Theoretical Concept

The continuous-time Markov model for kinesin stepping from Fig. 6a in Ref. [1] describes a molecular motor moving along a microtubule as shown in Figure 1. It analyses the individual stepwise movements of kinesin travelling along microtubules.

Description of the continuous-time Markov model

At state **K**, there is absence of ATP and hence kinesin motor attach to the microtubule with one head, leaving the other one free. ATP molecules then bind to the head that is attached to the microtubule (State **K.T**) at a rate of k_1. At a transition rate of k_2, there is a nucleotide hydrolysis process from ATP to ADP, releasing a protein P_i and leading to state **K.D**. Finally, at a transition rate of k_{3f} or k_{3b}, the motor can either take a forward 8-nm step or detach and take a backward step respectively.

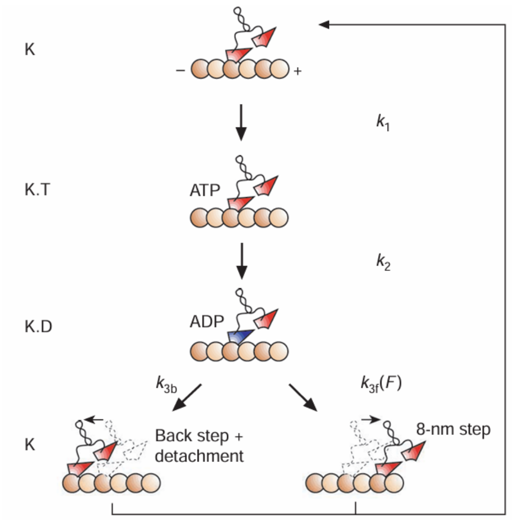

Figure 1: Stepping kinetics of the bidirectional movements [1]

The continuous-time Markov model for the kinesin stepping can be simply represented as shown in Figure 2.

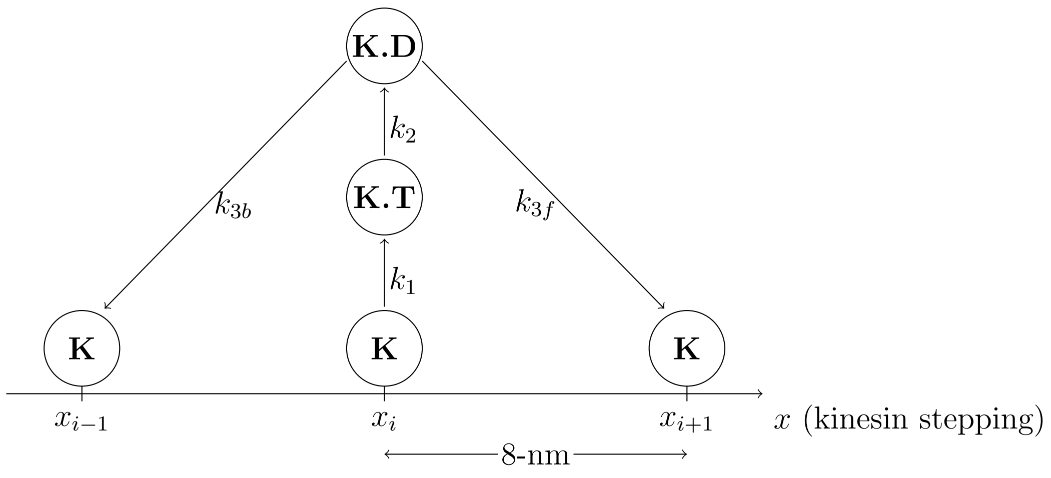

Figure 2: State Evolution and kinesin stepping

We have a 3-state continuous-time Markov model, where:

- **State K:** Corresponds to kinesin motor attached to the microtubule with one motor head.
- **State K.T:** Corresponds to kinesin with bound ATP. The ATP molecule binds to the head. This ATP binding initiates the next step in the cycle, preparing the kinesin for hydrolysis.
- **State K.D:** Corresponds to kinesin with bound ADP (or ADP + Pi). The ATP molecule that is bound to the head is hydrolyzed. After this, the kinesin motor moves either to the forward or backward direction, releasing ADP molecules.

Also, x represents the position of the kinesin motor along the microtubule. It is influenced by the stochastic transitions between the states. The release of ADP ($K.D \to K$) completes the cycle, resetting the motor for another step.

Master equation in terms of $\mathbf{k}_1$, $\mathbf{k}_2$, $\mathbf{k}_{3f}$, $\mathbf{k}_{3b}$

The Master equation for the probabilities P_K, $P_{K.T}$, and $P_{K.D}$ can be written as:

$$\frac{dP_K}{dt} = -k_1 P_K + (k_{3f} + k_{3b}) P_{K.D}$$

$$\frac{dP_{K.T}}{dt} = -k_2 P_{K.T} + k_1 P_K$$

$$\frac{dP_{K.D}}{dt} = -(k_{3f} + k_{3b}) P_{K.D} + k_2 P_{K.T}$$

and can be written in matrix form as:

$$\frac{d}{dt}\begin{bmatrix} P_K \\ P_{K.T} \\ P_{K.D} \end{bmatrix} = \begin{bmatrix} -k_1 & 0 & (k_{3f}+k_{3b}) \\ k_1 & -k2 & 0 \\ 0 & k_2 & -(k_{3f}+k_{3b}) \end{bmatrix} \begin{bmatrix} P_K \\ P_{K.T} \\ P_{K.D} \end{bmatrix}$$

which implies

$$\dot{\vec{P}}(t) = W\vec{P}(t)$$

Stationary probability as a function of $\mathbf{k}_1$, $\mathbf{k}_2$, $\mathbf{k}_{3f}$, $\mathbf{k}_{3b}$

At steady state, $\dot{\vec{P}}(t) = \vec{0}$. Hence the stationary probability $\vec{P}^{st}$ is giving as $\boldsymbol{W}\vec{P}^{st} = \vec{0}$.

Thus,

$$\begin{bmatrix} -k_1 & 0 & (k_{3f}+k_{3b}) \\ k_1 & -k2 & 0 \\ 0 & k_2 & -(k_{3f}+k_{3b}) \end{bmatrix} \begin{bmatrix} P_K^{st} \\ P_{K.T}^{st} \\ P_{K.D}^{st} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

which implies

$$-k_1 P_K^{st} + (k_{3f} + k_{3b}) P_{K.D}^{st} = 0 \tag{1}$$

$$-k_2 P_{K.T}^{st} + k_1 P_K^{st} = 0 \tag{2}$$

$$-(k_{3f} + k_{3b}) P_{K.D}^{st} + k_2 P_{K.T}^{st} = 0 \tag{3}$$

Based on the Komolkorov normalization condition,

$$P_K^{st} + P_{K.T}^{st} + P_{K.D}^{st} = 1 \tag{4}$$

From Equation 2:

$$P_K^{st} = \frac{k_2}{k_1} P_{K.T}^{st} \tag{2*}$$

Putting Equation 2* into Equation 4, we get:

$$\frac{k_2}{k_1} P_{K.T}^{st} + P_{K.T}^{st} + P_{K.D}^{st} = 1$$

$$P_{K.T}^{st} \left(\frac{k_2}{k_1} + 1 \right) + P_{K.D}^{st} = 1 \tag{5}$$

From Equation 3:

$$P_{K.T}^{st} = \frac{(k_{3f} + k_{3b})}{k_2} \cdot P_{K.D}^{st} \tag{3*}$$

Putting Equation 3* into Equation 5, we get:

$$\frac{(k_{3f} + k_{3b})}{k_2} \cdot P_{K.D}^{st} \left(\frac{k_2}{k_1} + 1 \right) + P_{K.D}^{st} = 1$$

$$P_{K.D}^{st} \left[\left(\frac{(k_{3f} + k_{3b})}{k_2} \right) \left(\frac{k_2}{k_1} + 1 \right) + 1 \right] = 1 \tag{6}$$

From Equation 1:

$$P_{K.D}^{st} = \frac{k_1}{(k_{3f} + k_{3b})} P_K^{st} \tag{1*}$$

Putting Equation 1* into Equation 6, we get:

$$\frac{k_1}{(k_{3f}+k_{3b})}\cdot P_K^{st}\left[\left(\frac{(k_{3f}+k_{3b})}{k_2}\right)\left(\frac{k_2}{k_1}+1\right)+1\right]=1 \tag{7}$$

Simplifying Equation 7;

$$P_K^{st}=\frac{1}{\frac{k_1}{(k_{3f}+k_{3b})}\left[\left(\frac{(k_{3f}+k_{3b})}{k_2}\right)\left(\frac{k_2}{k_1}+1\right)+1\right]}$$

$$=\frac{1}{\frac{k_1}{(k_{3f}+k_{3b})}\left[\left(\frac{(k_{3f}+k_{3b})}{k_2}\right)\left(\frac{k_2+k_1}{k_1}\right)+1\right]}$$

$$=\frac{1}{\frac{k_1}{(k_{3f}+k_{3b})}\left[\frac{(k_{3f}+k_{3b})(k_2+k_1)+k_2k_1}{k_2k_1}\right]}$$

$$P_K^{st}=\frac{k_2(k_{3f}+k_{3b})}{(k_{3f}+k_{3b})(k_2+k_1)+k_2k_1} \tag{8}$$

Putting Equation 8 into Equation 1*:

$$P_{K.D}^{st}=\frac{k_1}{(k_{3f}+k_{3b})}\cdot\frac{k_2(k_{3f}+k_{3b})}{(k_{3f}+k_{3b})(k_2+k_1)+k_2k_1}=\frac{k_1k_2}{(k_{3f}+k_{3b})(k_2+k_1)+k_2k_1} \tag{9}$$

And putting Equation 9 into Equation 3*:

$$P_{K.T}^{st}=\frac{(k_{3f}+k_{3b})}{k_2}\cdot\frac{k_1k_2}{(k_{3f}+k_{3b})(k_2+k_1)+k_2k_1}=\frac{k_1(k_{3f}+k_{3b})}{(k_{3f}+k_{3b})(k_2+k_1)+k_2k_1} \tag{10}$$

Thus, the stationary probabilities are:

$$P_K^{st}=\frac{k_2(k_{3f}+k_{3b})}{(k_{3f}+k_{3b})(k_2+k_1)+k_2k_1}$$

$$P_{K.T}^{st}=\frac{k_1(k_{3f}+k_{3b})}{(k_{3f}+k_{3b})(k_2+k_1)+k_2k_1}$$

$$P_{K.D}^{st}=\frac{k_1k_2}{(k_{3f}+k_{3b})(k_2+k_1)+k_2k_1}$$

Mean velocity v of the motor by considering step size L

The mean velocity of the kinesin stepping can be expressed in terms k_1, k_2, k_{3f} and k_{3b} by considering the length of the step and the probability of forward or backward stepping. Thus the product of the step length and the net probability current from **K.D** to **K** (J_{net}).

The probability current from **K.D** to **K** is given as;

$$J^{st}_{K.D \to K} = W_{K,K.D} \cdot P^{st}_{K.D} - W_{K.D,K} \cdot P^{st}_{K} = (k_{3f} + k_{3b}) P^{st}_{K.D} - 0 \cdot P^{st}_{K} = \frac{k_1 k_2 (k_{3f} + k_{3b})}{(k_{3f} + k_{3b})(k_2 + k_1) + k_2 k_1}$$

It is important to note that the forward steps with transition rates (k_{3f}) have a positive effect on the velocity whiles the backward steps with transition rates (k_{3b}) have a negative effect on the velocity. Hence the net probability current is the value of the probability current for which the rates k_{3f} is positive and k_{3b} is negative.

Thus

$$J_{\text{net}} = W_{K,K.D} \cdot P^{st}_{K.D} - W_{K.D,K} \cdot P^{st}_{K} = (k_{3f} + (-k_{3b})) P^{st}_{K.D} - 0 \cdot P^{st}_{K} = \frac{(k_{3f} - k_{3b}) k_1 k_2}{(k_{3f} + k_{3b})(k_2 + k_1) + k_2 k_1}$$

Hence mean velocity is given as:

$$\begin{aligned} v &= L \times J_{\text{net}} \\ v &= \frac{L(k_{3f} - k_{3b}) k_1 k_2}{(k_{3f} + k_{3b})(k_2 + k_1) + k_2 k_1} \end{aligned} \tag{11}$$

Using the information from Table 1 in Ref. [1], the mean velocity v for force values $F = 0$ and $F = 3.8pN$ can be computed by using the respected values of k_1, k_2, k_{3f} and k_{3b} for each force in Equation 11.

It is important to note that the values of k_1 were given in $\mu M^{-1} s^{-1}$. Thus we have to multiply k_1 by the concentration of ATP either at saturating ($1mM = 10^3 \mu M^{-1}$) or at limiting ($10 \mu M^{-1}$) [1] to get it in s^{-1}. For this work, we will use the saturating concentration of ATP ($10^3 \mu M^{-1}$). This is because at saturating concentrations, the rate constants are not limited by ATP availability, providing a more accurate measure of the intrinsic properties of the kinesin's stepping mechanism.

The number of ATP molecules hydrolyzed per unit time r can also be computed by considering the probability that ATP molecules bind to the head that is attached to the microtubule (thus probability of moving to State **K.T**). This implies:

$$r = k_2 \cdot P^{st}_{K.T} = \frac{k_2 k_1 (k_{3f} + k_{3b})}{(k_{3f} + k_{3b})(k_2 + k_1) + k_2 k_1} \tag{12}$$

And the efficiency is given as:

$$\eta = \frac{v}{rL} \tag{13}$$

From Table 1 in Ref. [1],

- **F = 0** implies: $k_1 = 3.4\ \mu M^{-1}s^{-1}$; $k_2 = 140s^{-1}$; $k_{3f} = 770s^{-1}$; $k_{3b} = 3.5s^{-1}$
  $k_1 = 3.4\ \mu M^{-1}s^{-1} \times 10^3 \mu M^{-1} = 3400s^{-1}$

- **F = 3.8*pN*** implies: $k_1 = 1.4\ \mu M^{-1}s^{-1}$; $k_2 = 140s^{-1}$; $k_{3f} = 47s^{-1}$; $k_{3b} = 3.1s^{-1}$
  $k_1 = 1.4\ \mu M^{-1}s^{-1} \times 10^3 \mu M^{-1} = 1400s^{-1}$

Thus at saturating concentration,

i. **F = 0:**

$$\text{Mean velocity } v = \frac{8nm \times (770-3.5)s^{-1} \times 3400s^{-1} \times 140s^{-1}}{(770+3.5)s^{-1} \times (140+3400)s^{-1} + (140s^{-1} \times 3400s^{-1})} = 908.12\ nms^{-1}$$

$$r = \frac{140s^{-1} \times 3400s^{-1} \times (770+3.5)s^{-1}}{(770+3.5)s^{-1} \times (140+3400)s^{-1} + (140s^{-1} \times 3400s^{-1})} = 114.55\ s^{-1}$$

$$\eta = \frac{908.12\ nms^{-1}}{114.55\ s^{-1} \times 8nm} = 0.991$$

ii. **F = 3.8*pN*:**

$$\text{Mean velocity } v = \frac{8nm \times (47-3.1)s^{-1} \times 1400s^{-1} \times 140s^{-1}}{(47+3.1)s^{-1} \times (140+1400)s^{-1} + (140s^{-1} \times 1400s^{-1})} = 252.00\ nms^{-1}$$

$$r = \frac{140s^{-1} \times 1400s^{-1} \times (47+3.1)s^{-1}}{(47+3.1)s^{-1} \times (140+1400)s^{-1} + (140s^{-1} \times 1400s^{-1})} = 35.95\ s^{-1}$$

$$\eta = \frac{287.6\ nms^{-1}}{36.0\ s^{-1} \times 8nm} = 0.876$$

It can be observed that $F = 0$ is more efficient than $F = 3.8pN$.

Simulations

We started by defining a python function `gillespie_simulation` that simulates overtime the state evolution and respective position (forward or backward step) as shown in Listing 1. We ran the respective parameters for $F = 0$ and $F = 3.8pN$ on the Gillespie simulation (Listing 2).

- **Simulation Results for F = 0**
  Figure 3 shows the kinesin motor's state evolution and position trace over 5 seconds when no external force is applied. The top plot depicts the motor's state transitions over time, while the bottom plot illustrates the position trace of step size 8nm.

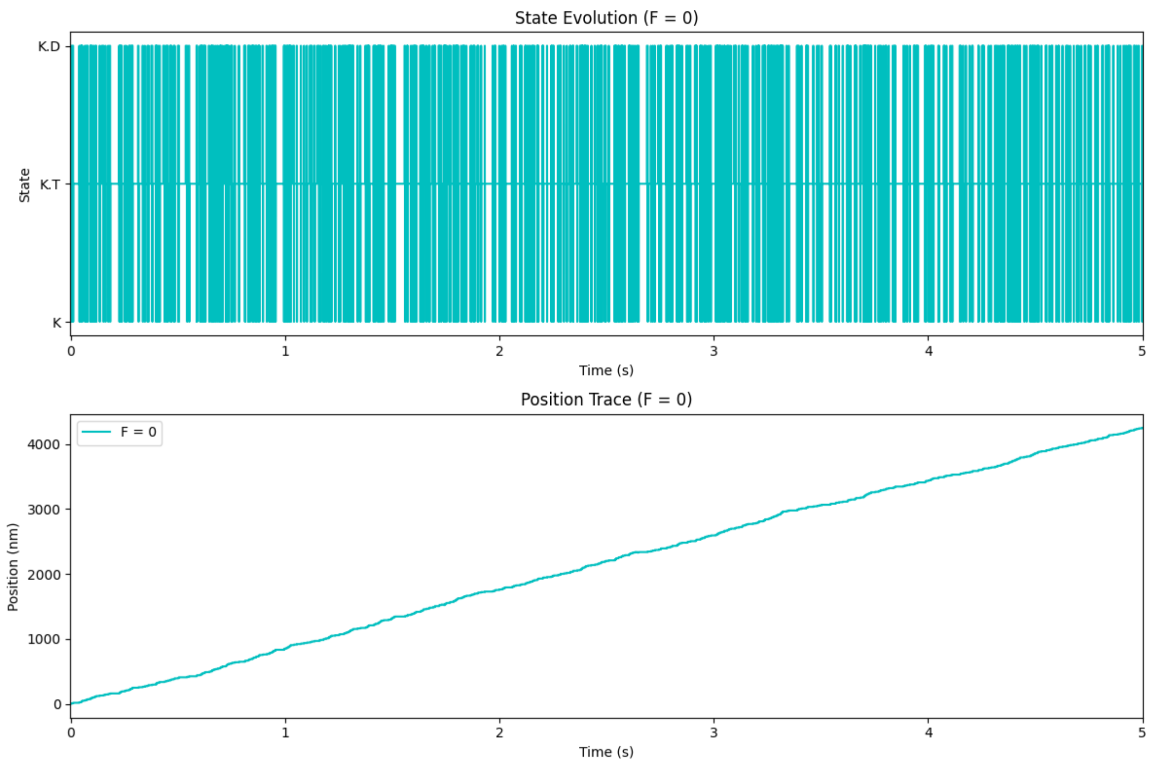

Figure 3: Simulation Results for F = 0. (Listing 3)

To provide a clearer view of the motor's initial dynamics, we zoomed in on the **first 0.1 seconds** as shown in Figure 4.

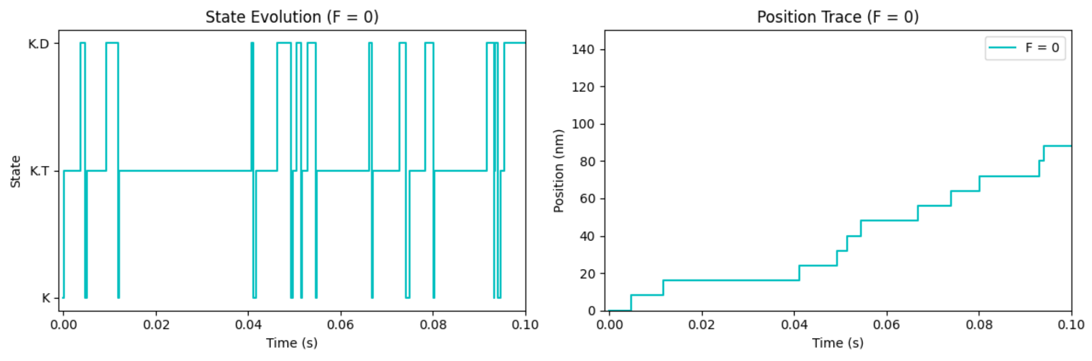

Figure 4: Zoomed Simulation Results for F = 0. (Listing 4,)

- **Simulation Results for F = 3.8pN**
  The results for the kinesin motor under a force of 3.8 pN is as shown in Figure 5 below.

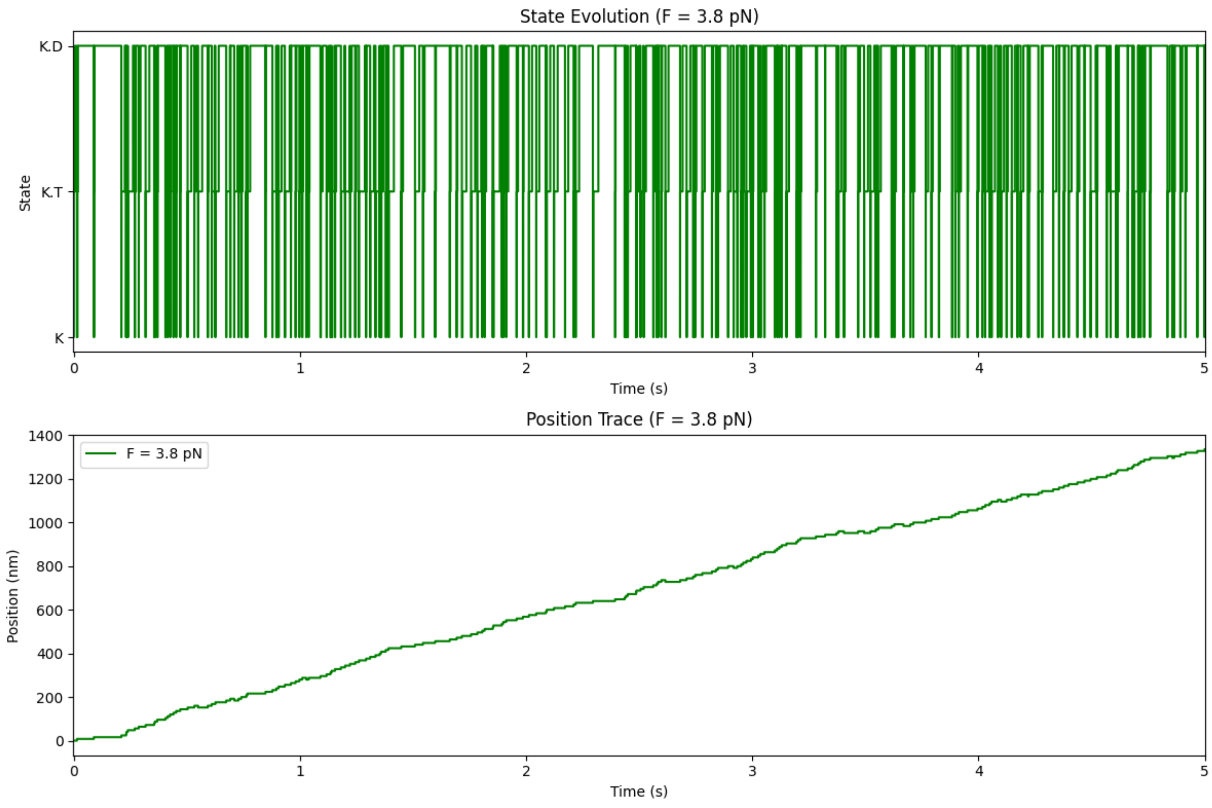

Figure 5: Simulation Results for F = 3.8pN (Listing 5,)

And the zoomed plot showing the **first 0.5 seconds** as shown in Figure 6

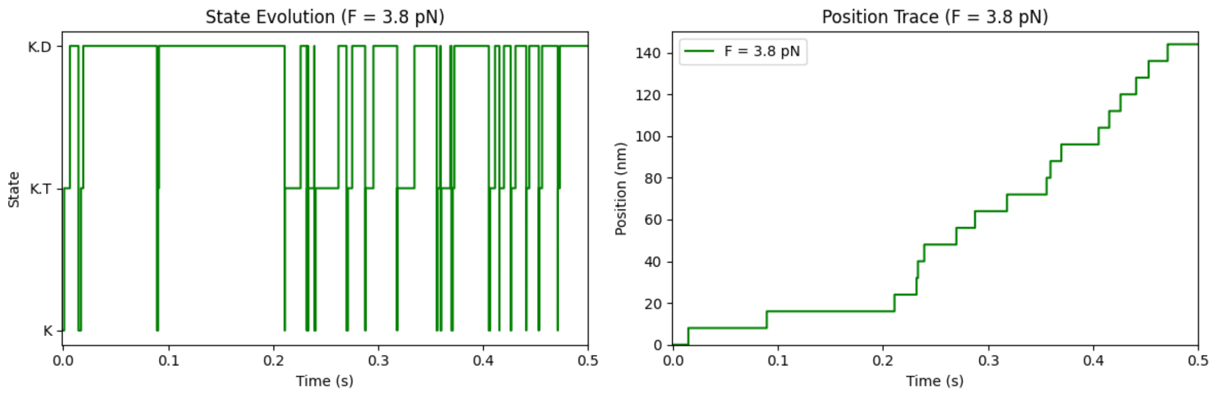

Figure 6: Zoomed Simulation Results for F = 3.8pN (Listing 6,)

From the plots, it can be seen that at $F = 0$, there are significantly more stepping than at $F = 3.8pN$

We defined a function `multi_simulations` in Python that performs multiple simulations of the Gillespie algorithm (Listing 7).

After performing 10^3 simulations of total duration 5s for both $F = 0$ and $F = 3.8pN$ (Listing 8), the mean velocity and the number of ATP molecules hydrolyzed per unit time from these simulations can be compared with the theoretical prediction from (1b.) as shown in Table 1.

	Theoretical Prediction		**Simulation** (10^3)	
	$F = 0$	$F = 3.8\,\mathbf{pN}$	$F = 0$	$F = 3.8\,\mathbf{pN}$
Mean velocity (nm/s)	908.12	252.00	908.60	252.47
ATP molecules hydrolyzed per unit time (1/s)	114.55	35.95	114.76	36.05

Table 1: Comparison of theoretical predictions and simulation results. (Listing 8,)

The waiting time in state **K.D** follows an exponential distribution due to the Markovian property. Its rate parameter is the sum of forward and backward rates from **K.D**.

Thus $\lambda_{K.D} = k_{3f} + k_{3b}$ and hence the probability density function of the waiting time is

$$\begin{aligned} f_{K.D}(t) &= \lambda_{K.D} e^{-\lambda_{K.D} t} \\ &= (k_{3f} + k_{3b}) e^{-(k_{3f}+k_{3b})t} \end{aligned} \tag{14}$$

For each case study, we will obtain the waiting time distribution in state K.D analytically and compare its value with the empirical waiting time distribution obtained from the simulations in (d).

- **CASE I : F = 0**

  We can obtain the waiting time distribution in state K.D analytically for $F = 0$ by substituting the respective values of k_{3f} and k_{3b} for $F = 0$ from Table 1 in Ref. [1].

  Thus for $F = 0$, $\lambda = 770 + 3.5 = 773.5$ and hence

$$f_{K.D}(t) = \lambda e^{-\lambda t} = 773.5e^{-773.5t}.$$

  In order to be able to compare with the empirical distribution, we wrote a python code to generate the analytic distribution (Listing 9). Also, we extracted the waiting times for K.D in F = 0 from the simulation by using the function `extract_waiting_times` (Listing 10). We plotted the histrogram for the extracted waiting times as shown in Figure 7.

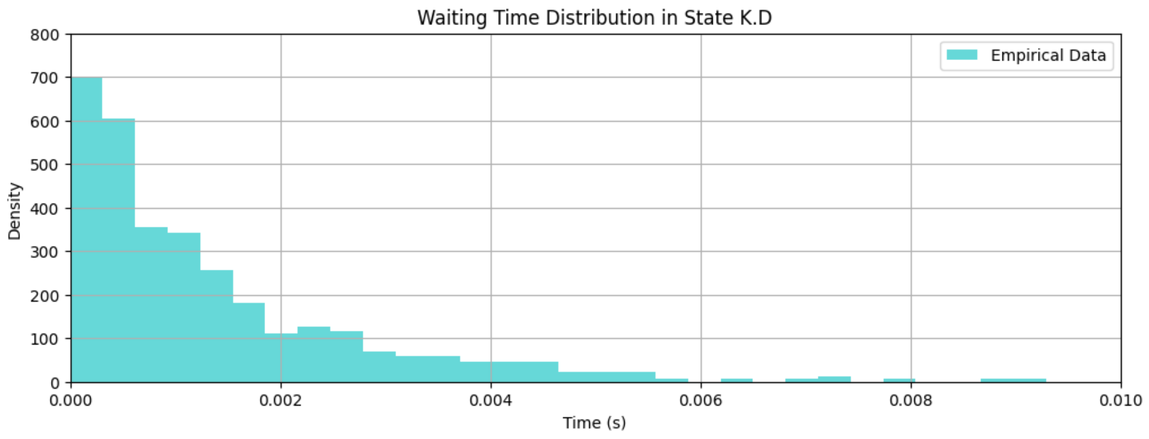

Figure 7: Histogram of empirical waiting times for F = 0. (Listing 11,)

  We then fitted the mean empirical waiting time and the resulting distribution was compared with the analytical distribution as shown in Figure 8. It is important to note that the empirical rate parameter $\bar{\lambda} = \dfrac{1}{\mu}$, where μ is the empirical mean.

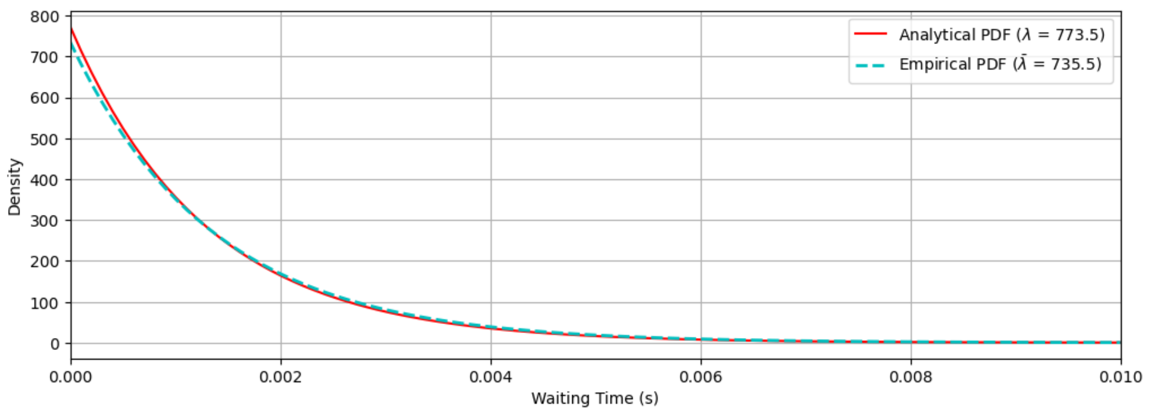

Figure 8: Waiting time distribution of analytical vs empirical for F = 0. (Listing 12,)

Hence the analytical pdf has $\lambda = 773.5$ whiles the empirical pdf, which is random has $\bar{\lambda} = 735.5$

- **CASE II : F = 3.8pN**
  Thus for $F = 3.8pN$, $\lambda = 47 + 3.1 = 50.1$ and hence

$$f_{K.D}(t) = \lambda e^{-\lambda t} = 50.1 e^{50.1t}.$$

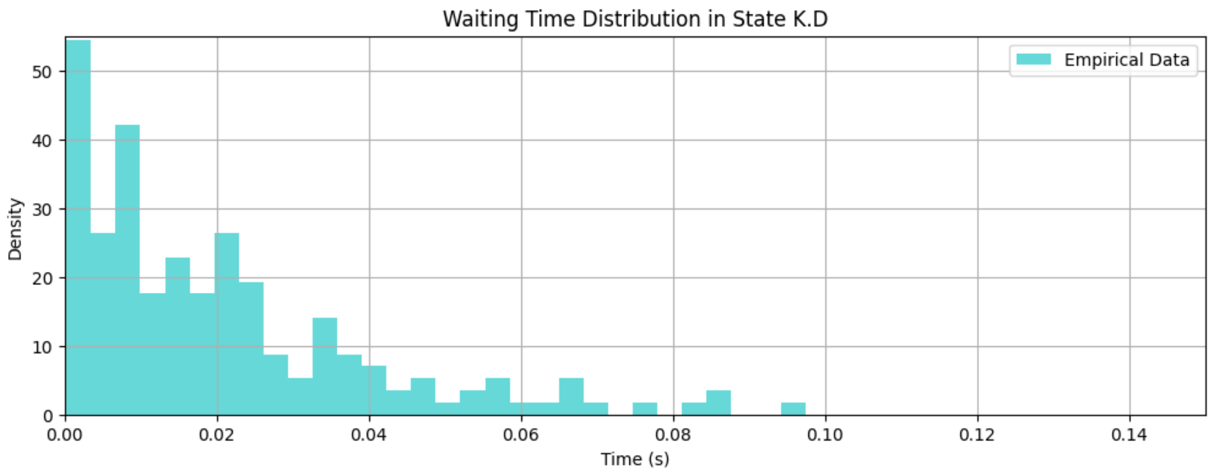

Figure 9: Histogram of empirical waiting times for F = 3.8pN (Listing 13,)

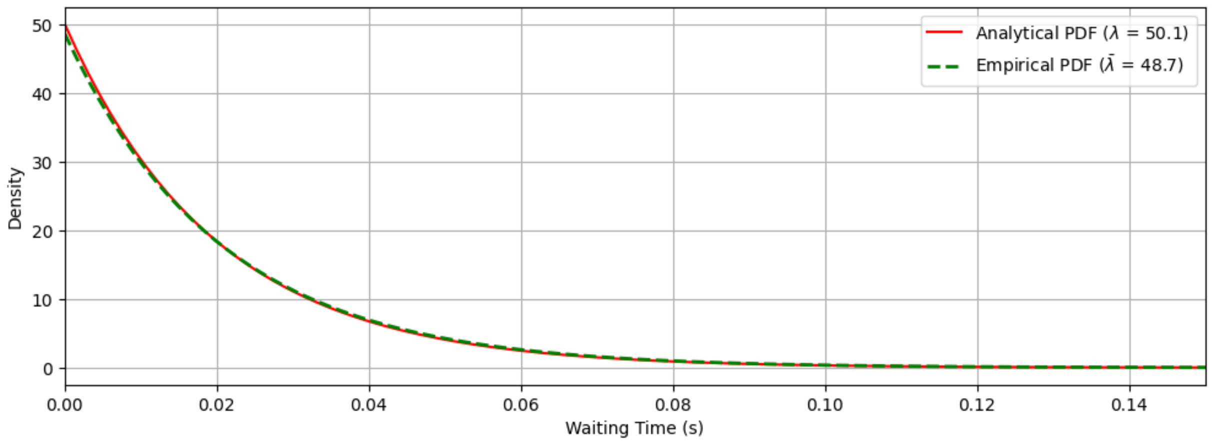

Figure 10: Waiting time distribution of analytical vs empirical for F = 3.8pN (Listing 14,)

Hence the analytical pdf has $\lambda = 50.1$ whiles the empirical pdf, which is random has $\bar{\lambda} = 48.7$

Waiting time of K.D for F = 0 decays exponentially faster than for F = 3.8pN.

References

[1] Masayoshi Nishiyama, Hideo Higuchi, and Toshio Yanagida. Chemomechanical coupling of the forward and backward steps of single kinesin molecules. Nature Cell Biology, 4(10):790–797, 2002.

[2] Andre C Barato and Udo Seifert. Thermodynamic uncertainty relation for biomolecular processes. Physical review letters, 114(15):158101, 2015.

Appendices

```python
import numpy as np
import matplotlib.pyplot as plt

#Gillespie simulation function
def gillespie_simulation(k1, k2, k3f, k3b, total_time, L):
    state = 'K' #Initial state
    time = 0 #Start time
    times = [0] #Record times
    states = [state] #Record states
    position = [0] #Record position

    while time < total_time:
      #Define transition rates based on current state
      if state == 'K':
        rates = [k1]
      elif state == 'K.T':
        rates = [k2]
      elif state == 'K.D':
        rates = [k3f, k3b]

      #Total rate and time until the next event
      total_rate = sum(rates)
      if total_rate == 0:
        break

      dt = np.random.exponential(1 / total_rate) #Total rate for individual
         state.
      time += dt

      #Determine the event
      r = np.random.rand() * total_rate #Random number between 0 and
         total_rate
      cumulative_rate = 0
      for i, rate in enumerate(rates):
        cumulative_rate += rate
        if r < cumulative_rate:
          event = i #event will be 0 for backward (k3b) or 1 for forward (
             k3f)
          break

      #Update state and position based on the event
      if state == 'K' and event == 0:
        state = 'K.T'
      elif state == 'K.T' and event == 0:
        state = 'K.D'
      elif state == 'K.D':
        if event == 0:
          state = 'K'
          position.append(position[-1] + L) #Forward step
        elif event == 1:
          state = 'K'
          position.append(position[-1] - L) #Backward step

      #Record the state and time
      times.append(time)
      states.append(state)

      #Fill in position with initial position if no step occurred
      if len(position) < len(times):
        position.append(position[-1])

  return np.array(times), np.array(states), np.array(position)
```

Listing 1: Gillespie algorithm for evolution of states and position of kinesin

```
#Parameters for the simulation
L = 8  #step size in nm
total_time = 5  #total simulation time in seconds
  #Saturated concentration parameters
k1_F0, k2_F0, k3f_F0, k3b_F0 = 3400, 140, 770, 3.5 #Transition rates for F=0
k1_F38, k2_F38, k3f_F38, k3b_F38 = 1400, 140, 47, 3.1 #Transition rates for F = 3.8 pN

# Simulate for F = 0
times_F0, states_F0, position_F0 = gillespie_simulation(k1_F0, k2_F0, k3f_F0, k3b_F0, total_time, L)

# Simulate for F = 3.8 pN
times_F38, states_F38, position_F38 = gillespie_simulation(k1_F38, k2_F38, k3f_F38, k3b_F38, total_time, L)
```

Listing 2: Parameter definition and simulation

```
# Plotting results for F = 0
fig, axes = plt.subplots(2, 1, figsize=(12, 8))

# State evolution
axes[0].plot(times_F0, states_F0,'c', drawstyle='steps-post', label='F␣=␣0')
axes[0].set_title('State␣Evolution␣(F␣=␣0)')
axes[0].set_xlabel('Time␣(s)')
axes[0].set_ylabel('State')
axes[0].set_xlim(-0.001, total_time)

# Position trace
axes[1].plot(times_F0,position_F0,'c', drawstyle='steps-post',label='F␣=␣0')
axes[1].set_title('Position␣Trace␣(F␣=␣0)')
axes[1].set_xlabel('Time␣(s)')
axes[1].set_ylabel('Position␣(nm)')
axes[1].set_xlim(-0.001, total_time)

plt.tight_layout()
plt.legend()
plt.show()
```

Listing 3: Plot of simulation results for `F = 0`

```python
# Plotting results for F = 0
fig, axes = plt.subplots(1, 2, figsize=(12, 4))

# State evolution
axes[0].plot(times_F0, states_F0,'c', drawstyle='steps-post', label='F␣=␣0')
axes[0].set_title('State␣Evolution␣(F␣=␣0)')
axes[0].set_xlabel('Time␣(s)')
axes[0].set_ylabel('State')
axes[0].set_xlim(-0.001, 0.1)

# Position trace
axes[1].plot(times_F0, position_F0,'c',drawstyle='steps-post',label='F␣=␣0')
axes[1].set_title('Position␣Trace␣(F␣=␣0)')
axes[1].set_xlabel('Time␣(s)')
axes[1].set_ylabel('Position␣(nm)')
axes[1].set_xlim(-0.001, 0.1)
axes[1].set_ylim(0, 150)

plt.tight_layout()
plt.legend()
plt.show()
```

Listing 4: Zoomed plot of simulation results for `F = 0`

```python
# Plotting results for F = 3.8 pN
fig, axes = plt.subplots(2, 1, figsize=(12, 8))

# State evolution
axes[0].plot(times_F38, states_F38,'g', drawstyle='steps-post', label='F␣=␣3.8␣pN')
axes[0].set_title('State␣Evolution␣(F␣=␣3.8␣pN)')
axes[0].set_xlabel('Time␣(s)')
axes[0].set_ylabel('State')
axes[0].set_xlim(-0.001, total_time)

# Position trace
axes[1].plot(times_F38, position_F38,'g', drawstyle='steps-post', label='F␣=␣3.8␣pN')
axes[1].set_title('Position␣Trace␣(F␣=␣3.8␣pN)')
axes[1].set_xlabel('Time␣(s)')
axes[1].set_ylabel('Position␣(nm)')
axes[1].set_xlim(-0.001, total_time)

plt.tight_layout()
plt.legend()
plt.show()
```

Listing 5: Plot of simulation results for `F = 3.8 pN`

```python
# Plotting results for F = 3.8 pN
fig, axes = plt.subplots(1, 2, figsize=(12, 4))

# State evolution
axes[0].plot(times_F38, states_F38,'g', drawstyle='steps-post', label='F = 3.8 pN')  # Use descriptive states directly
axes[0].set_title('State Evolution (F = 3.8 pN)')
axes[0].set_xlabel('Time (s)')
axes[0].set_ylabel('State')
axes[0].set_xlim(-0.001, 0.5)

# Position trace
axes[1].plot(times_F38, position_F38,'g', drawstyle='steps-post', label='F = 3.8 pN')
axes[1].set_title('Position Trace (F = 3.8 pN)')
axes[1].set_xlabel('Time (s)')
axes[1].set_ylabel('Position (nm)')
axes[1].set_xlim(-0.001, 0.5)
axes[1].set_ylim(0, 150)

plt.tight_layout()
plt.legend()
plt.show()
```

Listing 6: Zoomed plot of simulation results for `F = 3.8 pN`

```python
#Multiple simulations for a given force
def multi_simulations(k1, k2, k3f, k3b, total_time, L, num_simulations):
    velocities = []  #store velocities (nm/s)
    atp_hydrolysis_rates = []  #store ATP hydrolysis rates (1/s)

    for _ in range(num_simulations):
        #Run a Gillespie simulation
        times, states, positions = gillespie_simulation(k1, k2, k3f, k3b,
            total_time, L)

        #Compute velocity (total displacement / total time)
        displacement = positions[-1] - positions[0]  #total displacement
        velocity = displacement / total_time
        velocities.append(velocity)

        #Compute ATP hydrolysis rate (number of transitions KT -> KD per
            second)
        atp_hydrolysis = sum(1 for i in range(len(states) - 1) if states[i]
            == 'K.T' and states[i + 1] == 'K.D')
        atp_hydrolysis_rate = atp_hydrolysis / total_time
        atp_hydrolysis_rates.append(atp_hydrolysis_rate)

    return np.mean(velocities), np.mean(atp_hydrolysis_rates)
```

Listing 7: Multiple simulation function for Gillespie algorithm

```python
#Number of simulations
num_simulations = 1000

#Compute for F = 0
mean_velocity_F0, mean_atp_rate_F0 = multi_simulations(k1_F0, k2_F0, k3f_F0, k3b_F0, total_time, L, num_simulations)

#Compute for F = 3.8 pN
mean_velocity_F38, mean_atp_rate_F38 = multi_simulations(k1_F38, k2_F38, k3f_F38, k3b_F38, total_time, L, num_simulations)

#Print results
print(f'F␣=␣0␣pN:␣Mean␣velocity␣=␣{mean_velocity_F0:.2f}␣nm/s,␣Mean␣ATP␣hydrolysis␣rate␣=␣{mean_atp_rate_F0:.2f}␣1/s')
print(f'F␣=␣3.8␣pN:␣Mean␣velocity␣=␣{mean_velocity_F38:.2f}␣nm/s,␣Mean␣ATP␣hydrolysis␣rate␣=␣{mean_atp_rate_F38:.2f}␣1/s')
```

Listing 8: Mean velocity and mean ATP hydrolysis from multiple simulations

```python
import numpy as np
import matplotlib.pyplot as plt

#Parameters for F = 0
k3f_F0 = 770
k3b_F0 = 3.5
lambda_KD_F0 = k3f_F0 + k3b_F0

#Parameters for F = 3.8
k3f_F38 = 47
k3b_F38 = 3.1
lambda_KD_F38 = k3f_F38 + k3b_F38

#Generate analytical distribution F = 0.
t_values_F0 = np.linspace(0, 0.01, 100)
f_KD_F0 = lambda_KD_F0 * np.exp(-lambda_KD_F0 * t_values_F0)

#Generate analytical distribution F = 3.8pN.
t_values_F38 = np.linspace(0, 0.15, 100)
f_KD_F38 = lambda_KD_F38 * np.exp(-lambda_KD_F38 * t_values_F38)
```

Listing 9: Generation of analytical distribution

```python
def extract_waiting_times(times, states, target_state):
    waiting_times = [] #record waiting times
    in_target_state = False
    start_time = 0

    for i in range(len(states)):
        if states[i] == target_state:
            if not in_target_state:
                #Entering the target state
                start_time = times[i]
                in_target_state = True
        else:
            if in_target_state:
                #Exiting the target state
                waiting_time = times[i] - start_time
                waiting_times.append(waiting_time)
                in_target_state = False

    return waiting_times
```

Listing 10: Extract waiting times function

```python
# Extract waiting times for K.D in F = 0 simulation
waiting_times_KD_F0 = extract_waiting_times(times_F0, states_F0,
    target_state='K.D')

#Plot the analytical distribution
plt.figure(figsize=(12, 4))
# Plot histogram of empirical data
plt.hist(waiting_times_KD_F0, bins=30, density=True, alpha=0.6, color='c',
    label='Empirical␣Data')
plt.title('Waiting␣Time␣Distribution␣in␣State␣K.D')
plt.xlabel('Time␣(s)')
plt.ylabel('Density')
plt.xlim(0,0.01)
plt.ylim(0,800)
plt.legend()
plt.grid(True)

# Show the plot
plt.show()
```

Listing 11: Histogram of empirical waiting times for `F = 0`

```python
#Compute empirical mean
mean_waiting_time_F0 = np.mean(waiting_times_KD_F0)
print(f'Mean␣waiting␣time␣in␣K.D␣(F␣=␣0):␣{mean_waiting_time_F0:.4f}␣s')

#Fit an exponential distribution to the emperical mean
from scipy.stats import expon
emperical_F0 = (1 / mean_waiting_time_F0) * np.exp(- t_values_F0 /
    mean_waiting_time_F0)

#Plot the analytical distribution
plt.figure(figsize=(12, 4))
plt.plot(t_values_F0, f_KD_F0, 'r-', label=f'Analytical␣PDF␣($\lambda$␣=␣{
    lambda_KD_F0:.1f})')

#Plot the fitted exponential PDF for Empirical
plt.plot(t_values_F0, emperical_F0, 'c', linewidth=2, label=f'Empirical␣PDF␣
    ($\\bar{{\\lambda}}$␣=␣{1/mean_waiting_time_F0:.1f})',linestyle='dashed')
plt.xlabel('Waiting␣Time␣(s)')
plt.ylabel('Density')
plt.xlim(0, 0.01)
plt.grid(True)

plt.legend()
plt.show()
```

Listing 12: Waiting time distribution in state K.D for `F = 0`

```python
# Extract waiting times for K.D in F = 3.8pN simulation
waiting_times_KD_F38 = extract_waiting_times(times_F38, states_F38,
    target_state='K.D')

#Plot the analytical distribution
plt.figure(figsize=(12, 4))
# Plot histogram of empirical data
plt.hist(waiting_times_KD_F38, bins=30, density=True, alpha=0.6, color='c',
    label='Empirical␣Data')
plt.title('Waiting␣Time␣Distribution␣in␣State␣K.D')
plt.xlabel('Time␣(s)')
plt.ylabel('Density')
plt.xlim(0,0.15)
plt.ylim(0,55)
plt.legend()
plt.grid(True)

# Show the plot
plt.show()
```

Listing 13: Histogram of empirical waiting times for `F = 3.8 pN`

```python
#Compute empirical mean
mean_waiting_time_F38 = np.mean(waiting_times_KD_F38)
print(f'Mean␣waiting␣time␣in␣K.D␣(F␣=␣38):␣{mean_waiting_time_F38:.4f}␣s')

#Fit an exponential distribution to the empirical mean
from scipy.stats import expon
emperical_F38 = (1 / mean_waiting_time_F38) * np.exp(- t_values_F38 / mean_waiting_time_F38)

#Plot the analytical distribution
plt.figure(figsize=(12, 4))
plt.plot(t_values_F38, f_KD_F38, 'r-', label=f'Analytical␣PDF␣($\lambda$␣=␣{lambda_KD_F38:.1f})')

#Plot the fitted exponential PDF for Empirical
plt.plot(t_values_F38, emperical_F38, 'g', linewidth=2, label=f'Empirical␣PDF␣($\\bar{{\\lambda}}$␣=␣{1/mean_waiting_time_F38:.1f})',linestyle='dashed')
plt.xlabel('Waiting␣Time␣(s)')
plt.ylabel('Density')
plt.xlim(0, 0.15)
plt.grid(True)

plt.legend()
plt.show()
```

Listing 14: Waiting time distribution in state K.D for `F = 3.8 pN`