approaches to initializing the system (e.g. all particles free, all particles equally spaced, all lattice sites having p probability of being occupied), but for the results presented here, the following process was used:

1. Compute the number of bound and free particles as $N_b = NR/(1+R)$ and $N_f = N - N_b$.
2. For each bound particle, place it at an unoccupied site on the ring.
3. For each free particle, place it at an unoccupied site on the ring.

At this initial stage of the project, we are exploring the following parameter space:

Parameter	Primary	Secondary	Fixed
K_{on}	$10^{-3}, 10^{-2}, 10^{-1}, 10^{0}, 10^{1}, 10^{2}, 10^{3}$	$10^{-2}, 10^{-1}, 10^{0}$	10^{-1}
R	0.1, 0.2, 0.5, 1, 2, 5, 10	0.1, 1, 10	1
p	0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9	0.1, 0.5, 0.9	0.5

Table 1: The parameter space to be tested. Each member of the class was assigned a primary and secondary variable range to explore.

The primary goal of this project is to examine how these parameters affect the behaviour of particles undergoing SFD. An important preliminary step, however, is to determine appropriate system size L to avoid finite-size effects, equilibration time t_{eq} to avoid initial transients, and maximum simulation time t_{max} to ensure that the system has equilibrated to its final behaviour.

2 Results

For a system with periodic boundary conditions, care must be taken to avoid finite-size effects. Ring sizes ranging from $L = 10$ to $L = 400$ were simulated up to $t_{\max} = 10^4$ at the fixed parameter values in Table 1, and the mean squared displacement with statistical error bars are plotted in Figure 1. To achieve good statistics, and sample the same overall number of particles, averages were taken over varying numbers of runs M such that $N = pLM = 10^4$.

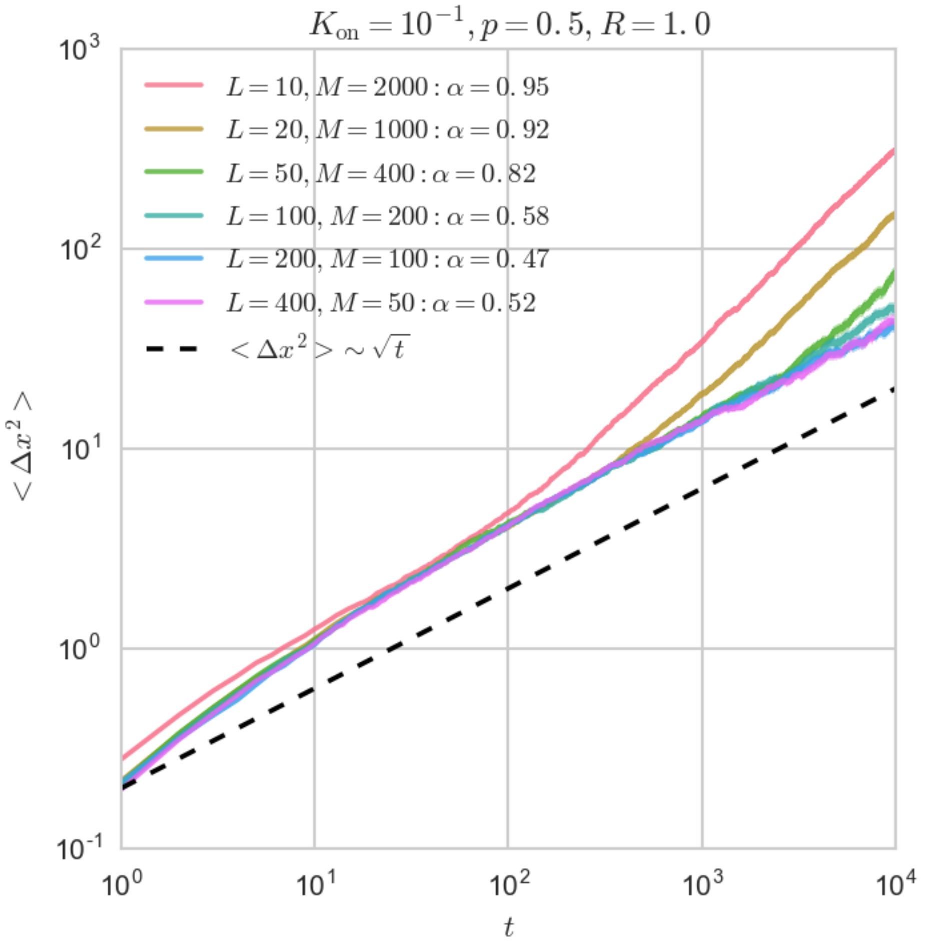

Figure 1: Mean squared displacement (± SEM) vs. time with varying system size L. Each set of data was fit to $< \Delta x^2 >= At^{\alpha}$, and best fit estimates for α are listed in the legend. Also plotted is a reference line corresponding to the familiar SFD result $< \Delta x^2 >\sim \sqrt{t}$.

At intermediate times (between $t = 10$ and 10^3), most of the data appears to scale as expected for SFD ($\sim \sqrt{t}$), but all of the data besides $L = 200$ and 400 looks to reach a diverging point where the particles exhibit superdiffusive ($\alpha > 0.5$) behaviour. This diverging point is earlier for smaller system sizes. From this result, we can conclude that a system size of $L = 200$ is an appropriate choice for this time scale, and for our fixed parameters.

Figure 1 raises another question: what is causing the super-diffusive behaviour at early times? One explanation is that the initial condition (randomly placed particles, bound/free ratio R) is not in equilibrium, so the system requires an equilibration time to reach its steady state before measurements should be taken. Figure 2 shows averages from $M = 100$ simulations for equilibration times ranging from $t_{\text{eq}} = 0$ to 10^4.

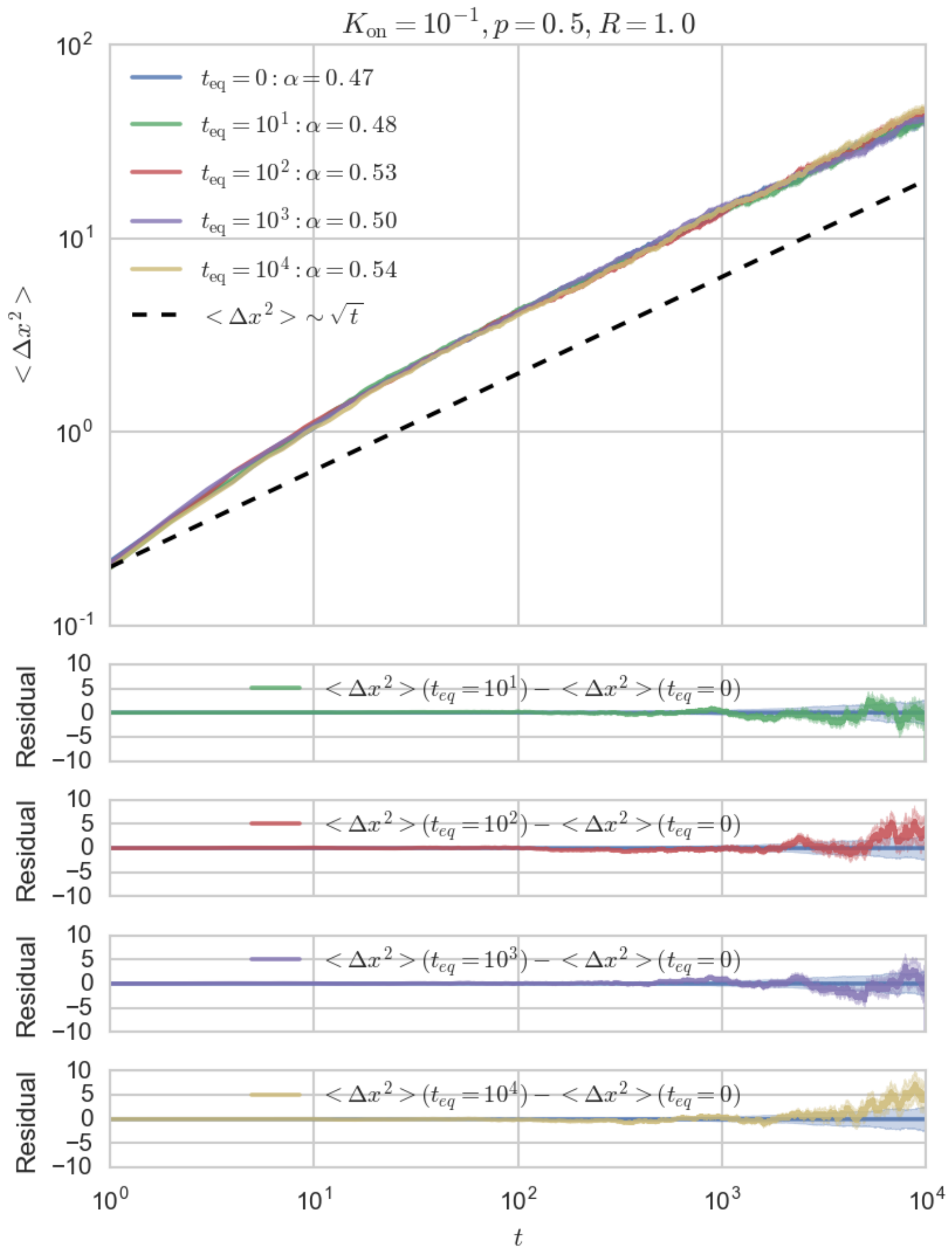

Figure 2: Mean squared displacement ($\pm$ SEM) vs. time with varying equilibration time t_{eq}. Each set of data was fit to $< \Delta x^2 > At^{\alpha}$, and best fit estimates for α are listed in the legend. Plotted below are the residuals of the displacement compared to $t_{\text{eq}} = 0$.

Despite some minor fluctuations at late times, the data lie on top of each other. This suggests that the initial condition has little to no effect on the measurement of $< \Delta x^2 >$, and consequently the best way to avoid the transient regime is to ignore it and only fit to data after a certain time For these parameters, it would appear that time is around $t > 10^2$.

As with any experiment, especially a collaborative one with independent simulation code, reproducibility is essential. Plotted in Figure 3 are sets of data from simulations ran by different group members at fixed parameter values. While there is disagreement at late times, especially compared to JP's data, this may just be due to statistical fluctuations at such large length scales. More cross-comparisons should be done at different parameter values to further ensure reproducibility.

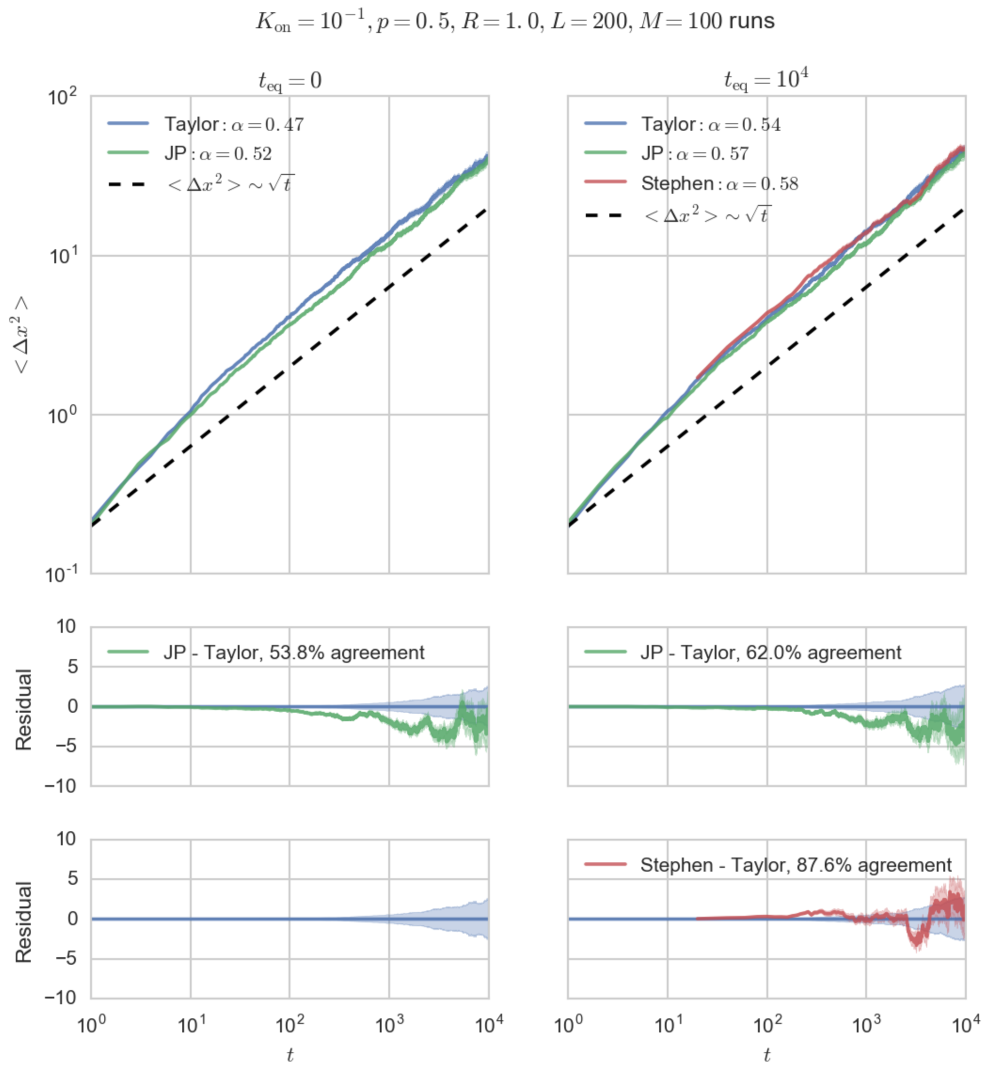

Figure 3: Mean squared displacement (± SEM) vs. time from simulations ran by different group members. Each set of data was fit to $< \Delta x^2 >= At^\alpha$, and best fit estimates for α are listed in the legend. Plotted below are the residuals of the displacement compared to Taylor's data, along with percent agreement within error bars.

With preliminary analysis complete, we now move on to exploring the parameter space of K_{on} primary and p secondary. A system size $L = 200$ was used, and $M = 100$ simulations were averaged to produce the data presented in Figure 4.

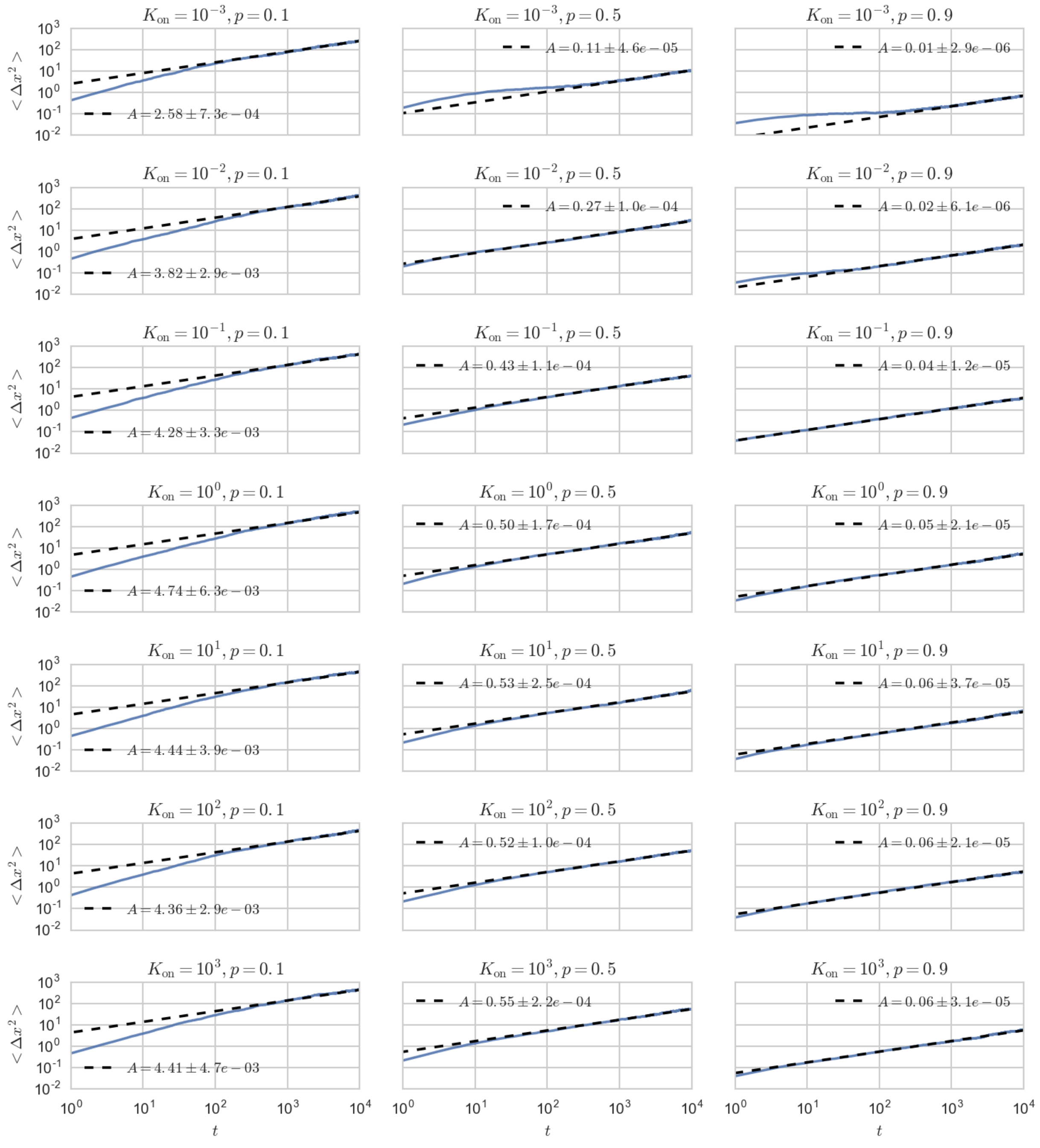

Figure 4: Mean squared displacement ($\pm$ SEM) vs. time for varying K_{on} and p values. Each set of data was fit to $< \Delta x^2 >= A\sqrt{t}$ for $t > 10^2$, and best fit estimates for the prefactor A are listed.

An immediate observation from the above figure is that the initial super-diffusive behaviour lasts longer for $p = 0.1$, up to $t = 10^3$ at high values of K_{on}. A possible explanation is that at lower densities, particles have more space to diffuse before running into other particles and "settling" to their steady-state behaviour. This may lead us to using a larger $t_{\max}$ for low p values in the future, but this should not affect the estimated fit parameters here significantly. Curiously, mean squared displacement for $K_{\mathrm{on}} = 10^{-3}$ and $p = 0.5, 0.9$ (also $K_{\mathrm{on}} = 10^{-2}$ and $p = 0.9$ to a lesser degree) seems to exhibit sub-diffusive ($\alpha < 0.5$) behaviour at early times. This should certainly be investigated further, and for smaller values of K_{on} than the range presented here.

The fit prefactors A are plotted versus K_{on} and p in Figure 5 below.

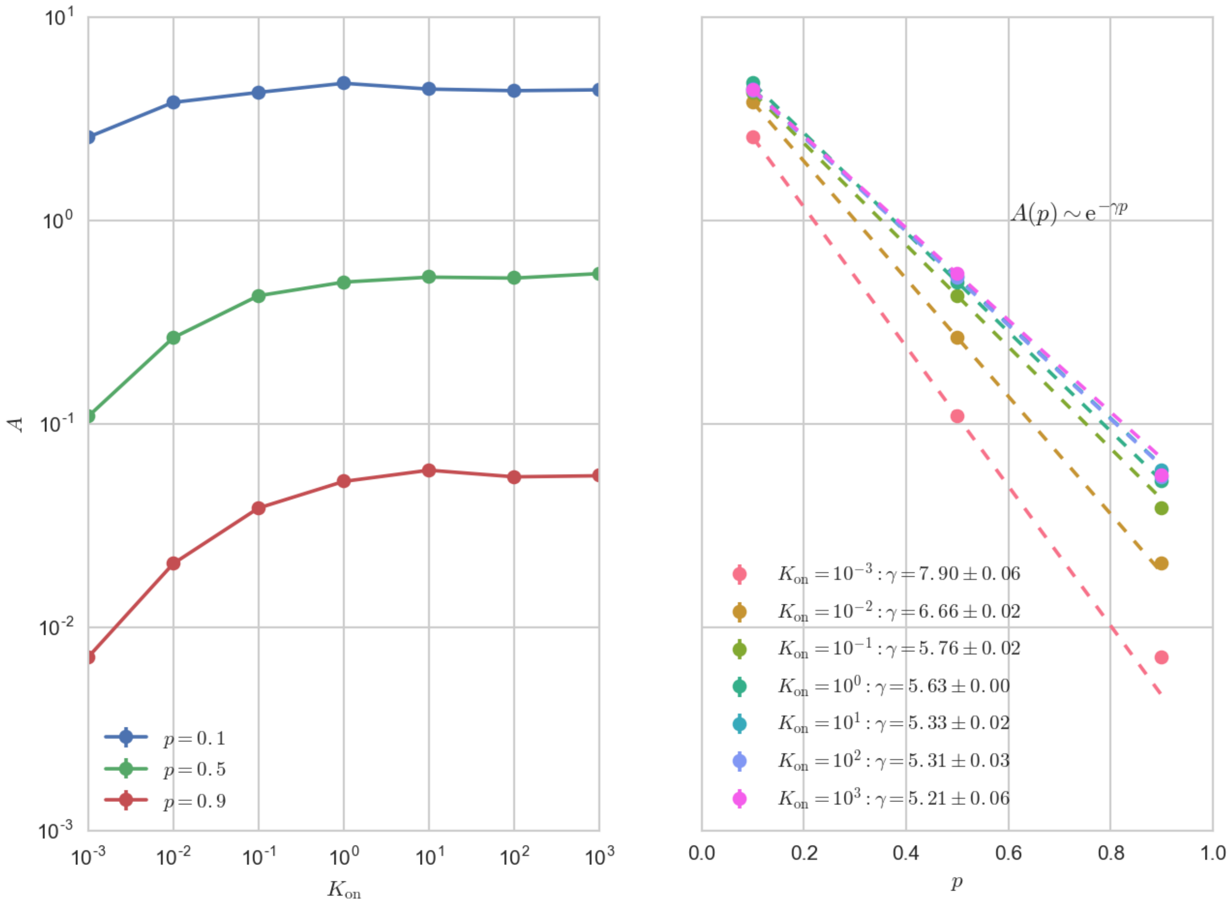

Figure 5: Prefactors A ($\pm$ error from the least squares fit) from Figure 4. A vs. p was fit to an exponential decay and the decay rates γ are listed in the legend.

With increasing K_{on}, A asymptotically approaches a constant, the value of which increases with decreasing p. This tells us that beyond $K_{\mathrm{on}} = 10$, the binding and unbinding rates are so fast that no difference can be discerned in the diffusive behaviour. A more appropriate K_{on} range to explore may be from 10^{-5} (or so) to 10^1. From both plots, it is clear that A decreases with p. This is expected because for high density, particles have less room to diffuse and thus have a smaller "amplitude" of displacement. Although only 3 values of

p were tested here, there is evidence of some scaling behaviour of A with p – specifically, it looks to be exponentially decaying. It will be interesting to compare the results of varying p primary and K_{on} secondary by Stephen, and investigate this relationship further.

3 Simulation code

```python
import numpy as np
import os
import time

def place_particles(N, L, R, initial_condition='ratio'):
    # Given the inital condition, set intial number of
    #  free and bound particles
    if initial_condition == 'ratio':
        Nb = int(N * R / (1+R))
        Nf = N - Nb
    elif initial_condition == 'bound':
        Nb = N
        Nf = 0
    elif initial_condition == 'unbound':
        Nf = N
        Nb = 0

    # Initialize the relevant arrays
    ring = np.zeros(L) # -1 = bound, 0 = unoccupied, 1 = unbound
    part_pos = np.zeros(N) # Particle positions on the ring
    part_states = np.zeros(N) # Particle states

    for i in range(N):
        # Keep picking a site until it's unoccupied
        site = np.random.randint(L)
        while ring[site] != 0:
            site = np.random.randint(L)

        part_pos[i] = site
        if i < Nb:
            # Place the particle on the ring in a bound state
            ring[site] = -1
            part_states[i] = -1
        else:
            # Otherwise place it unbound
            ring[site] = 1
            part_states[i] = 1
    return ring, part_pos, part_states

def simulate(N, L, ring, part_pos, part_states, kon, koff, tmax, teq=0):
    # First determine the number of particles that are bound/unbound
    Nf = len(part_states[part_states == 1])
    Nb = N − Nf

    # The time step is the average time for each particle to take one step
    # Each Monte Carlo step will therefore increment the time by 1/N
    nsteps = int(N * (tmax + teq))
    dt = 1 / N

    # Use arrays to keep track of ring state and displacement versus time
    ring_t = np.zeros([L, tmax+1])
    ring_t[:,0] = ring
    part_disp = np.zeros(N)
    disp_t = np.zeros([N, tmax+1])

    # Use the Gillespie algorithm to determine the first event
    gamma = Nb*koff + Nf*kon # Sum of rates
    tnext = − (1/gamma) * np.log(np.random.random_sample())
    Pb = Nf*kon / gamma # Probability of binding

    t = 0.0
    t_sample = 0
    for n in range(nsteps):
        if t >= tnext:
            # Determine what kind of event
            if np.random.random_sample() < Pb:
                # Binding event
                # Get the indicies of the unbound particles
                unbound = np.where(part_states == 1)[0]
                # Randomly choose the particle to be bound
                i = np.random.choice(unbound)
                part_states[i] = −1
                ring[part_pos[i]] = −1
                Nb += 1
                Nf −= 1
            else:
                # Unbinding event
                bound = np.where(part_states == −1)[0]
                i = np.random.choice(bound)
                part_states[i] = 1
                ring[part_pos[i]] = 1
                Nf += 1
                Nb -= 1

            # Find the next event
            gamma = Nb*koff + Nf*kon
            tnext = t - (1/gamma) * np.log(np.random.random_sample())
            Pb = Nf*kon / gamma

        # Pick a random particle to move
        i = np.random.randint(N)

        # If the particle is free
        if part_states[i] == 1:
            # Choose a direction
            dx = np.random.choice([-1, 1])

            new_pos = (part_pos[i] + dx) % L # Periodic boundary conditions

            # If the new position is unoccupied
            if ring[new_pos] == 0:
                # Update the ring occupancy
                ring[new_pos] = 1
                ring[part_pos[i]] = 0
                # Move the particle
                part_pos[i] = new_pos
                # If the equilibration time has passed, update the
                #  particle displacement
                if t >= teq:
                    part_disp[i] += dx

        if (t >= teq):
            t_sample += 1

            # If a full time step has passed (each particle has been moved
            #  on average), record position and displacement
            if (t_sample % N == 0):
                ring_t[:,int(t_sample/N)] = ring
                disp_t[:,int(t_sample/N)] = part_disp

        # Update the time
        t += dt

    return ring_t, disp_t

def write_data(data, kon, p, R, tmax, L, M, columns='time,dx2,dx2_err',
               output_directory='', run_time=0.0):
```

```
    filename = 'k{0}_p{1}_r{2}.csv'.format(kon, p, R)

    header = """kon = %f
             p = %f
             R = %f
             tmax  = %d
             teq = %d
             L  = %d
             M  = %d
             run time %f seconds
             columns: %s
             """ % (kon, p, R, tmax, teq, L, M, run_time, columns)

    with open(output_directory + filename, 'wb') as f:
        np.savetxt(f, data, delimiter=',', header=header)
    f.close()

def run_simulations(kon, p, R, tmax, L, M, teq=0, seed=None):
    if seed is not None:
        np.random.seed(seed)

    koff = kon / R # Unbinding rate
    N = int(p * L) # Number of particles

    # Hold the average particle displacements in an array
    avg_part_disp2 = np.empty([M, tmax+1])
    avg_part_disp2_err = np.empty([M, tmax+1])

    for run in range(M):
        ring, part_pos, part_states = place_particles(N, L, R)
        ring_t, disp_t = simulate(N, L, ring, part_pos, part_states,
                                  kon, koff, tmax, teq)

        avg_part_disp2[run,:] = np.mean(disp_t**2, axis=0)
        avg_part_disp2_err[run,:] = np.std(disp_t**2, axis=0) / \
                                    np.sqrt(disp_t.shape[0])

    # Calculate the mean and standard error over the M runs
    avg_run_disp2 = np.mean(avg_part_disp2, axis=0)
    avg_run_disp2_err = np.std(avg_part_disp2, axis=0) / \
                        np.sqrt(avg_part_disp2.shape[0])
    return avg_run_disp2, avg_run_disp2_err

if __name__ == '__main__':
```

```
kon_fixed = 1e-1 # Binding rate
p_fixed = 0.5 # Fraction of sites occupied
R_fixed = 1.0 # Ratio of binding to unbinding
teq = 0 # Equilibration time
tmax = int(1e4) # Maximum simulation time (beyond equilibration)
L = 200 # Ring size
M = 100 # Number of simulations

kon_primary = [10**i for i in range(-3, 3)]
p_secondary = [0.1, 0.5, 0.9]

start_time = time.time()

for kon in kon_primary:
    for p in p_secondary:
        stime = time.time()
        print(kon, p)

        avg_run_disp2, avg_run_disp2_err = run_simulations(kon, p,
                                                  R_fixed, tmax, L, M, teq)
        t = np.arange(len(avg_run_disp2))
        data = np.transpose([t, avg_run_disp2, avg_run_disp2_err])
        output_directory = '../data/avg_runs/'
        columns = 'time,dx2,dx2_err'

        output_directory += 'm{0}_l{1}_tm{2}_te{3}/' \
                            .format(M, L, tmax, teq)
        if not os.path.exists(output_directory):
            os.makedirs(output_directory)

        rtime = time.time() - stime
        write_data(data, kon, p, R_fixed, tmax, L, M,
                   columns=columns,
                   output_directory=output_directory, run_time=rtime)

        print('Finished after {0:.1f} seconds'.format(rtime))

run_time = time.time() - start_time
print('Finished all runs after {0:.1f} seconds'.format(run_time))
```

References

[1] Theodore Edward Harris. Diffusion with "collisions" between particles. *Journal of Applied Probability*, 2(2):323–338, 1965.

[2] Guillaume Montagnac, Vannary Meas-Yedid, Marie Irondelle, Antonio Castro-Castro, Michel Franco, Toshinobu Shida, Maxence V Nachury, Alexandre Benmerah, Jean-Christophe Olivo-Marin, and Philippe Chavrier. αtat1 catalyses microtubule acetylation at clathrin-coated pits. *Nature*, 502(7472):567–570, 2013.

[3] Toshinobu Shida, Juan G Cueva, Zhenjie Xu, Miriam B Goodman, and Maxence V Nachury. The major α-tubulin k40 acetyltransferase αtat1 promotes rapid ciliogenesis and efficient mechanosensation. *Proceedings of the National Academy of Sciences*, 107(50):21517–21522, 2010.