FPE Simulations of Polymer Stretching in Batchelor-Kraichnan Flows

CL677: Modelling Stochastic and Turbulent Transport

Course Project Report

Members

Gaurav Awasthi: 200020036

Gautam Prabhudesai: 200020055

Shreya Makkar: 200020133

Guide: Prof. Jason Picardo

Department of Chemical Engineering

Indian Institute of Technology Bombay

10 May 2024

Table of Contents

Introduction

The behavior of polymers within a random smooth flow is explored using the Hookean dumbbell model. In its coiled state, the probability density function (pdf) of the polymer reaches a stationary state characterized by power-law tails, both for small and large arguments compared to the equilibrium length[1] .At equilibrium, a polymer molecule assumes a compact, spherical shape with a radius denoted as R_0. However, when subjected to a non-homogeneous flow, the polymer experiences deformation and stretching, causing its end-to-end distance, denoted as R, to potentially exceed the equilibrium size R_0 [1].

Theoretical and numerical investigations indicate that across a broad spectrum of scales, the polymer's elasticity can be effectively described by Hooke's law [3]. Within random flows, the ratio of the relaxation time to the characteristic stretching time is quantified by the Weissenberg number, denoted as Wi (where $Wi \equiv \lambda_T$) [1]. When Wi < 1, contraction occurs rapidly, leading the molecule to converge towards its equilibrium size. In this scenario, the polymer remains in its coiled state. Conversely, when Wi > 1, the elastic forces are relatively weak, allowing velocity gradients within the flow to significantly deform the molecule[1]. In such instances, the dynamics of the polymer are markedly influenced by the properties of the advecting flow.

In turbulent flow polymers are under fluctuating strain due to the flow and undergo mechanical degradation which results in their scission. This has a major impact on its applications as drag reducing agents. Polymers experience huge straining in large reynolds number flow which can distort them. This phenomenon has been confirmed in experiments and numerical simulations. In experiments it is observed that bulk of the polymers lie in the bulk of the fluid where the flow is approximately isotropic turbulent. Even in low reynolds number turbulent flow due to chaotic flow the polymers are stretched. The distribution of the polymer extension q has a wide power law core due to fluctuating strains[2]. The Rouse model where the polymer is modeled as a bead spring system is one of the most common representations of polymers in flow. We can introduce scission into this by assuming that the polymers will break at a certain percent of the maximum extension.

Theory

We can write the SDE for the extension of polymer in turbulent flow as follows:

$$dq_i = \frac{-f(q)q_i}{2\tau}dt + \sqrt{\frac{q_{eq}^2}{3\tau}} \circ dW_i + q_j \circ d\Gamma_{ij}$$

Where the last term is the one which incorporates the extension due to flow stretching. Here the torsional noise has a correlational matrix $\langle d\Gamma_{kl}\, d\Gamma_{mn}\rangle = K^{klmn}\delta(t-t')$.

We approximate our turbulent flow using the Batchelor-Kraichnan model. So we have

$$K^{klab} = \frac{\lambda}{3}[4\delta_{ka}\delta_{lb} - \delta_{kl}\delta_{ab} - \delta_{kb}\delta_{al}]$$

Corresponding to this we get our Fokker-Planck equation for the dumbbell in the phase space as

$$\frac{\partial p}{\partial t} = \frac{\partial}{\partial q_i}\left(\frac{f(q)}{2\tau}q_i p\right) + \frac{1}{2}\left(\frac{q_{eq}^2}{3\tau}\right)\frac{\partial^2 p}{\partial q_i^2} + \frac{\lambda}{6}\left[4\frac{\partial^2(pq^2)}{\partial q_i^2} - 2\frac{\partial^2}{\partial q_i \partial q_m}(q_i q_m p)\right]$$

This equation is a 3 dimensional diffusion equation. Instead we solve the Fokker-Planck equation for the magnitude of the elongation $q = (q_i q_i)^{1/2}$.

The probability density function of the elongation, P(q,t), obeys the Fokker–Planck equation associated with Equation:

$$\frac{\partial P(q,t)}{\partial t} = -\frac{\partial(D_1 P(q,t))}{\partial q} + \frac{\partial^2(D_2 P(q,t))}{\partial q^2}$$

where,

$$D_1 = \left(\frac{8\,Wi}{3} - f(q)\right)q + \frac{2}{3}\frac{q_{eq}^2}{q} \text{ and } D_2 = \frac{q_{eq}^2}{3} + \frac{2}{3}W_i q^2$$

For the case with no scission we search for a solution such that the associated probability current vanishes both in zero and at infinity. These conditions (called reflecting boundary conditions) correspond to requiring that there is no flow of probability through the boundaries of the domain of definition. So our boundary condition in the case of no scission would be

$D_1 P(q,t) + \frac{\partial}{\partial q}(D_2 P(q,t)) = 0$ at q = 0 and q_{max}.

On the other hand in case of scission we assume that the polymers break when they stretch to 80% of the maximum extension. So we would have the Dirichlet boundary condition at the breaking point. So our boundary condition in the case of scission would be

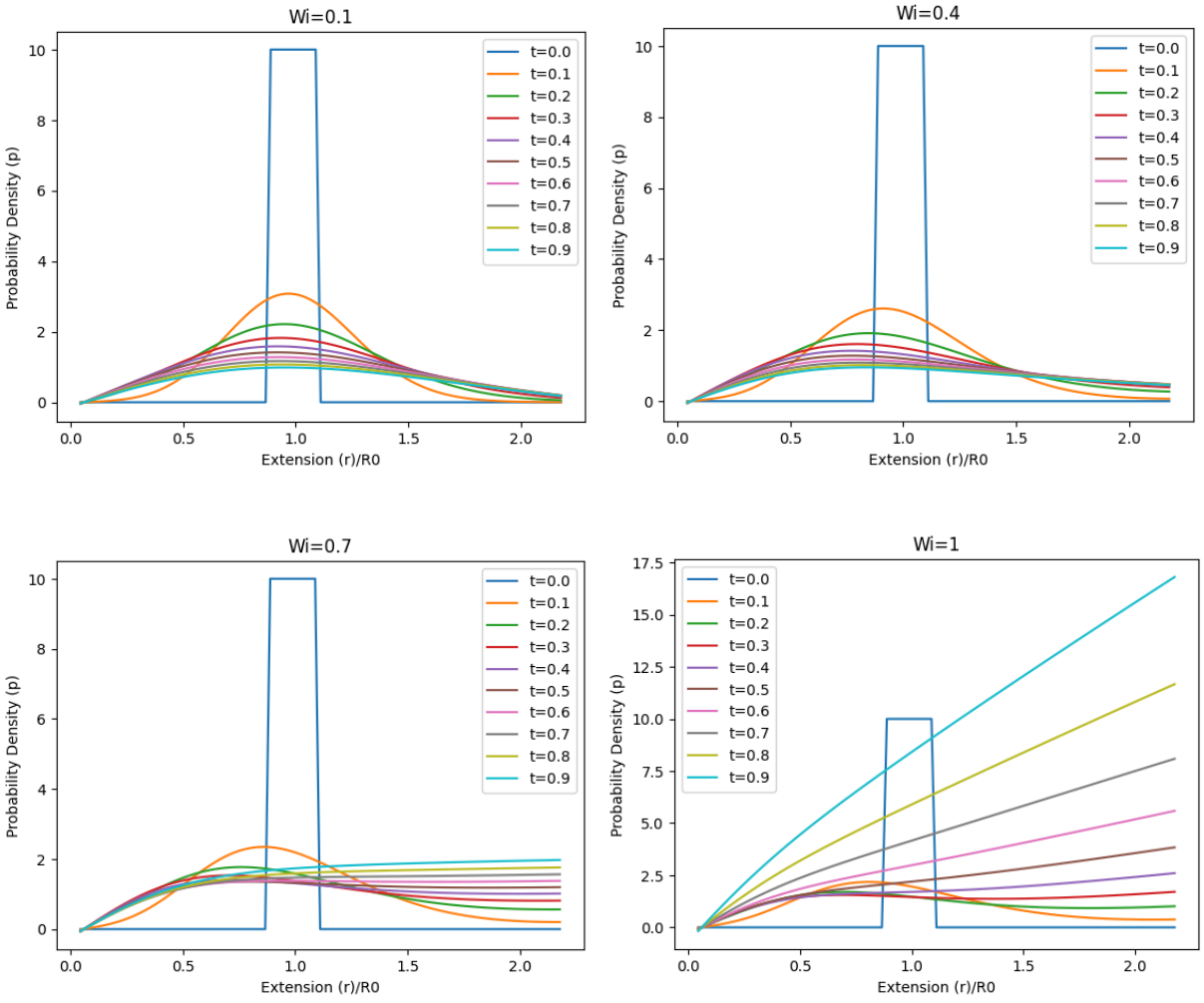
$D_1 P(q,t) + \frac{\partial}{\partial q}(D_2 P(q,t)) = 0$ at q=0 and $P(0.8q_{max}, t) = 0$ at the breaking point.

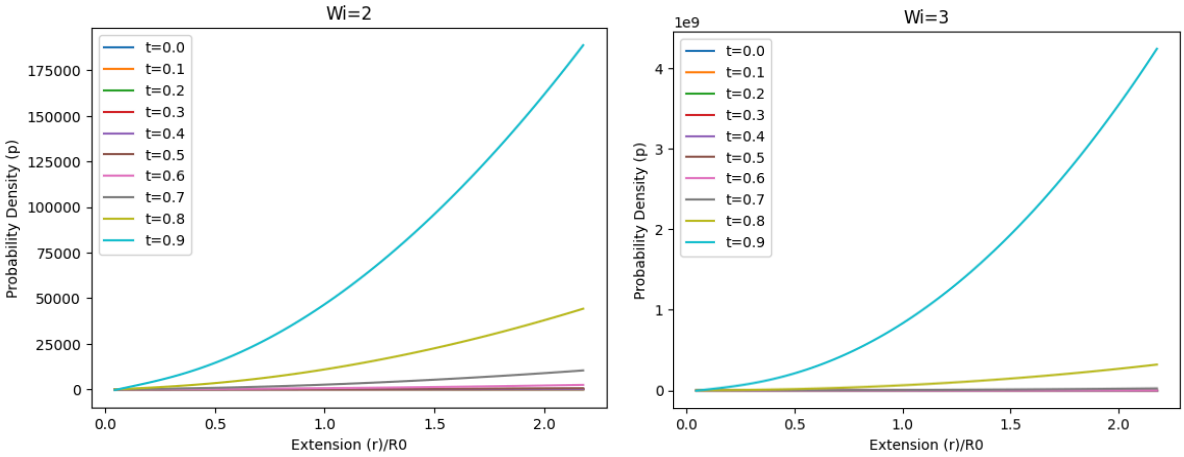
We can solve these equations numerically by using the central difference formula for the derivatives in q and then solving the set of ODE.

Results & Discussion

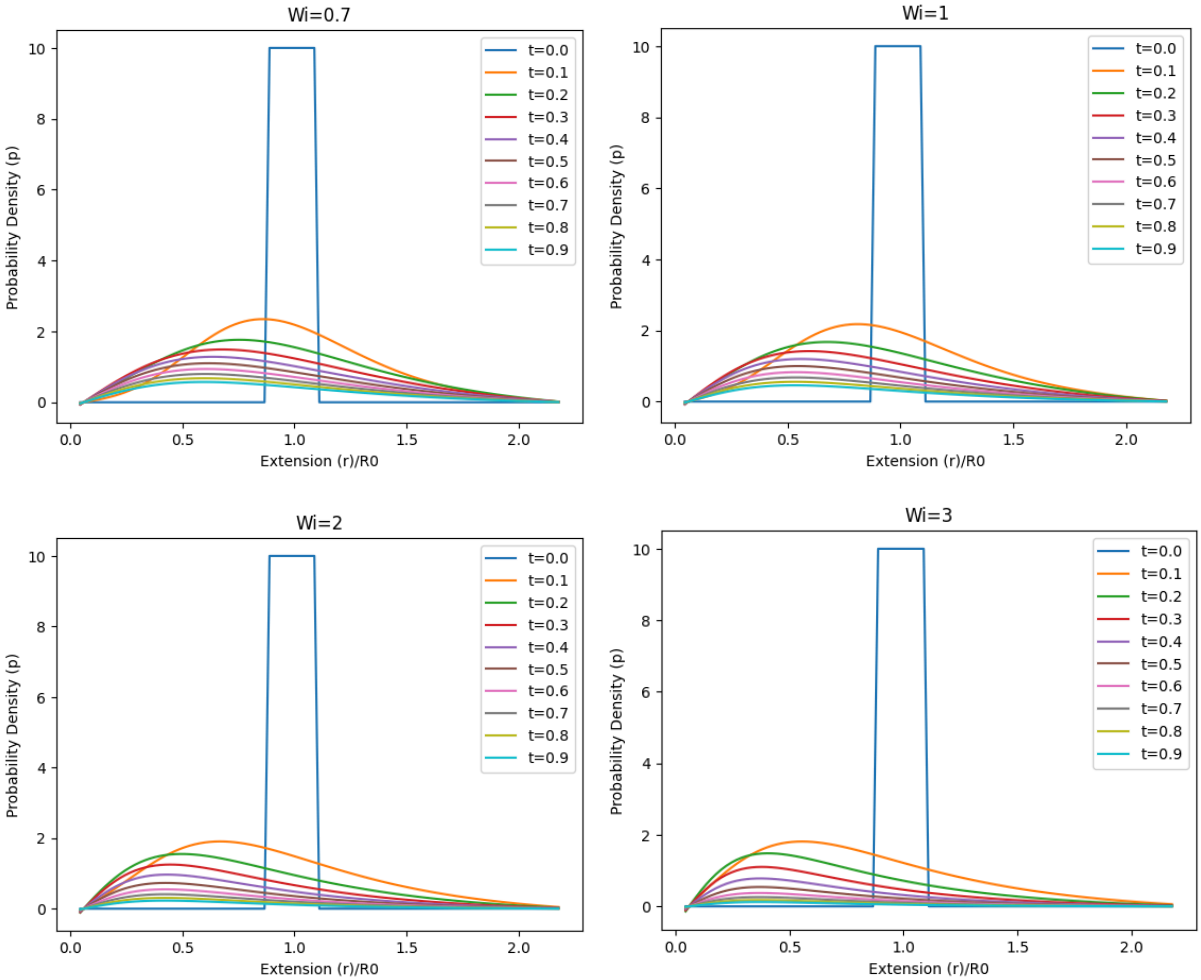
We choose r_{min} = 0.02, r_{max} = 1, and r_{eq} = 0.45. The latter value is actually calculated based on the elastic energy and the temperature, but we assume a hypothetical value since the characteristics of the polymer are unknown. Here, we take $f(q) = 1$.

1. FPE simulation for the evolution of the pdf p(r,t) with no-flux boundary conditions at zero and maximum extension:



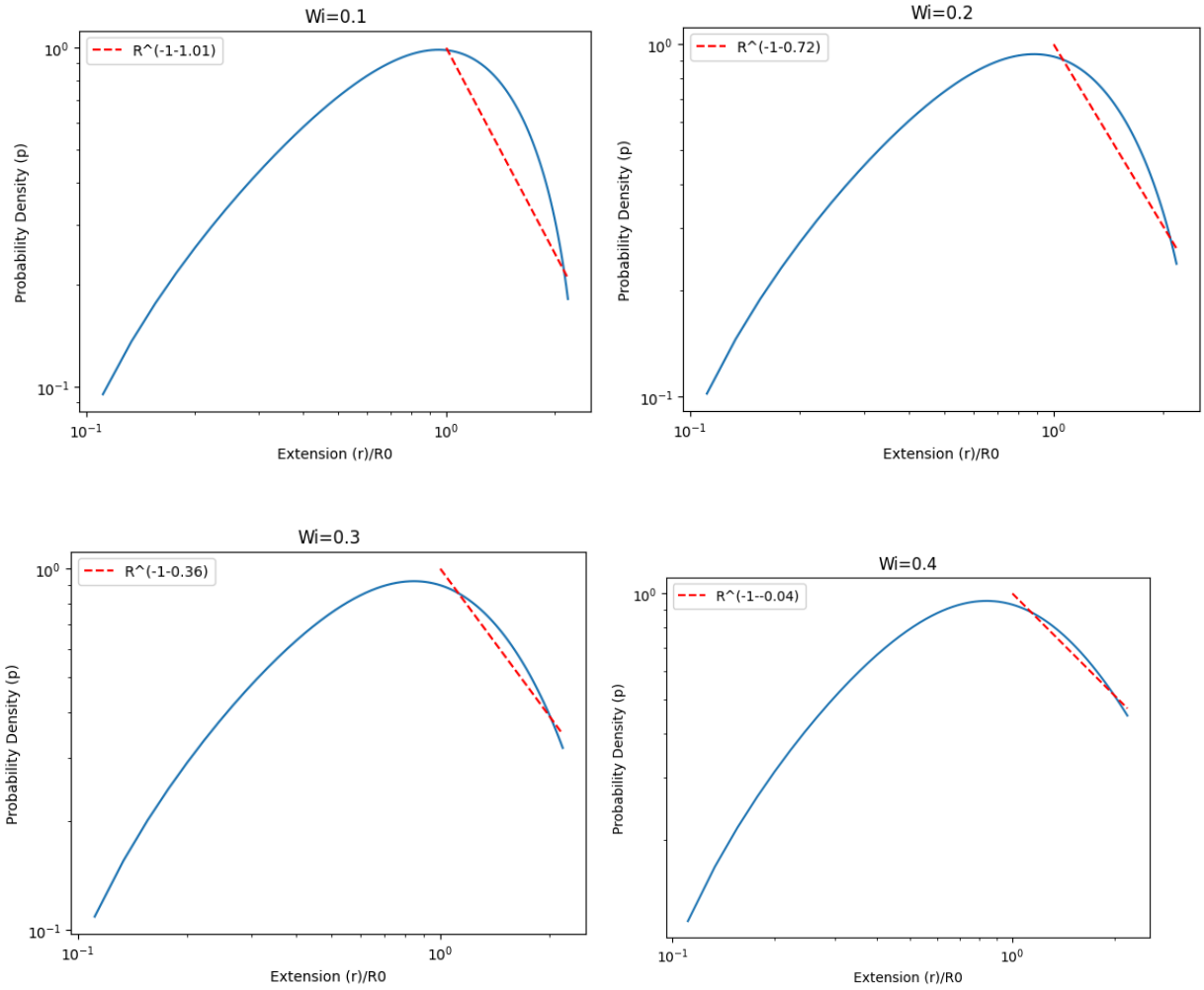


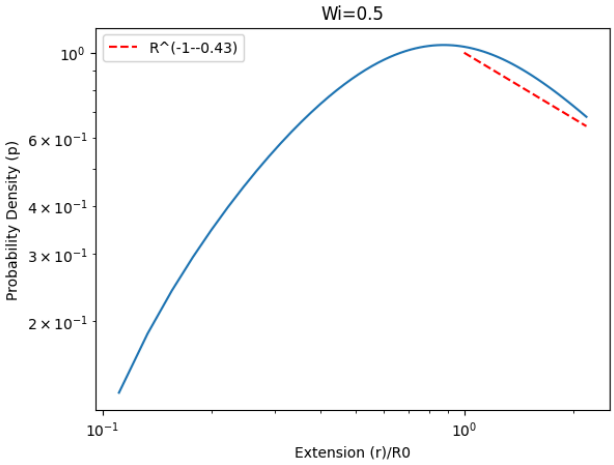
We notice that the distributions start diverging for higher values of Wi. On taking $f(q) = \frac{1}{1-(\frac{q}{q_{max}})^2}$, this is avoided.

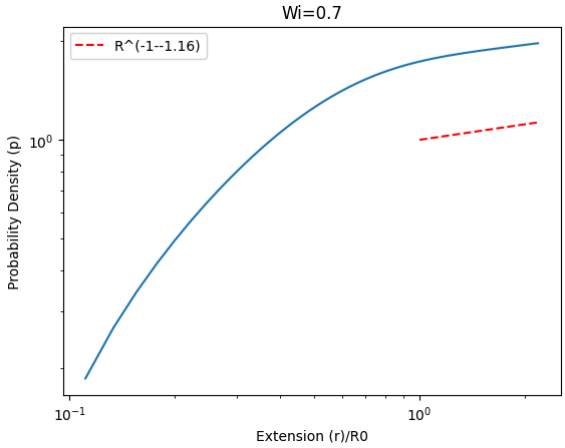


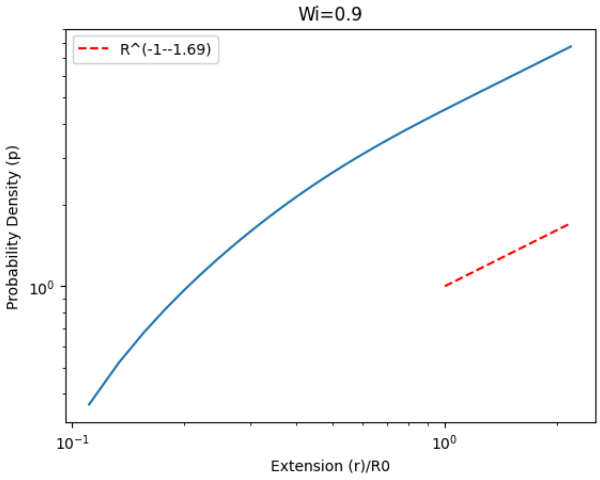
Power law exponent for varying Wi:

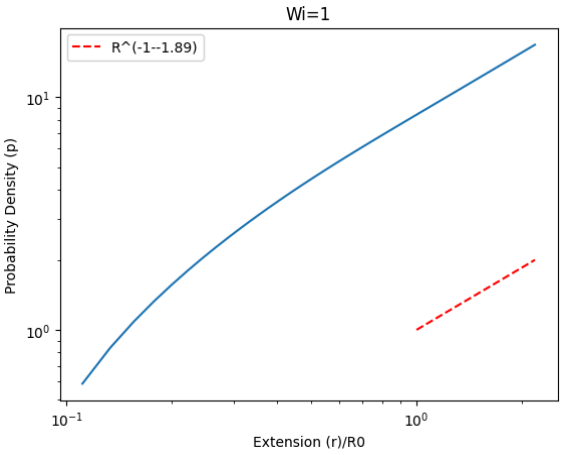
The tail end of the probability distribution (beyond $R=R_{eq}$) is known to decay as $R^{-1-\alpha}$, where α forms the power law exponent. The plots below show α as a function of Wi.

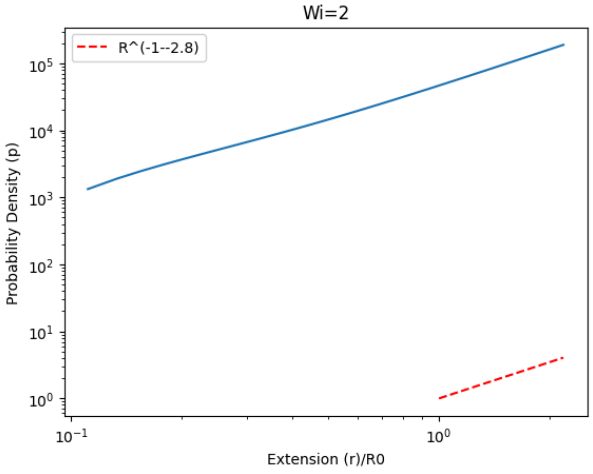


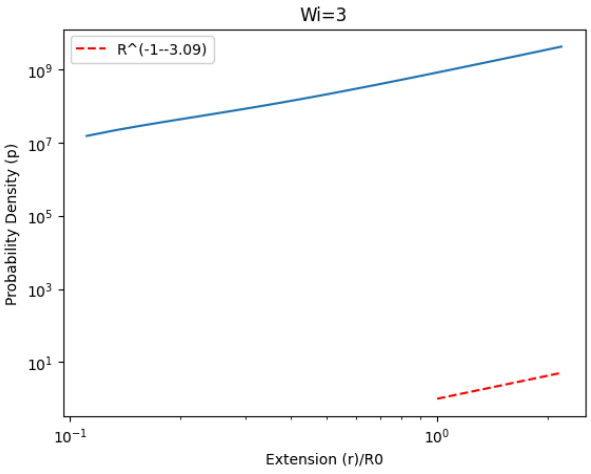

Wi=0.5
R^(-1--0.43)
Probability Density (p)
Extension (r)/R0

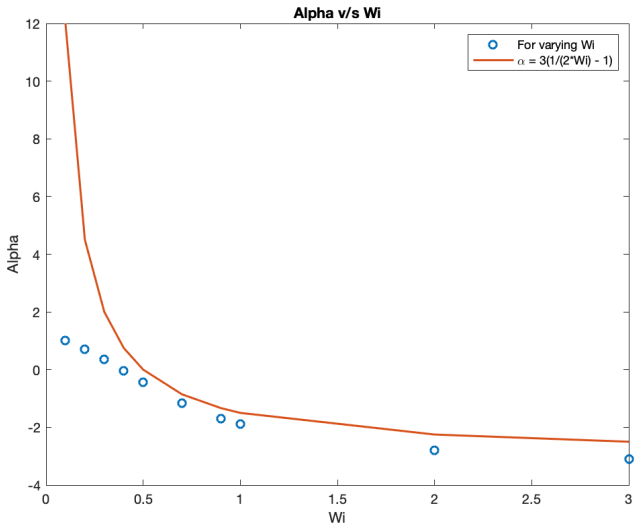

Wi=0.7
R^(-1--1.16)
Probability Density (p)
Extension (r)/R0


Wi=0.9
R^(-1--1.69)
Probability Density (p)
Extension (r)/R0

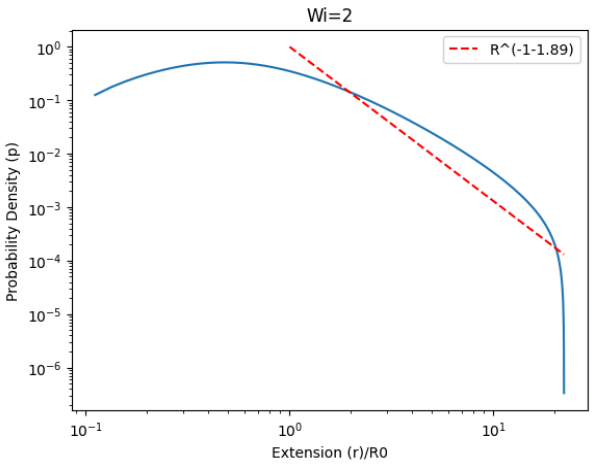

Wi=1
R^(-1--1.89)
Probability Density (p)
Extension (r)/R0

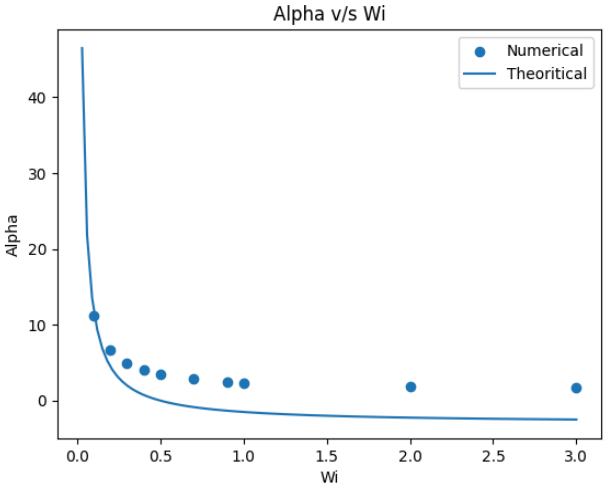

Wi=2
R^(-1--2.8)
Probability Density (p)
Extension (r)/R0


Wi=3
R^(-1--3.09)
Probability Density (p)
Extension (r)/R0



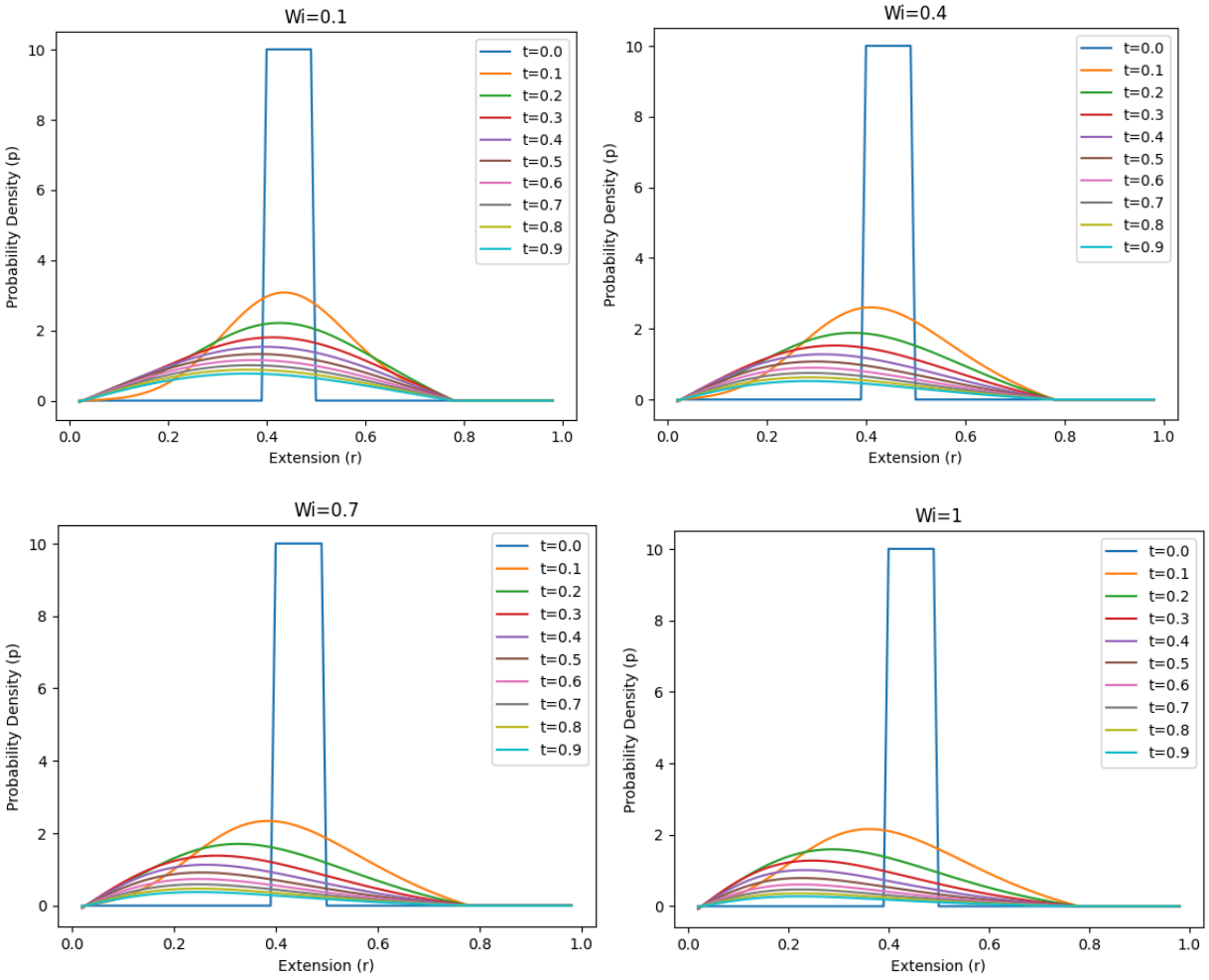
Even for the FENE force we get power law behavior for the decaying tail of the PDF.

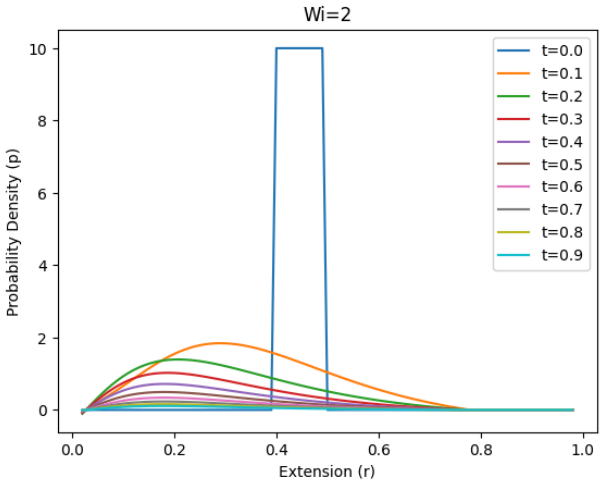


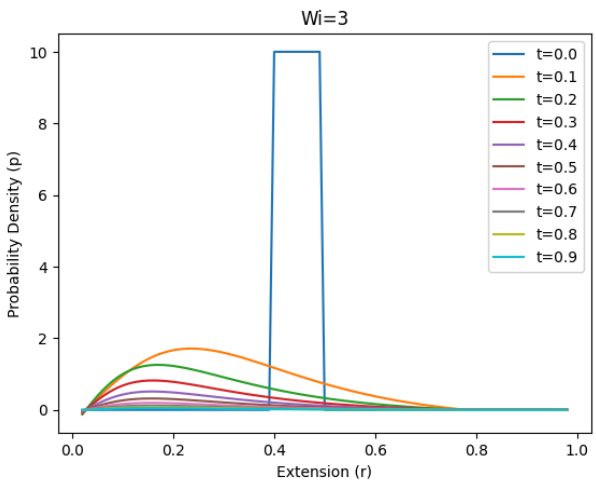


2. Polymer Scisson

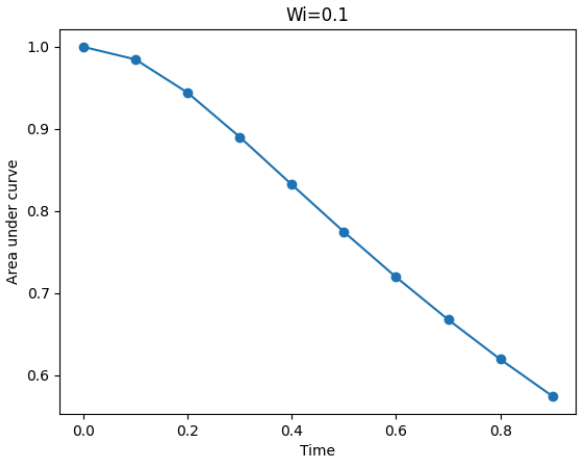
We assume that the dumbbell breaks when it stretches up to 80% of its maximum extension. No-flux boundary condition at R_{max} is replaced by a homogeneous Dirichlet condition setting the probability to 0 at $0.8*R_{max}$.

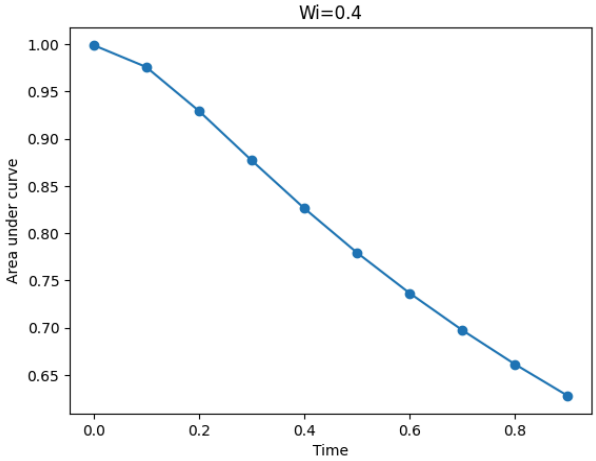


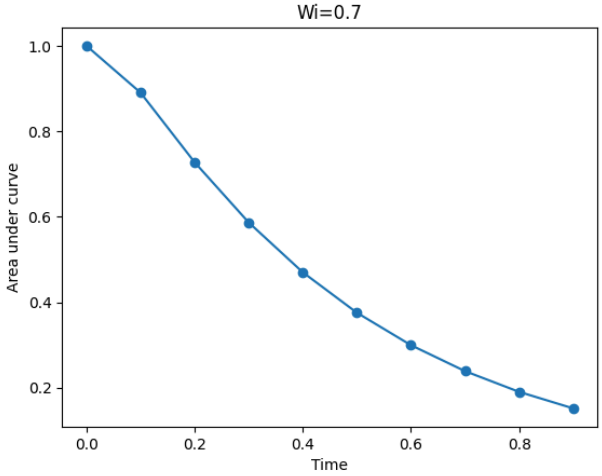


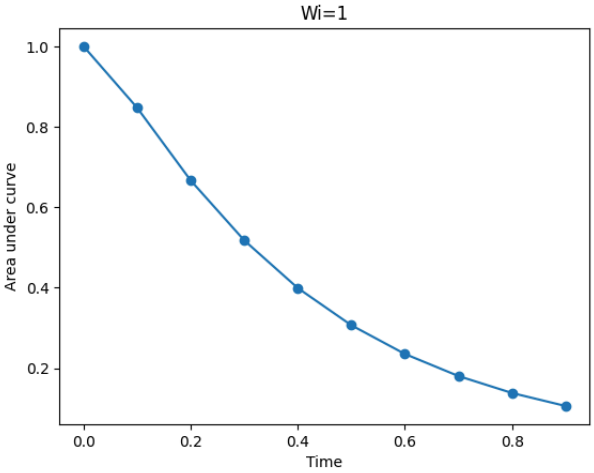


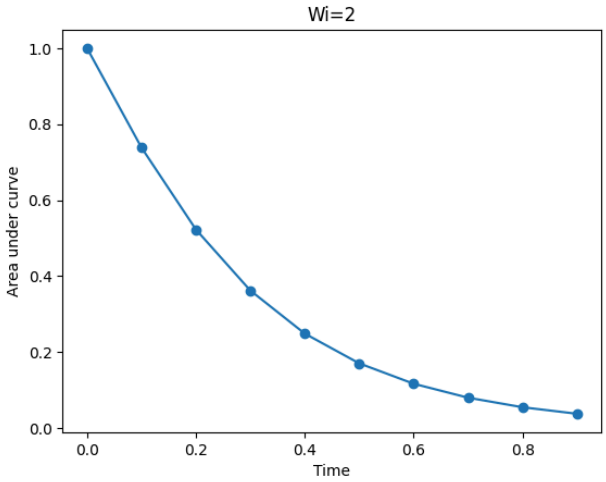
Early time behavior:

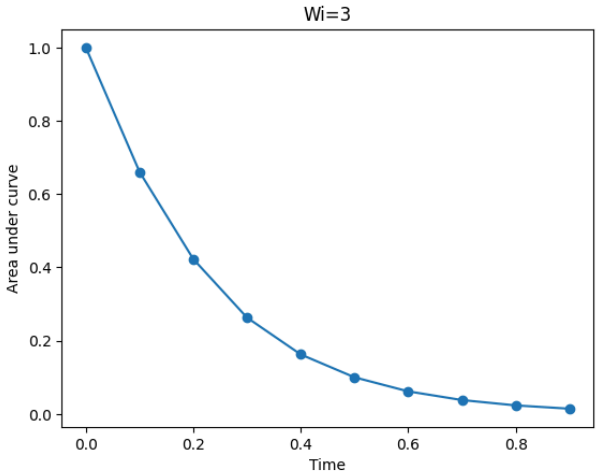


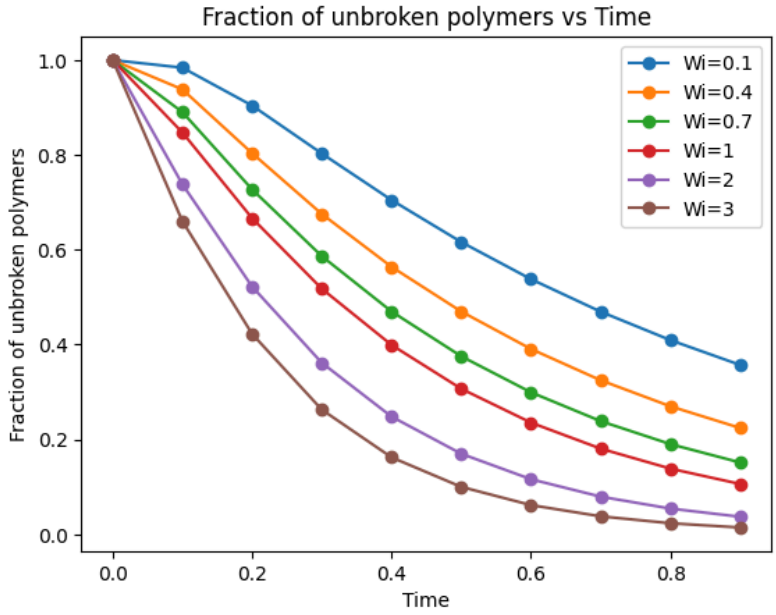




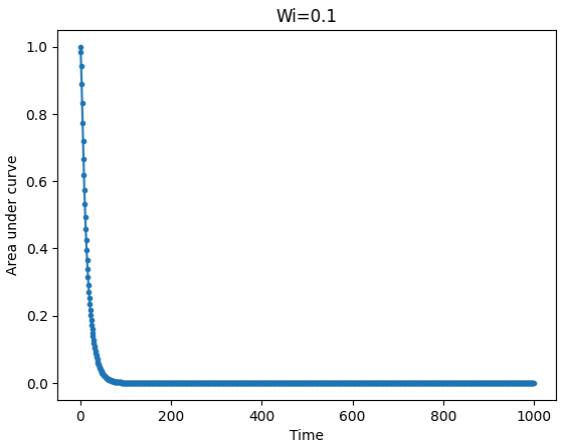


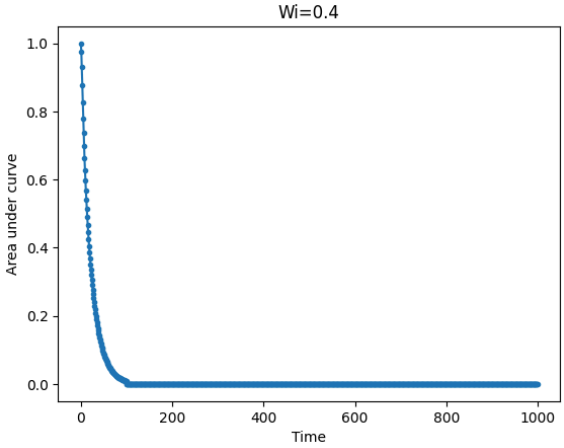

Wi=2
Area under curve
Time
0.0
0.2
0.4
0.6
0.8
1.0

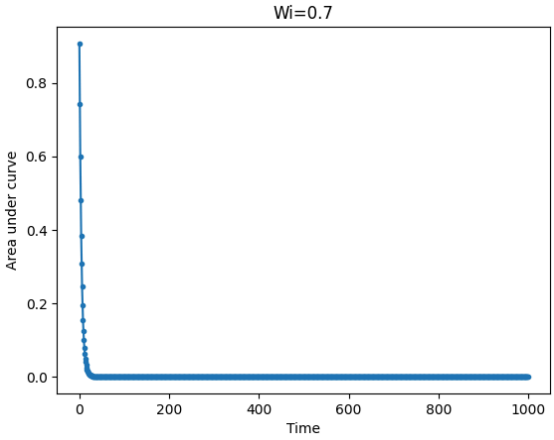

Wi=3
Area under curve
Time
0.0
0.2
0.4
0.6
0.8
1.0

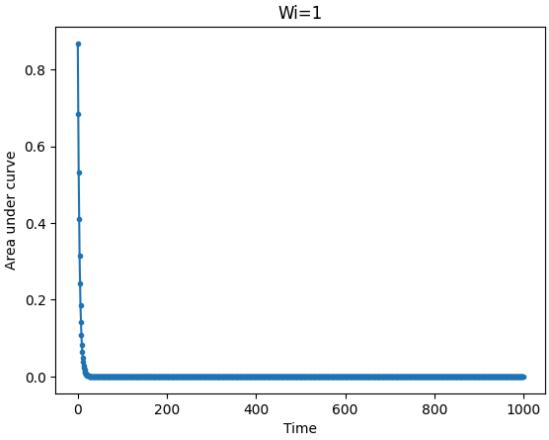

Fraction of unbroken polymers vs Time
Fraction of unbroken polymers
Time
Wi=0.1
Wi=0.4
Wi=0.7
Wi=1
Wi=2
Wi=3
0.0
0.2
0.4
0.6
0.8
1.0

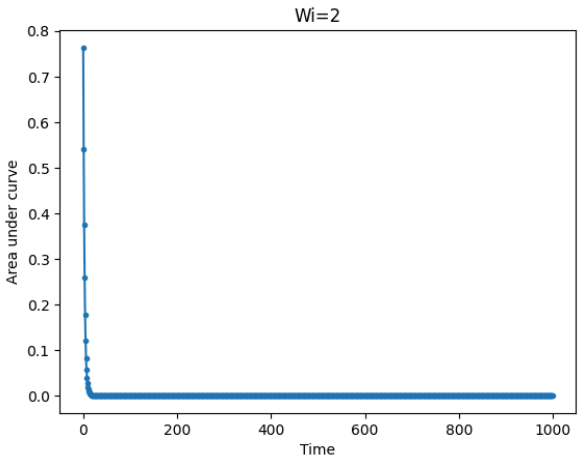
Long time behavior:

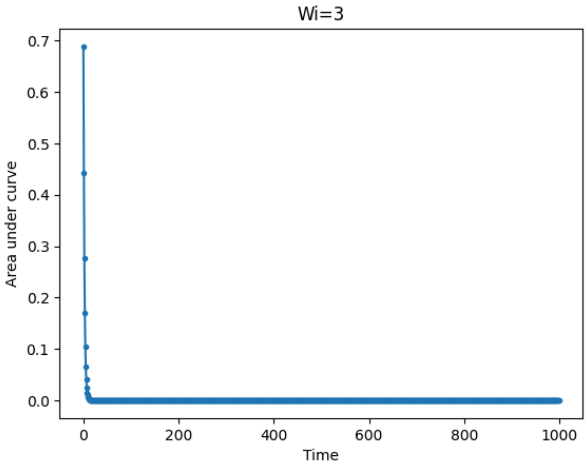


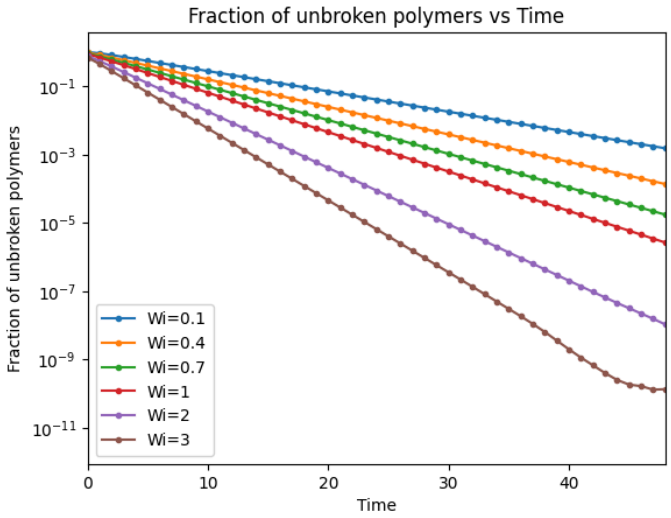












Thus, we observe that the expected exponential decay in the number of surviving unbroken polymers is seen. The rate of decay also increases with Wi, as expected.

Bibliography

1. Celani, A., Musacchio, S., & Vincenzi, D. (2005). Polymer transport in random flow. Journal of statistical physics, 118, 531-554.
2. Vincenzi, D., Watanabe, T., Ray, S. S., & Picardo, J. R. (2021). Polymer scission in turbulent flows. Journal of Fluid Mechanics, 912, A18.
3. J. W. Hatfield and S. R. Quake, Dynamic properties of an extended polymer in solution, Phys. Rev. Lett. 82:3548–3551 (1999)

Appendix

Code for Part 1:

```
import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from scipy.integrate import odeint

Wi = 0.5



qmax= 1

qmin = 0.02

qeq = 0.45

delq = 0.01

q_arr = np.arange(qmin,qmax-delq,delq)



T = 10

Nt= 1000

dt = T/Nt

t_range = np.arange(0,T,dt)



q_arr = np.round(q_arr, 2)

p0 = np.zeros_like(q_arr)
```

```
for i in np.arange(np.where(q_arr == qeq - 0.1/2)[0], np.where(q_arr ==
qeq + 0.1/2)[0]):

   p0[i] = 1/0.1



q = np.asarray(q_arr)

def f(q):

   return 1

def D1(q):

   return (8*Wi/3 - f(q))*q + + 2/(3*q)*qeq**2

def D2(q):

   return (qeq**2)/3 + (2/3)*Wi*q**2

## no flux BCs at both ends



def pde(p0,t):

   p = p0

   ret_array = []

   dp0dt = -(D1(q[1])*p[1]/(2*delq) -
D1(q[0]-delq)*(D2(q[1])*p[1]-2*delq*D1(q[0])*p[0])/D2(q[0]-delq))/(2*delq)
+ (D2(q[1])*p[1] - 2*D2(q[0])*p[0] +
D2(q[1])*p[1]-2*delq*D1(q[0])*p[0])/(delq**2)

   ret_array += [dp0dt]

   for i in range(1,np.shape(p)[0]-1):

       dpidt = -(D1(q[i+1])*p[i+1] - D1(q[i-1])*p[i-1])/2*delq +
(D2(q[i+1])*p[i+1] - 2*D2(q[i])*p[i] + D2(q[i-1])*p[i-1])/delq**2

       ret_array.append(dpidt)
```

```
    dpn1dt = -(D1(q[-1]+delq)*(2*delq*D1(q[-1])*p[-1]) +
D2(q[-2])*p[-2]/D2(q[-1]+delq) - D1(q[-2])*p[-2])/(2*delq) +
(2*delq*D1(q[-1])*p[-1] + D2(q[-2])*p[-2] - 2*D2(q[-1])*p[-1] +
D2(q[-2])*p[-2])/delq**2

    ret_array += [dpn1dt]

    return ret_array



Wi_values = [0.1,0.4,0.7,1, 2, 3]

for Wi in Wi_values:

    p = odeint(pde, p0, t_range)

    a = np.transpose(p)



    lgd = []

    for i in np.arange(0,int(Nt/10),int(Nt/100)):

        c = []

        for x in a:

            c.append(x[i])

        plt.plot(q/qeq,c)



        lgd.append(f't={np.round(i*dt,2)}')

        plt.title(f'Wi={Wi}')

        plt.ylabel('Probability Density (p)')

        plt.xlabel('Extension (r)/R0')
```

```
    plt.legend(lgd)

    plt.show()



from sklearn.linear_model import LinearRegression

# Arrays to store the fitted values of k for each Wi

alpha_values = []

Wi_values_l=[0.1,0.2,0.3,0.4,0.5,0.7,0.9,1,2,3]

for Wi in Wi_values_l:

    p = odeint(pde, p0, t_range)

    a = np.transpose(p)



    lgd = []



    time_step_index = np.where(np.isclose(t_range, 0.9))[0][0]

    a_at_time_step = a[:, time_step_index]

    plt.loglog((q[3:]/qeq),(a_at_time_step[3:]))

    plt.title(f'Wi={Wi}')

    plt.ylabel('Probability Density (p)')

    plt.xlabel('Extension (r)/R0')



    valid_r = q[q>=qeq]

    valid_p = a_at_time_step[q>=qeq]

    valid_r = valid_r[valid_p >0]
```

```
    valid_p = valid_p[valid_p >0]

    m,c = np.polyfit(np.log(valid_r),np.log(valid_p),1)

    # print(m,c)

    alpha = -1-m

    alpha_values.append(alpha)

    # Plot fitted line

    plt.plot(valid_r/qeq, (valid_r/qeq)**m, linestyle='--', color='red',
label=f'R^(-1-{np.round(alpha,2)})')#Fitted alpha={alpha:.2f}')

    plt.legend()

    plt.show()
```

Code for Part 2:

```
def pde(p0,t):

    p = p0

    ret_array = []

    dp0dt = -(D1(q[1])*p[1]/(2*delq) -
D1(q[0]-delq)*(D2(q[1])*p[1]-2*delq*D1(q[0])*p[0])/D2(q[0]-delq))/(2*delq)
+ (D2(q[1])*p[1] - 2*D2(q[0])*p[0] +
D2(q[1])*p[1]-2*delq*D1(q[0])*p[0])/(delq**2)

    ret_array += [dp0dt]

    for i in range(1,int(0.8*(np.shape(p)[0]-1))):
```

```
        dpidt = -(D1(q[i+1])*p[i+1] - D1(q[i-1])*p[i-1])/2*delq +
(D2(q[i+1])*p[i+1] - 2*D2(q[i])*p[i] + D2(q[i-1])*p[i-1])/delq**2

        ret_array.append(dpidt)

    for i in range(int(0.8*(np.shape(p)[0]-1)),np.shape(p)[0]):

        dpidt = 0

        ret_array.append(dpidt)

    return ret_array

Wi_values = [0.1,0.4,0.7,1, 2, 3]

points = np.zeros((np.shape(Wi_values)[0],10,np.shape(q)[0]))



for j in range(np.shape(Wi_values)[0]):

    Wi = Wi_values[j]

    p = odeint(pde, p0, t_range)

    a = np.transpose(p)

    lgd = []

    for i in np.arange(0,int(Nt/10),int(Nt/100)):

        c = []

        for x in a:

            c.append(x[i])

        plt.plot(q,c)

        plt.title(f'Wi={Wi}')

        lgd.append(f't={np.round(i*dt,2)}')

        plt.ylabel('Probability Density (p)')

        plt.xlabel('Extension (r)')
```

```
        points[j,int(i*dt*10),:] = c

    plt.legend(lgd)

    plt.show()

area = np.zeros((4,10,1))

for i in range(4):

    for j in range(10):

        area[i,j,:] = np.trapz(points[i,j,:],dx=delq)

for i in range(4):

    plt.plot(np.arange(0,1,0.1),area[i,:,:],marker='o')

    plt.xlabel('Time')

    plt.ylabel('Area under curve')

    plt.title(f'Wi={Wi_values[i]}')

    plt.show()

for i in range(4):


plt.plot(np.arange(0,1,0.1),area[i,:,:],marker='o',label=f'Wi={Wi_values[i
]}')

    plt.xlabel('Time')

    plt.legend()

    plt.ylabel('Fraction of unbroken polymers')

    plt.title('Fraction of unbroken polymers vs Time')

Wi_values = [0.1,0.4,0.7,1]

points = np.zeros((np.shape(Wi_values)[0],Nt,np.shape(q)[0]))
```

```
for j in range(np.shape(Wi_values)[0]):

   Wi = Wi_values[j]

   p = odeint(pde, p0, t_range)

   a = np.transpose(p)

   for i in range(Nt):

       c = []

       for x in a:

           c.append(x[i])

       points[j,int(i*dt*10),:] = c
area = np.zeros((4,Nt,1))

for i in range(4):

   for j in range(Nt):

       area[i,j,:] = np.trapz(points[i,j,:],dx=delq)

for i in range(4):

   plt.plot(range(Nt),area[i,:,:],marker='.')

   plt.xlabel('Time')

   plt.ylabel('Area under curve')

   plt.title(f'Wi={Wi_values[i]}')

   plt.show()

for i in range(4):


plt.semilogy(range(Nt),area[i,:,:],marker='.',label=f'Wi={Wi_values[i]}')

   plt.xlabel('Time')

   plt.xlim((0,65))
```

```
  plt.legend()

  plt.ylabel('Fraction of unbroken polymers')

  plt.title('Fraction of unbroken polymers vs Time')
```