Gillespie Algorithm:

For $\xrightarrow{\lambda} \emptyset$ (k-particle annihilation reaction).

(a). Master equation.

$$\frac{\partial P_n(t)}{\partial t} = \lambda \left(\underbrace{\binom{n+k}{k} P_{n+k}(t)}_{\text{gain term}} - \underbrace{\binom{n}{k} P_n(t)}_{\text{loss term}} \right).$$

combination in which the reaction occurs

number of possible combinations of the n particles for k reactants to occur

$(n-k) \xleftarrow{\text{loss}} (n) \xleftarrow{\text{gain}} (n+k)$

(b).

We want to argue that this is equivalent to a biased random walk, with $q = 1-p$ the probability of staying and p = the prob. of hopping k-steps. With $p = \lambda \binom{n}{k}$.

$(n-k) \xleftarrow{p} (n) \circlearrowleft 1-p$

$p = \underbrace{\lambda \binom{n}{k}}_{\text{total no. of reaction}} dt$ (for annihilation).

$\underbrace{}_{\text{prob. of reaction}}$

Likewise:

$1-p$: prob. of "no reaction" (failure).

- For use of this to make sense, we need to make dt be sufficiently small. (otherwise more than one reaction can occur in a single time-step and our one-step model fails).

This isn't very efficient! (computationally).

- initially, we have lots of particles ⇒ "fast" reactions: $\binom{n}{k}\lambda$ is big;
- As n decreases, it will take progressively longer for a reaction to occur (on average), which means we'll have lots of "pointless" steps that are clogging our simulation, i.e., we're forced to generate many (pseudo-)random numbers that most likely will not advance the simulation (by advance, I mean move towards $n=0$).

(2). · Prob. that it changes: $p = rn\,dt$
· Prob. that it doesn't change: $q = 1 - rn\,dt$.

What about after $\tau = A\,dt$:

· $q_n(\tau) = \underbrace{q \cdot q \cdots q}_{A\ \text{times}} = q^A = (1 - rn\,dt)^A$.

· $q_n(\tau + d\tau) = q^{A+1} = q \cdot q_n(\tau)$.
& $d\tau = dt$

· In this derivation, I assume that only one reaction occurs in $(\tau, \tau + d\tau)$

This way: $\frac{q_n(\tau + d\tau) - q_n(\tau)}{d\tau} = q_n(\tau) \frac{q - 1}{d\tau} \Rightarrow$

$\Rightarrow \frac{dq_n(\tau)}{d\tau} = -q_n(\tau) \cdot rn \frac{dt}{d\tau} \Rightarrow \frac{dq_n(\tau)}{d\tau} = -rn\,q_n(\tau)$. QED.

$\Rightarrow q_n(\tau) = q_n(0) \cdot e^{-rn\tau} \Rightarrow$

$\Rightarrow q_n(\tau) = e^{-rn\tau}$.

· Reasoning: & $\tau = 0$: $q_n(0) = 1$
· If $\tau = 0$, then the number of reactants certainly does not change! $\Rightarrow$
$\Rightarrow$ Prob. of not changing: $= 1 = q_n(0)$.

· Now, for $p_n(\tau)$:

· $p_n(\tau) = q^{A-1} \cdot p$
· $p_n(\tau + d\tau) = q^A \cdot p$
· & $d\tau = dt$.

$\Rightarrow \frac{p_n(\tau + d\tau) - p_n(\tau)}{d\tau} = q^{A-1} p \cdot \frac{q - 1}{d\tau} \Rightarrow$

$\Rightarrow \frac{dp_n(\tau)}{d\tau} = -p_n(\tau) \cdot rn \frac{dt}{d\tau} \Rightarrow \frac{dp_n(\tau)}{d\tau} = -rn\,p_n(\tau)$

$\Rightarrow p_n(\tau) = p_n(0) \cdot e^{-rn\tau}$

· initial condition? $p_n(\tau = 0) = rn$?
I'm not sure about this...

· Used in the Gillespie Algorithm.

$F(\tau) = 1 - e^{-rn\tau}$
$\Rightarrow F(\tau) = x$, $x \in \text{Uniform}(0,1)$
$\Rightarrow \boxed{\tau = -\frac{\ln(1-x)}{r \cdot n}}$.

(3). Computational implementation of the Gillespie Algorithm.

$p_n(\tau) = rn\,e^{-rn\tau} \Rightarrow p_n(A\tau)$ ... $p_n(0) \cdot e^{-rn\tau} \Rightarrow$
$\Rightarrow p_n(\ldots)\,n \cdot \tau$

Page 2.

Mean-field description (needed for comparison with the experimental results).

$$\frac{\partial P(n,t)}{\partial t} = \frac{\lambda}{k!}\left[\frac{(n+k)!}{n!}\cdot P(n+k,t) - \frac{n!}{(n-k)!}P(n,t)\right] \Rightarrow$$

$(n+k)\dots(n+1)$ $n(n-1)\dots(n-k+1)$

multiply by n and $\sum_n$

$$\Rightarrow \frac{\partial\langle n(t)\rangle}{\partial t} = \frac{\lambda}{k!}\left[\sum_{n'=k}^{\infty} n'(n'-1)\dots(n'-k)P(n',t) - \sum_{n=0}^{\infty} n^2(n-1)\dots(n-k+1)P(n,t)\right] \Rightarrow$$

$n+k \to n'$; $n'=k$: can start at 0 since these initial terms don't contribute to the sum.

$k=3$

$$\Rightarrow \frac{\partial\langle n(t)\rangle}{\partial t} = \frac{\lambda}{k!}\sum_{n=0}^{\infty}\left[n(n-1)(n-2)(n-3)P(n,t) - n^2(n-1)(n-2)P(n,t)\right] \Rightarrow$$

$$\Rightarrow \frac{\partial\langle n(t)\rangle}{\partial t} = \frac{\lambda}{3!}\Big[\langle n(n-1)(n-2)(n-3)\rangle - \langle n^2(n-1)(n-2)\rangle\Big].$$

$\Rightarrow$ ... After some algebra ... $\Rightarrow$ Here, I assume no correlation, i.e. $\langle n^2\rangle \approx \langle n\rangle^2$ and $\langle n^3\rangle \approx \langle n\rangle^3$.

$$\Rightarrow \frac{\partial\langle n(t)\rangle}{\partial t} = \frac{\lambda\cdot 3}{3!}\left(-\langle n\rangle^3 + 3\langle n\rangle^2 - 2\langle n\rangle\right) \Rightarrow$$

- if $\langle n\rangle \gg 1$: $\langle n\rangle^3$ dominates the RHS.

$$\Rightarrow \frac{\partial\langle n(t)\rangle}{\partial t} = -\frac{\lambda}{2}\cdot\langle n\rangle^3 \Rightarrow$$

$$\Rightarrow \frac{\partial\langle n(t)\rangle}{\langle n\rangle^3} = -\frac{\lambda}{2}\partial t \Rightarrow -\frac{1}{2\langle n(t)\rangle^2} = -\frac{\lambda t}{2} + C \Rightarrow$$

$$\Rightarrow \langle n(t)\rangle = \frac{1}{\sqrt{\lambda t + C}}.$$

$$\langle n(t=0)\rangle = n_0 \Rightarrow C = \frac{1}{n_0^2} \Rightarrow \boxed{\langle n(t)\rangle = \frac{1}{\sqrt{n_0^{-2} + \lambda t}}}$$

Gillespie-Algorithm_Notebook

May 14, 2023

1 Open TA: Gillespie algorithm

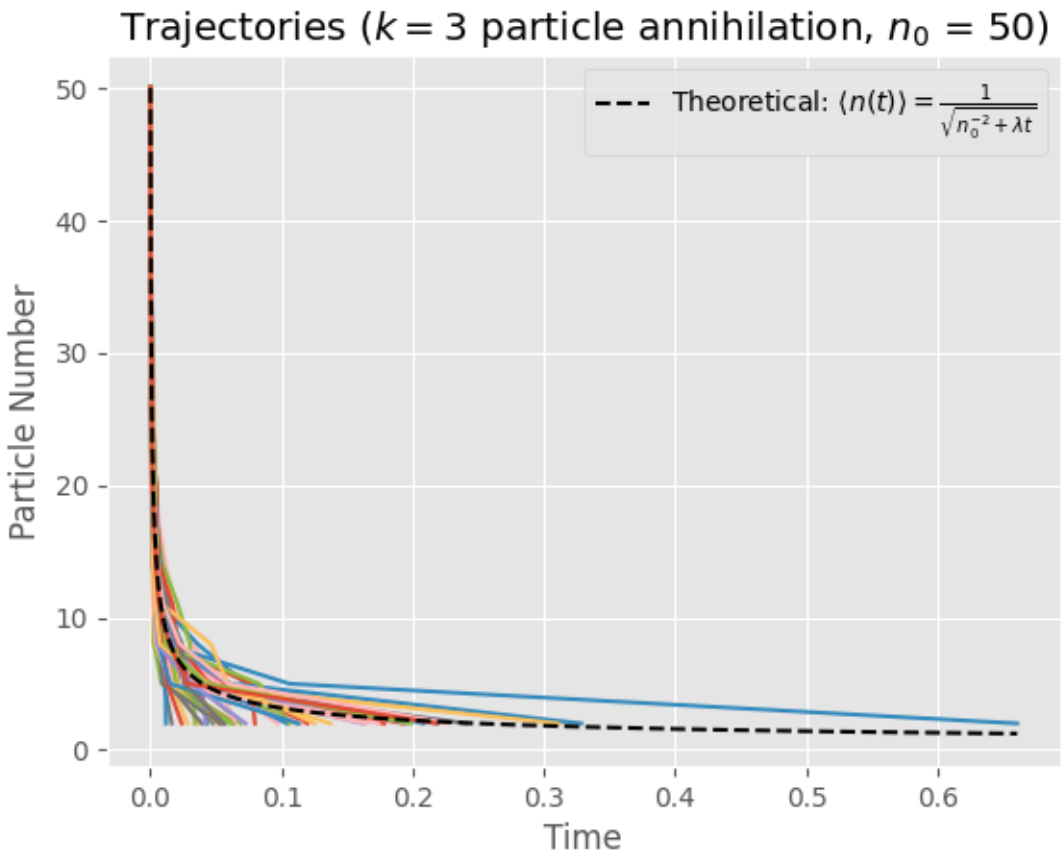
1.1 Question d): Implementation

1.1.1 Case A: $n_0 = 50$

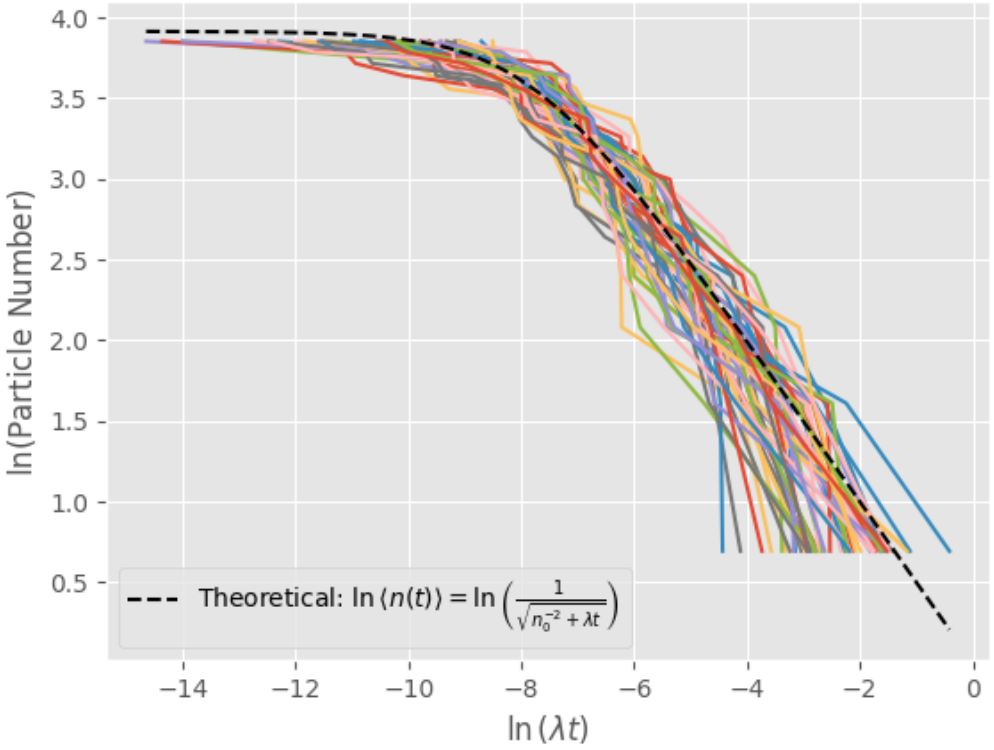
[ ]:
```
import random
import matplotlib.pyplot as plt
import numpy as np
import math
plt.style.use('ggplot')

lambda_val = 1.0         # Define the reaction rate constant
n0         = 50; n = n0  # Define the initial particle number
timesVec   = []          # List to store the trajectories
Flag       = True        # Flag
particles = [n0]

for iter in range(50):
    # Define the simulation results
    times = [0.0];
    n     = n0

    # Run the Gillespie algorithm
    while n >= 3:
        # Calculate the reaction probability
        rn = lambda_val * n * (n - 1) * (n - 2) / 6.

        # Draw the next reaction time from an exponential distribution
        tau = -math.log(random.random()) / rn

        # Update the time and particle number
        t = times[-1] + tau
        n -= 3

        # Store the results
        times.append(t)
        if(Flag): particles.append(n)
```

```
    timesVec.append(times)
    Flag = False

# Plot the results
for i in range(50):
    plt.plot(timesVec[i], particles)
More_t = np.linspace(min([sorted(tVec)[1] for tVec in timesVec]),␣
 ↪max([max(tVec) for tVec in timesVec]), 1_000_000)
# 'min([sorted(tVec)[1] for tVec in timesVec])': Used to exclude the 0.0 point␣
 ↪in time.
plt.plot(More_t, 1/np.sqrt(n0**(-2) + lambda_val*np.array(More_t)), linestyle =␣
 ↪'dashed',
        label = r'Theoretical: $\left\langlen(t)\right\rangle =␣
 ↪\frac{1}{\sqrt{n_0^{-2} + \lambda t}}$',
        color = 'black', linewidth = 1.5)
plt.xlabel('Time')
plt.ylabel('Particle Number')
plt.title(rf'Trajectories ($k = 3$ particle annihilation, $n_0$ = {n0})')
plt.legend()
plt.grid(True)
plt.show()

# Plot the results: log-log plot.
for i in range(50): # '[1:]': Needed since we start at t=0. log(0) = -inf,␣
 ↪which is problematic.
    plt.plot(np.log(lambda_val*np.array(timesVec[i])[1:]), np.log(particles)[1:
 ↪])
plt.plot(np.log(lambda_val*np.array(More_t)), np.log(1/np.sqrt(n0**(-2) +␣
 ↪lambda_val*np.array(More_t))), linestyle = 'dashed',
    label = r'Theoretical: $\ln{\left\langlen(t)\right\rangle} =␣
 ↪\ln{\left(\frac{1}{\sqrt{n_0^{-2} + \lambda t}}\right)}$',
    color = 'black', linewidth = 1.5) # Fix  this graph. It starts mid-way␣
 ↪through.
plt.xlabel('$\ln{(\lambda t)}$')
plt.ylabel(r'$\ln($Particle Number$)$')
plt.title(rf'Trajectories ($k = 3$ particle annihilation, $n_0$ = {n0}):␣
 ↪log-log plot')
plt.legend()
plt.grid(True)
plt.show()

# Experimental mean: Awkward definition though.
MeanVec = []
for j in range(len(timesVec[0])):
    MeanVec.append(sum([timesVec[i][j] for i in range(50)])/50.)
```

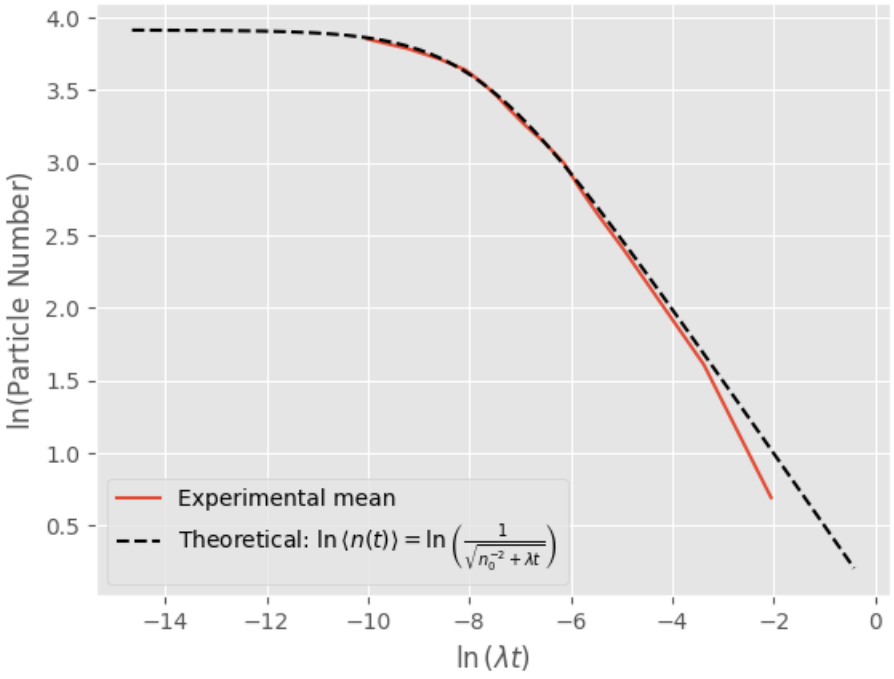
```
plt.plot(np.log(lambda_val*np.array(MeanVec)[1:]), np.log(particles)[1:], label␣
 ↪= 'Experimental mean')
plt.plot(np.log(lambda_val*np.array(More_t)), np.log(1/np.sqrt(n0**(-2) +␣
 ↪lambda_val*np.array(More_t))), linestyle = 'dashed',
    label = r'Theoretical: $\ln{\left\langlen(t)\right\rangle} =␣
 ↪\ln{\left(\frac{1}{\sqrt{n_0^{-2} + \lambda t}}\right)}$',
    color = 'black', linewidth = 1.5) # Fix  this graph. It starts mid-way␣
 ↪through.
plt.xlabel('$\ln{(\lambda t)}$')
plt.ylabel(r'$\ln($Particle Number$)$')
plt.title(rf'Gillespie algorithm ($k = 3$ particle annihilation, $n_0$ = {n0}):␣
 ↪log-log plot')
plt.legend()
plt.grid(True)
plt.show()
```



Trajectories ($k = 3$ particle annihilation, $n_0 = 50$): log-log plot



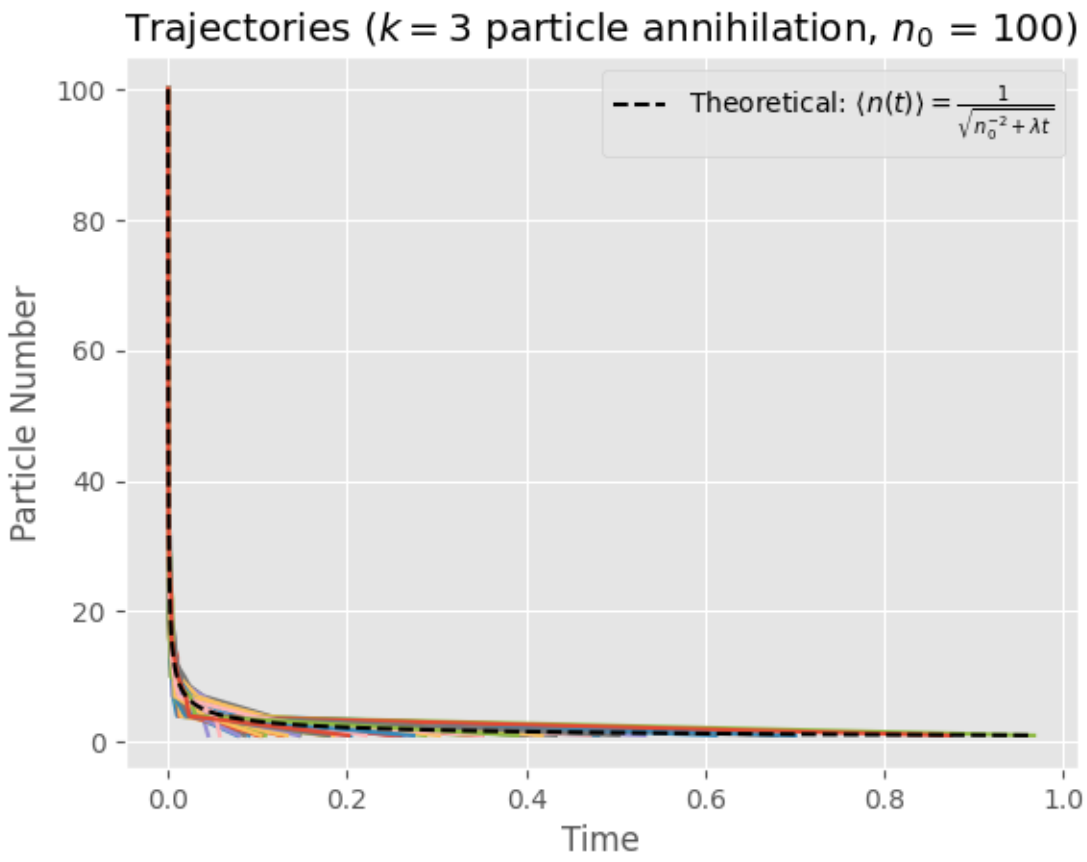
Gillespie algorithm ($k = 3$ particle annihilation, $n_0 = 50$): log-log plot



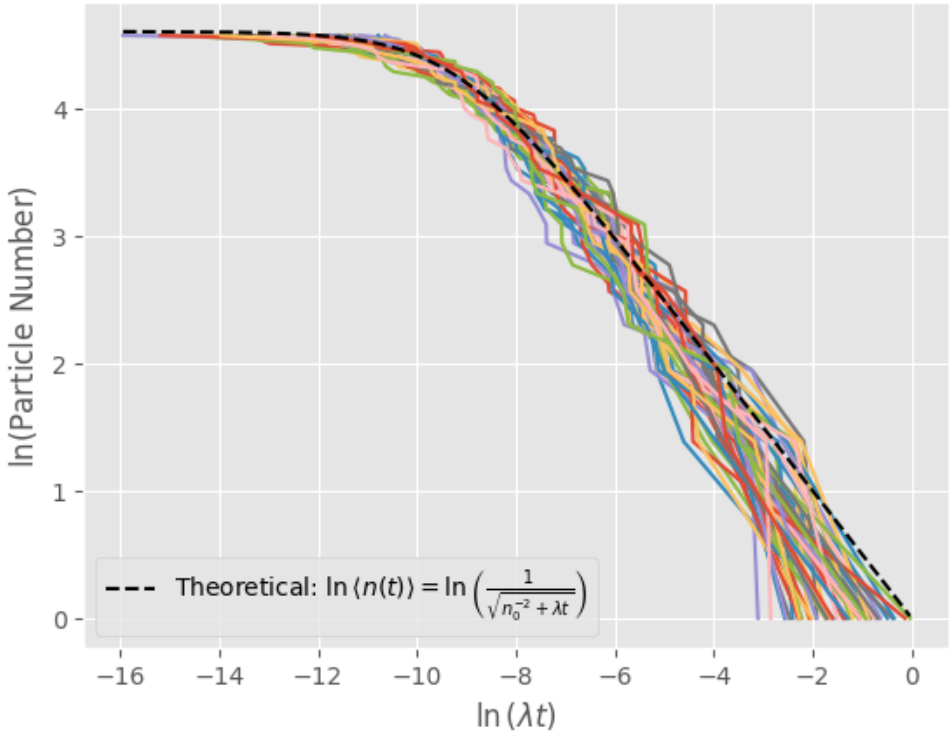
1.1.2 Case B: $n_0 = 100$

[ ]:
```
lambda_val = 1.0          # Define the reaction rate constant
n0         = 100; n = n0 # Define the initial particle number
timesVec   = []          # List to store the trajectories
Flag       = True        # Flag
particles = [n0]

for iter in range(50):
    times = [0.0];
    n     = n0
    while n >= 3:
        rn = lambda_val * n * (n - 1) * (n - 2) / 6.
        tau = -math.log(random.random()) / rn
        t = times[-1] + tau
        n -= 3
        times.append(t)
        if(Flag): particles.append(n)
    timesVec.append(times)
    Flag = False

# Plot the results
for i in range(50):
    plt.plot(timesVec[i], particles)
More_t = np.linspace(min([sorted(tVec)[1] for tVec in timesVec]),␣
 ↪max([max(tVec) for tVec in timesVec]), 1_000_000)
plt.plot(More_t, 1/np.sqrt(n0**(-2) + lambda_val*np.array(More_t)), linestyle =␣
 ↪'dashed',
        label = r'Theoretical: $\left\langlen(t)\right\rangle =␣
 ↪\frac{1}{\sqrt{n_0^{-2} + \lambda t}}$',
        color = 'black', linewidth = 1.5)
plt.xlabel('Time')
plt.ylabel('Particle Number')
plt.title(rf'Trajectories ($k = 3$ particle annihilation, $n_0$ = {n0})')
plt.legend()
plt.grid(True)
plt.show()

# Plot the results: log-log plot.
for i in range(50): # '[1:]': Needed since we start at t=0. log(0) = -inf,␣
 ↪which is problematic.
    plt.plot(np.log(lambda_val*np.array(timesVec[i])[1:]), np.log(particles)[1:
 ↪])
```

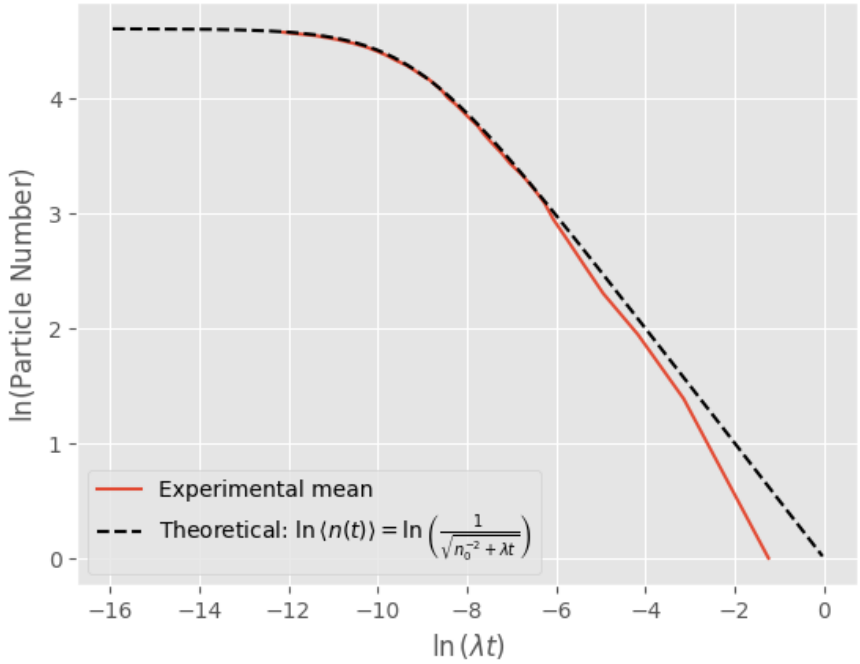
```
plt.plot(np.log(lambda_val*np.array(More_t)), np.log(1/np.sqrt(n0**(-2) +␣
 ↪lambda_val*np.array(More_t))), linestyle = 'dashed',
    label = r'Theoretical: $\ln{\left\langlen(t)\right\rangle} =␣
 ↪\ln{\left(\frac{1}{\sqrt{n_0^{-2} + \lambda t}}\right)}$',
    color = 'black', linewidth = 1.5) # Fix  this graph. It starts mid-way␣
 ↪through.
plt.xlabel('$\ln{(\lambda t)}$')
plt.ylabel(r'$\ln($Particle Number$)$')
plt.title(rf'Trajectories ($k = 3$ particle annihilation, $n_0$ = {n0}):␣
 ↪log-log plot')
plt.legend()
plt.grid(True)
plt.show()

# Experimental mean: Awkward definition though.
MeanVec = []
for j in range(len(timesVec[0])):
    MeanVec.append(sum([timesVec[i][j] for i in range(50)])/50.)

plt.plot(np.log(lambda_val*np.array(MeanVec)[1:]), np.log(particles)[1:], label␣
 ↪= 'Experimental mean')
plt.plot(np.log(lambda_val*np.array(More_t)), np.log(1/np.sqrt(n0**(-2) +␣
 ↪lambda_val*np.array(More_t))), linestyle = 'dashed',
    label = r'Theoretical: $\ln{\left\langlen(t)\right\rangle} =␣
 ↪\ln{\left(\frac{1}{\sqrt{n_0^{-2} + \lambda t}}\right)}$',
    color = 'black', linewidth = 1.5) # Fix  this graph. It starts mid-way␣
 ↪through.
plt.xlabel('$\ln{(\lambda t)}$')
plt.ylabel(r'$\ln($Particle Number$)$')
plt.title(rf'Gillespie algorithm ($k = 3$ particle annihilation, $n_0$ = {n0}):␣
 ↪log-log plot')
plt.legend()
plt.grid(True)
plt.show()
```


Trajectories ($k = 3$ particle annihilation, $n_0 = 100$)
Theoretical: $\langle n(t) \rangle = \frac{1}{\sqrt{n_0^{-2} + \lambda t}}$
Particle Number
Time
0
20
40
60
80
100
0.0
0.2
0.4
0.6
0.8
1.0

Trajectories ($k = 3$ particle annihilation, $n_0 = 100$): log-log plot



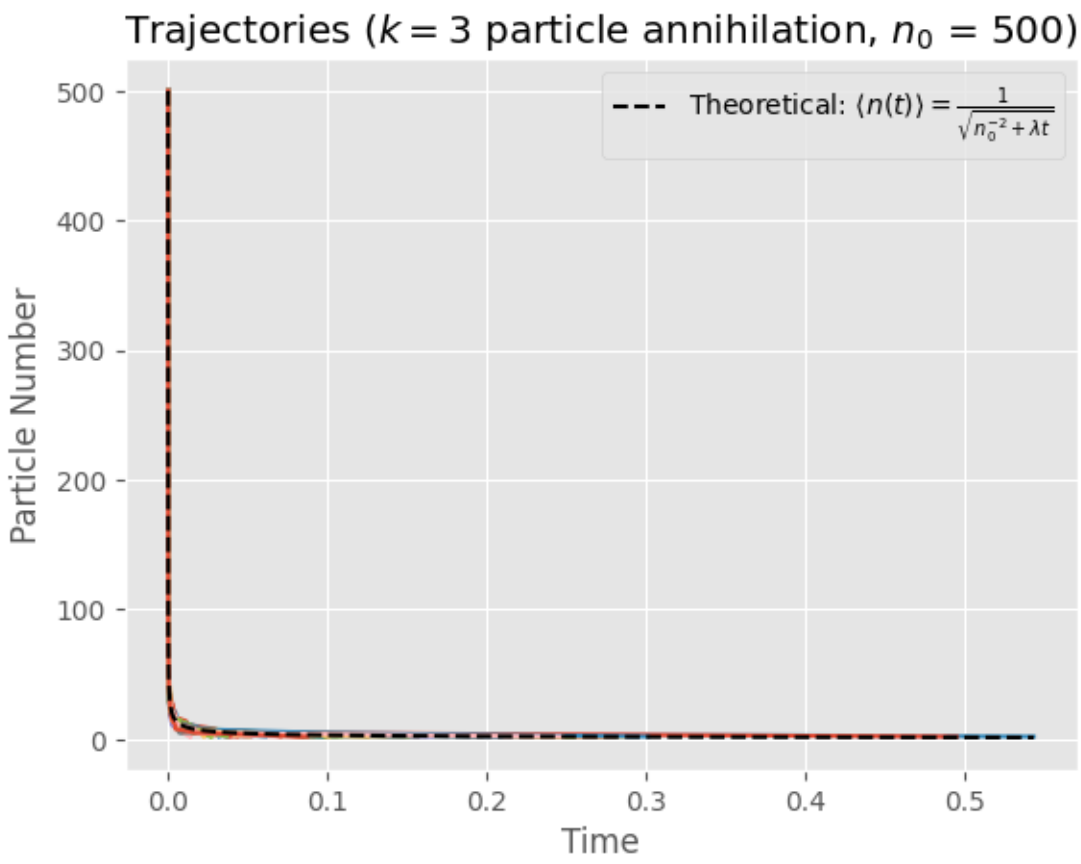
Gillespie algorithm ($k = 3$ particle annihilation, $n_0 = 100$): log-log plot



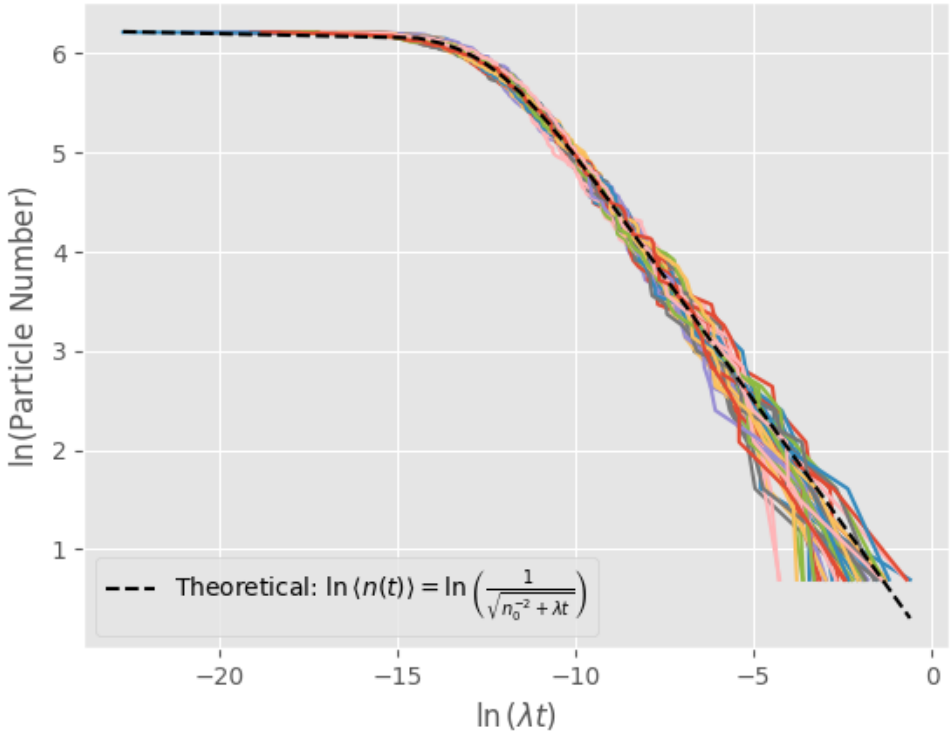
1.1.3 Case C: $n_0 = 500$

[ ]:
```
lambda_val = 1.0          # Define the reaction rate constant
n0         = 500; n = n0 # Define the initial particle number
timesVec   = []          # List to store the trajectories
Flag       = True        # Flag
particles = [n0]

for iter in range(50):
    times = [0.0];
    n     = n0
    while n >= 3:
        rn = lambda_val * n * (n - 1) * (n - 2) / 6.
        tau = -math.log(random.random()) / rn
        t = times[-1] + tau
        n -= 3
        times.append(t)
        if(Flag): particles.append(n)
    timesVec.append(times)
    Flag = False

# Plot the results
for i in range(50):
    plt.plot(timesVec[i], particles)
More_t = np.linspace(min([sorted(tVec)[1] for tVec in timesVec]),␣
 ↪max([max(tVec) for tVec in timesVec]), 1_000_000)
plt.plot(More_t, 1/np.sqrt(n0**(-2) + lambda_val*np.array(More_t)), linestyle =␣
 ↪'dashed',
        label = r'Theoretical: $\left\langle(t)\right\rangle =␣
 ↪\frac{1}{\sqrt{n_0^{-2} + \lambda t}}$',
        color = 'black', linewidth = 1.5)
plt.xlabel('Time')
plt.ylabel('Particle Number')
plt.title(rf'Trajectories ($k = 3$ particle annihilation, $n_0$ = {n0})')
plt.legend()
plt.grid(True)
plt.show()

# Plot the results: log-log plot.
for i in range(50): # '[1:]': Needed since we start at t=0. log(0) = -inf,␣
 ↪which is problematic.
    plt.plot(np.log(lambda_val*np.array(timesVec[i])[1:]), np.log(particles)[1:
 ↪])
plt.plot(np.log(lambda_val*np.array(More_t)), np.log(1/np.sqrt(n0**(-2) +␣
 ↪lambda_val*np.array(More_t))), linestyle = 'dashed',
```

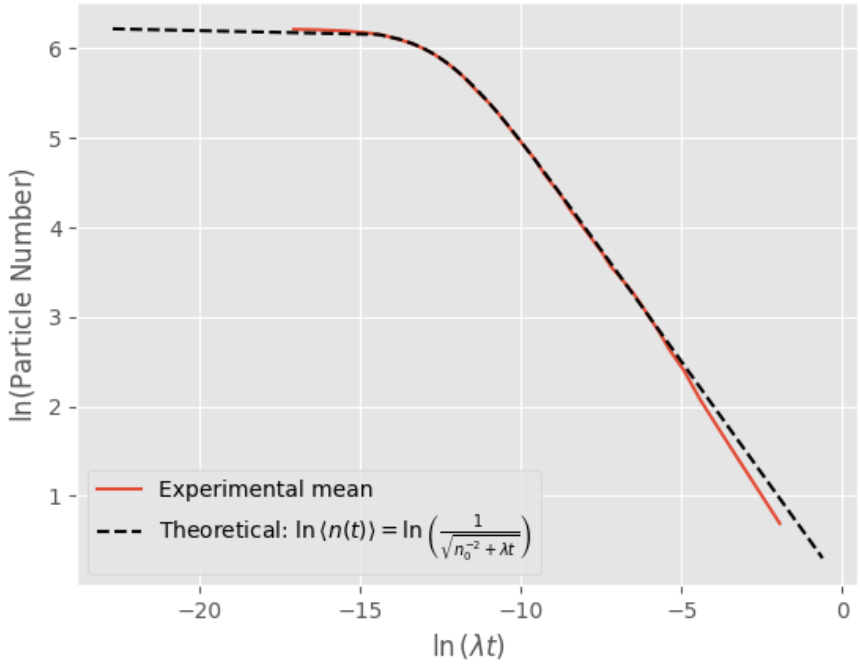
```
    label = r'Theoretical: $\ln{\left\langlen(t)\right\rangle} =␣
 ↪\ln{\left(\frac{1}{\sqrt{n_0^{-2} + \lambda t}}\right)}$',
    color = 'black', linewidth = 1.5) # Fix  this graph. It starts mid-way␣
 ↪through.
plt.xlabel('$\ln{(\lambda t)}$')
plt.ylabel(r'$\ln($Particle Number$)$')
plt.title(rf'Trajectories ($k = 3$ particle annihilation, $n_0$ = {n0}):␣
 ↪log-log plot')
plt.legend()
plt.grid(True)
plt.show()

# Experimental mean: Awkward definition though.
MeanVec = []
for j in range(len(timesVec[0])):
    MeanVec.append(sum([timesVec[i][j] for i in range(50)])/50.)

plt.plot(np.log(lambda_val*np.array(MeanVec)[1:]), np.log(particles)[1:], label␣
 ↪= 'Experimental mean')
plt.plot(np.log(lambda_val*np.array(More_t)), np.log(1/np.sqrt(n0**(-2) +␣
 ↪lambda_val*np.array(More_t))), linestyle = 'dashed',
    label = r'Theoretical: $\ln{\left\langlen(t)\right\rangle} =␣
 ↪\ln{\left(\frac{1}{\sqrt{n_0^{-2} + \lambda t}}\right)}$',
    color = 'black', linewidth = 1.5) # Fix  this graph. It starts mid-way␣
 ↪through.
plt.xlabel('$\ln{(\lambda t)}$')
plt.ylabel(r'$\ln($Particle Number$)$')
plt.title(rf'Gillespie algorithm ($k = 3$ particle annihilation, $n_0$ = {n0}):␣
 ↪log-log plot')
plt.legend()
plt.grid(True)
plt.show()
```


Trajectories ($k = 3$ particle annihilation, $n_0 = 500$)
Theoretical: $\langle n(t) \rangle = \frac{1}{\sqrt{n_0^{-2} + \lambda t}}$
Particle Number
Time
500
400
300
200
100
0
0.0
0.1
0.2
0.3
0.4
0.5

Trajectories ($k = 3$ particle annihilation, $n_0 = 500$): log-log plot



Gillespie algorithm ($k = 3$ particle annihilation, $n_0 = 500$): log-log plot



The results appear to be more accurate when n_0 is bigger. This is consistent with our assumption that $\langle n(t) \rangle \gg 1$. This is a lot more apparent in the initial part of the trajectory, for small t, where $n(t)$ is still large. In this region, the agreement with the theoretical prediction is clearly sharper. As $n(t)$ goes down, our approximation becomes less and less accurate, which explains the deviation of the experimental results from the theoretical prediction.