Diffussive and Stochastic Processes Programming Assignment 2024

Tomás Ricardo Basile Álvarez

1. Brownian Particle

Instructions

We consider a Brownian particle in the over-damped limit and under the effect of force from potential $U(X)$. The potential has the functional form

$$U(X) = \frac{1}{4}X^4 - \frac{1}{2}X^2 + X$$

The potential is plotted in Fig. 1.

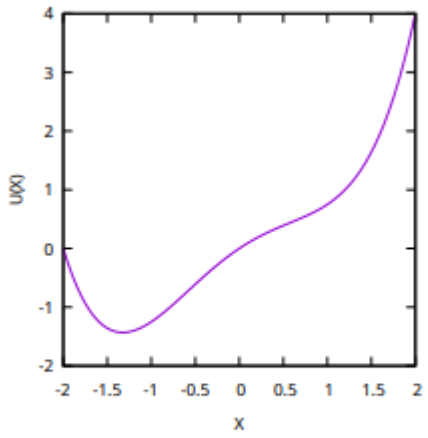


Figure 1: $U(X)$ defined in eq. (1).

Hence, the force is given by

$$-U'(X) = -(X^3 - X + 1)$$

where the prime denotes the derivative by its argument. The position of the particle at time t, $X(t)$, obeys the following Langevin equation:

$$\frac{dX}{dt} = -\frac{1}{\eta}(X^3 - X + 1) + \xi(t)$$

Here, η is the drag coefficient and a positive constant. $\xi(t)$ is a Gaussian white noise, and it satisfies

$$\langle \xi(t) \rangle = 0, \quad \langle \xi(t_1)\xi(t_2) \rangle = 2D\delta(t_1 - t_2)$$

Here, D is the diffusion coefficient and a positive constant. Set $D = 0.2$, $\eta = 2$. **Set the initial position to be** $X(0) = 1$. Numerically simulate the trajectory $X(t)$ by using the Euler method. Set the time step for integration to $\Delta t = 0.01$. Plot the following quantitie

1. Solution

In [10]:
```
import numpy as np
import matplotlib.pyplot as plt
np.random.seed(0)
```

First, we create a function that solves the differential equation.

For that, we use the Euler method, so that we divide the time interval into steps of size $\Delta t = 0.01$. Then, starting from $X(0) = 1$, at each step, we update X by:

$$X(t + \Delta t) = X(t) + \left(-\frac{1}{\eta}(X^3 - X + 1) \right) \Delta t + \Delta W.$$

In this expression, the first part is simply the normal Euler method for the equation $\frac{dX}{dt} = -\frac{1}{\eta}(X^3 - X + 1)$. On the other hand, as we saw in class, the term ΔW comes from integrating the $\xi(t)$ for the small time interval $(t, t + \Delta t)$. ΔW is a random number with mean 0, variance $2D\Delta t$ and distributed with a Gaussian distribution.

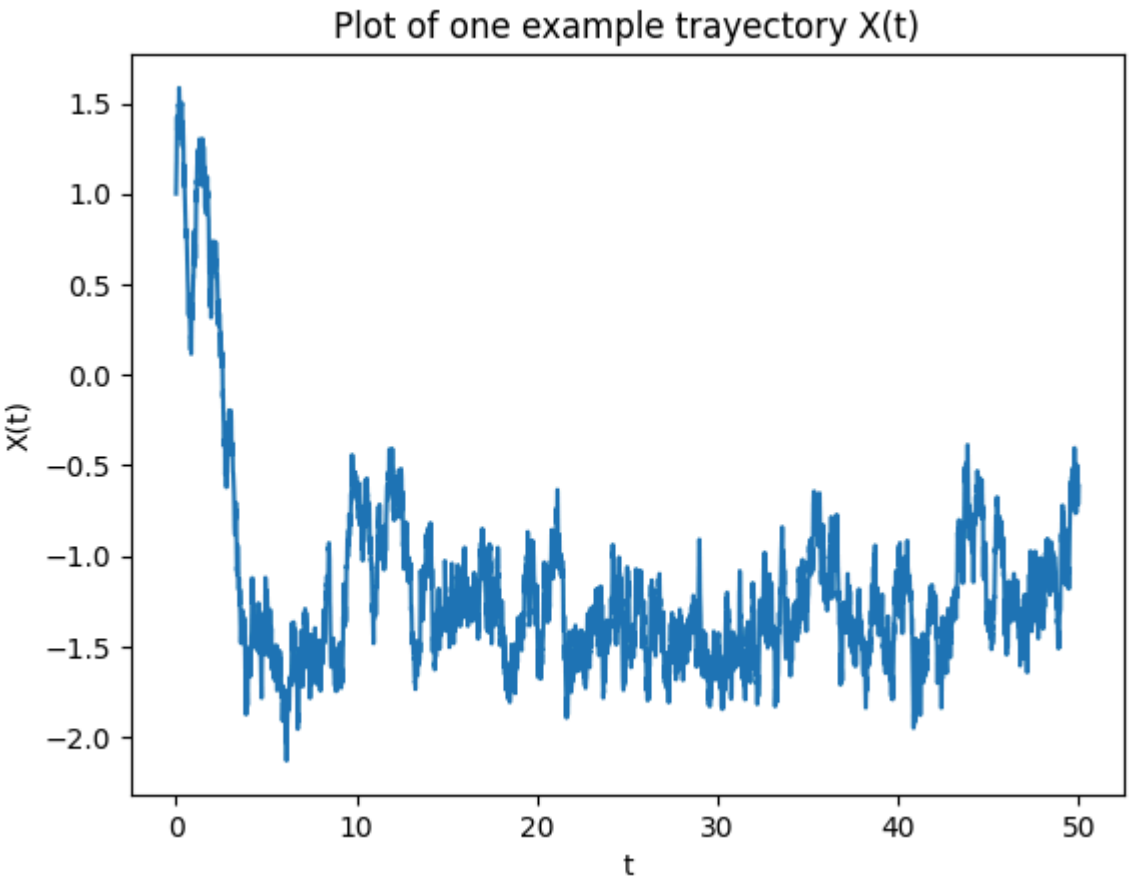
In [11]:
```
def simulate(X0, deltat, tmax, D, eta):
    # First we calculate the number of steps that we will need:
    steps = int(tmax//deltat)


    # Set the initial value of X to X0
    X = X0
    # We create a l9ist that will contain the trajectory of X
    Xlist = [X0]



    #We iterate over the steps
    for i in range(steps):

        # For each time step, we calculate a random Delta W
        # We obtain it from a random distribution with mean 0 and std (2D De
        deltaW = np.random.normal(0, (2*D*deltat)**(1/2))

        # We update X and append it to the list
        X += (- 1/eta * (X**3 - X + 1) )*deltat + deltaW
```

```
        Xlist.append(X)

    return(Xlist)
```

1. (i)

Plot one example trajectory $X(t)$ as a function of time from $t = 0$ to $t = 50$.

In [12]:
```
D = 0.2;  eta = 2;  X0 = 1;  Deltat = 0.01; tmax = 50

Xs = simulate(X0,Deltat,tmax,D,eta)

times = np.linspace(0,tmax,len(Xs))

plt.plot(times,Xs)
plt.xlabel("t")
plt.ylabel("X(t)")
plt.title("Plot of one example trayectory X(t)")
plt.show()
```



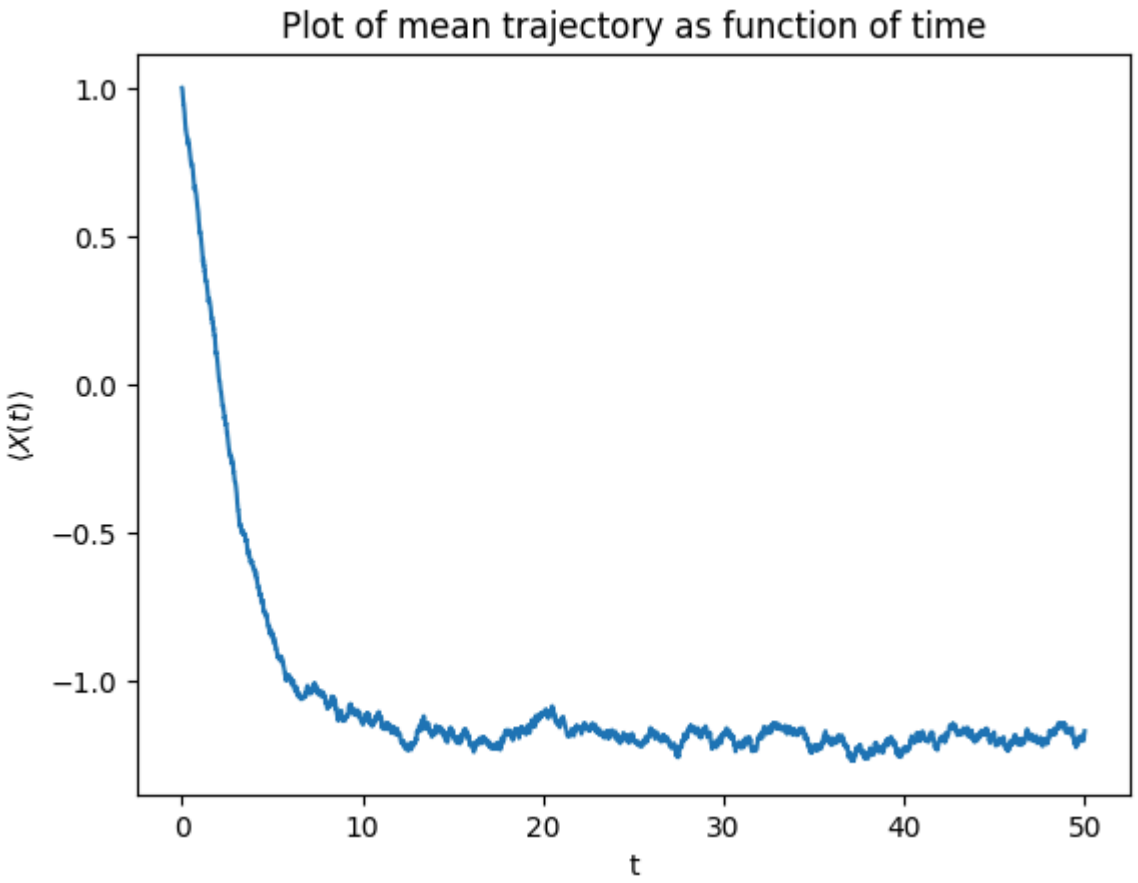
The result makes sense. Seeing the graph of the potential $U(X)$, we would expect that if X starts at 1, it would then start moving to lower values of X, in the direction where the potential gets smaller. This will continue until it getting close to the minimum of $U(X)$. That is what we see, with some extra noise.

1. (ii)

Plot the mean trajectory $\langle X(t) \rangle$ as a function of time from $t = 0$ to $t = 50$.

For this, we generate 200 random trajectories. Then, we average over them at each time to get the mean trajectory.

In [13]:
```
#allXs is a list were we will put all the trajectories
allXs= []

#iterate over different realizations
for j in range(200):
    #get a random trajectory and add it to the list.
    Xs = simulate(X0,Deltat,tmax,D,eta)
    allXs.append(Xs)


allXs = np.array(allXs)

#Calculate the mean across different trajectories
means = allXs.mean(axis=0)


plt.plot(times,means)
plt.title("Plot of mean trajectory as function of time")
plt.xlabel("t")
plt.ylabel(r"$\langle X(t) \rangle$")
plt.show()
```

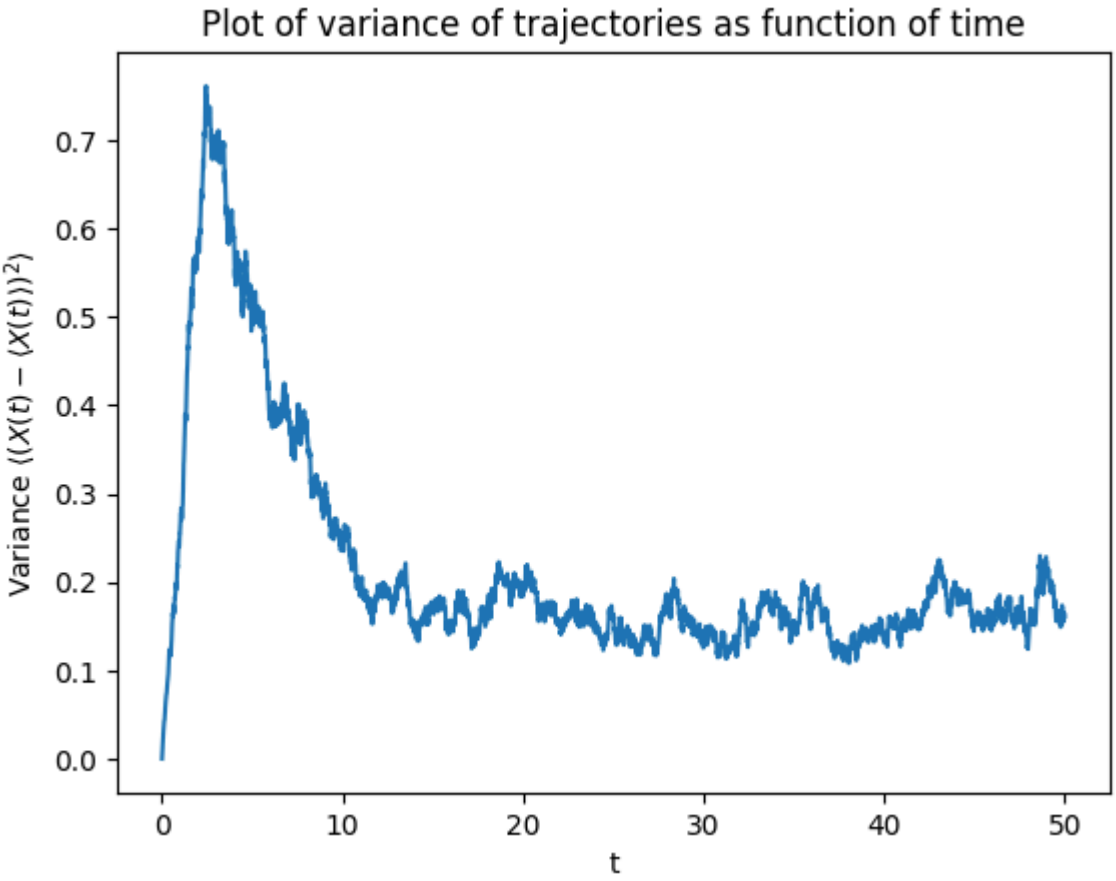


1. (iii)

Plot the variance as a function of time from $t = 0$ to $t = 50$.

Here we use the same trajectories as before but instead of calculating the mean, we calculate the std and then square it to get the variance.

In [14]:
```
# Calculate the variance of the trajectories.
# We do it simply using std function and then squaring it.

variances = allXs.std(axis=0)**2

plt.plot(times,variances)

plt.title("Plot of variance of trajectories as function of time")
plt.xlabel("t")
plt.ylabel(r"Variance $\langle (X(t) - \langle X(t) \rangle)^2  \rangle$")
plt.show()
```



2. Flu

Instructions

Suppose there are Ω people in total in a community (Ω is an integer and a constant). N people in the community have flu. If a person with the flu meets a healthy person who does not have the flu, there is a chance for a healthy person to catch the flu. The person who has the flu recovers at a constant rate per person. The recovered healthy person can catch the flu again. Also, a healthy person can catch the flu spontaneously at a small constant rate per person, representing the rare event of catching a cold from people outside of the community. Assuming that the population is well-mixed inside the community, we express this process as

- $N \to N+1$ at a rate $\alpha \frac{N}{\Omega}(\Omega - N) + \epsilon(\Omega - N)$
- $N \to N-1$ at a rate γN.

Here, α, ϵ, and γ are positive constants. $(\Omega - N)$ is the number of healthy people in the community and $\frac{N}{\Omega}$ is proportional to the density of people who have a cold. Set $\Omega = 100, \alpha = 2.5, \epsilon = 0.01$, and $\gamma = 1$. Set the initial number of people with the flu at time zero to 1, i.e., $N(0) = 1$. Simulate the process by using the Gillespie method.

2. Solution

2. (i)

Plot one simulated trajectory of $N(t)$ as a function of time t from $t = 0$ to $t = 100$.

We simulate it using the Gillespie algorithm. Particularly, we use the Method 1 from the notes. For this, we need to consider the different changes that can happen to the system and the rates at which they happen. This is stated directly in the problem statement:

- $N \to N + 1$ at a rate $r_{sick} = \alpha \frac{N}{\Omega}(\Omega - N) + \epsilon(\Omega - N)$
- $N \to N - 1$ at a rate $r_{cure} = \gamma N$.

Then, we have to calculate the total rate $K = r_{sick} + r_{cure}$. We select a random number a uniformly from 0 to 1 and use it to calculate the duration until the next event $\tau = -\frac{\ln a}{K}$. Then, we set the time forward by τ.

After that, we choose which of the events will happen depending on their rates. We do so by selecting a random number a' uniformly between 0 and 1 and if $a' < \frac{r_{sick}}{K}$, we proceed with the sick event, otherwise we proceed with the cure event.

We repeat this until the current time surpasses the maximum time t_{max}.

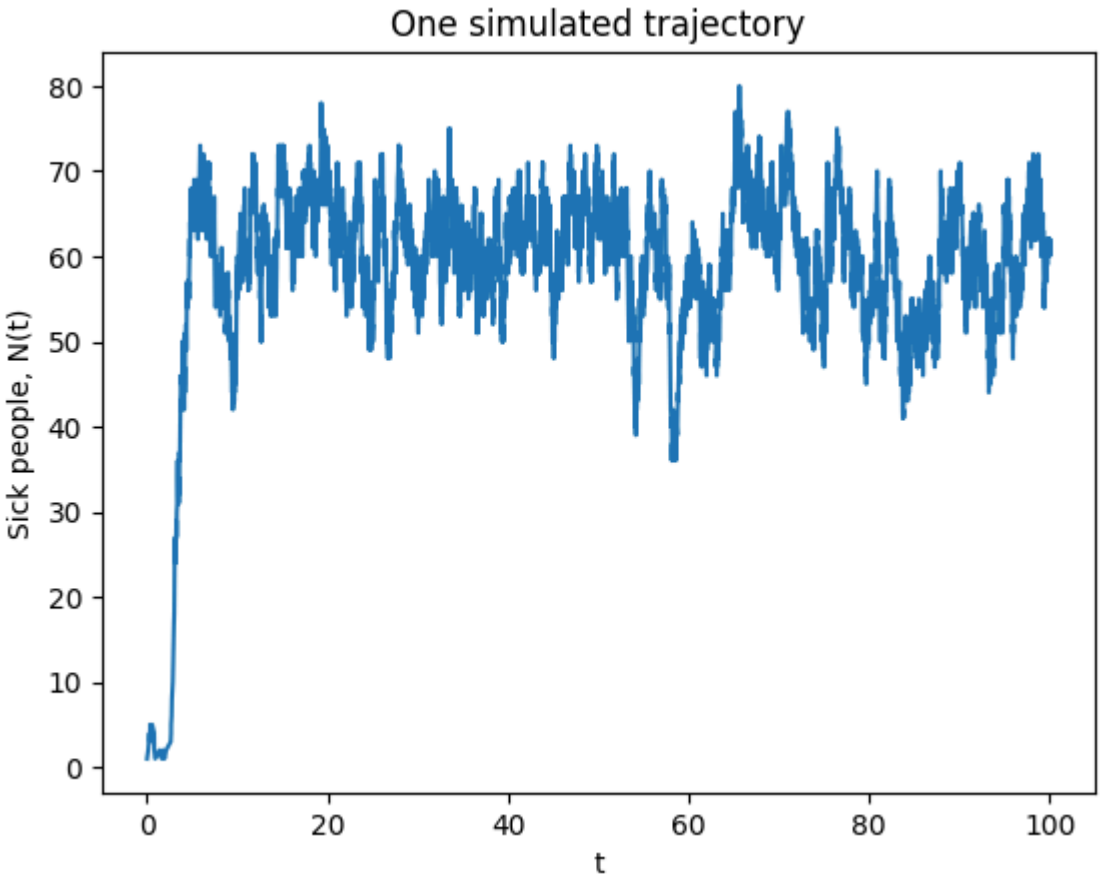
In [25]:
```
import numpy as np
import matplotlib.pyplot as plt
np.random.seed(0)
```

In [26]:
```
def run_gillespie(Omega,alpha,epsilon,gamma,N0,tmax=100):
    #initialize time:
    t = 0
    times = [0]

    #initialize number of sick people.
    N = N0
    #We will keep a list with the number of sick people.
    Nlist = [N0]



    while True:

        #Rate at which one extra person gets sick
        rate_sick = alpha*N/Omega * (Omega-N) + epsilon*(Omega-N)

        #Rate at which one person is cured.
        rate_cure = gamma*N
```

```
        #Total rate:
        K = rate_sick + rate_cure


        # Select time tau and set forward time by tau:
        a = np.random.random()
        tau = -np.log(a)/K
        t+=tau

        #We also keep a list of times at which there where jumps
        times.append(t)

        #If we excede the total time, we stop and dont update the variables
        if t>tmax:
            break


        # We select which event to do and do it
        a2 = np.random.random()

        if a2 < rate_sick / K:
            #Sick event:
            N += 1
        else:
            #Cure event:
            N -= 1



        Nlist.append(N)


    return(times,Nlist)
```

In [27]:

```
Omega = 100; alpha = 2.5; epsilon = 0.01; gamma = 1; N0 = 1


times,Nlist = run_gillespie(Omega,alpha,epsilon,gamma,N0)
plt.plot(times[:-1],Nlist)
plt.title("One simulated trajectory")
plt.xlabel("t")
plt.ylabel("Sick people, N(t)")
plt.show()
```



We see that the number of sick people increases until it reaches a steady state at around $N = 60$ and then fluctuates around it. We can check that this makes sense by calculating the value of N at the steady state and seeing if it is around 60. The steady state happens when the rate of one person getting sick is equal to the rate of one person getting cured, so that $\alpha \frac{N}{\Omega}(\Omega - N) + \epsilon(\Omega - N) = \gamma N$. Substituting the values of the parameters and solving for N, we get that $N = 60.2637$.

2. (ii)

After some time, the system reaches a steady state. Numerically calculate the mean $\langle N \rangle$ and the variance $\langle (N - \langle N \rangle)^2 \rangle$ in the steady-state from the simulation. When you are taking averages, pay attention that inter-event intervals in Gillespie simulation is uneven.

We will calculate these quantities both using the ensemble mean and the time mean, just to see that we get the same result.

1. Ensemble averages

For this, we will simulate many random trajectories and compare their values at time $t = 100$. The value of N at time 100 is the last value of N obtained from the simulation,

because the simulation stops when $t + \tau$ becomes greater than 100. When that happens, the simulation stops, N is not updated, so the last value returned by the simulation is the value at time $t = 100$.

In [28]:
```
#many_traj is a list were we will put the trajectories
many_traj = []

#Nfinal is a list with the values of N at time t=100
Nfinal = []

#iterate over different realizations
for j in range(100):
    #get a random trajectory
    times,Nlist = run_gillespie(Omega,alpha,epsilon,gamma,N0)
    #Take the last value of N (the value at time 100)
    Nfinal.append(Nlist[-1])

    many_traj.append(Nlist)
```

Ensemble mean:

In [29]:
```
print("The ensemble mean after reaching steady state is:")
print(np.mean(Nfinal))
```

```
The ensemble mean after reaching steady state is:
59.45
```

Ensemble variance:

In [30]:
```
print("The ensemble variance after reaching steady state is:")
print(np.std(Nfinal)**2)
```

```
The ensemble variance after reaching steady state is:
34.7075
```

2. Time averages

Now we take the average by considering only one trajectory and averaging over its values after it has reached the steady state. We take the average from time $t = 20$ onwards, because seeing the graph of the trajectory, this is when we can be absolutely sure that we have reached the steady state.

In [31]:
```
# This is the index when time surpasses 20 seconds
index_t20 = np.where(np.array(times)>20)[0][0]

#We take only the times and Nlist after 20 seconds have passed:
times_20 = times[index_t20:]
Nlist_20  = Nlist[index_t20:]
```

Now we calculate the time average of the $N(t)$. To do so, we need to take into account that the values of N last for different amounts of time. Therefore, we weight them by their

corresponding duration, so that we calculate the mean as $\sum_{i=1}^{\text{all events}} N_i \tau_i / \sum_{i=1} \tau_i$.

Considering how we made the algorithm, $Nlist[i]$ is the value of N between times $times[i]$ and $times[i+1]$, so the mean is done as follows:

In [32]:
```
total_time = times_20[-1]-times_20[0]
mean = 0

for i in range(len(times_20)-1):
    #We add to the mean the value of Nlist between jumps,
    #multiplied by the time the system was at that value.
    mean += (Nlist_20[i])*(times_20[i+1]-times_20[i])/total_time


print("The time mean after reaching steady state is:")
print(mean)
```

```
The time mean after reaching steady state is:
59.05904722974619
```

In [33]:
```
# Here we do the same for the variance, adding up together (Nlist[i]-mean)^2
total_time = times_20[-1]-times_20[0]

var=0
for i in range(len(times_20)-1):
    var += (times_20[i+1]-times_20[i])*(Nlist_20[i]-mean)**2/total_time


print("The time variance after reaching steady state is:")
print(var)
```

```
The time variance after reaching steady state is:
44.44730605335161
```

We see that the result for both ensemble and time averages are pretty much the same, a mean of around 60 and a variance of around 40.

3. Flu with immunity

Instructions

Let's modify the previous model to take into account immunity. Suppose a community has Ω people in total (Ω is an integer and a constant). N people in the community have the flu. We assume that a recovered person becomes immune to the flu and, hence, does not catch the flu. The immunity is lost at a constant rate per person. Denoting the number of immune people as M, the number of healthy but non-immune people becomes $(\Omega - N - M)$. We express this process as

- $N \to N+1$ at a rate $\alpha \frac{N}{\Omega}(\Omega - N - M) + \varepsilon(\Omega - N - M)$

- $N \to N-1$ and $M \to M+1$ at a rate γN.
- $M \to M-1$ at a rate βM.

Here, $\alpha, \varepsilon, \gamma, \beta$ are positive constants. Set $\Omega = 100, \alpha = 2.5, \epsilon = 0.01$, and $\gamma = 1$ and $\beta = 0.3$. Set the initial number of people with the flu at time zero to 1, i.e., $N(0) = 1$ and no one is immune at the start, i.e., $M(0) = 0$. Simulate the process by using the Gillespie method.

3. Solution

3. (i)

Plot one simulated trajectory of $N(t)$ and $M(t)$ as a function of time t from $t = 0$ to $t = 100$.

We do the same as for the previous exercise, but now the rates are a little different and we have an extra event in which an immune person looses immunity.

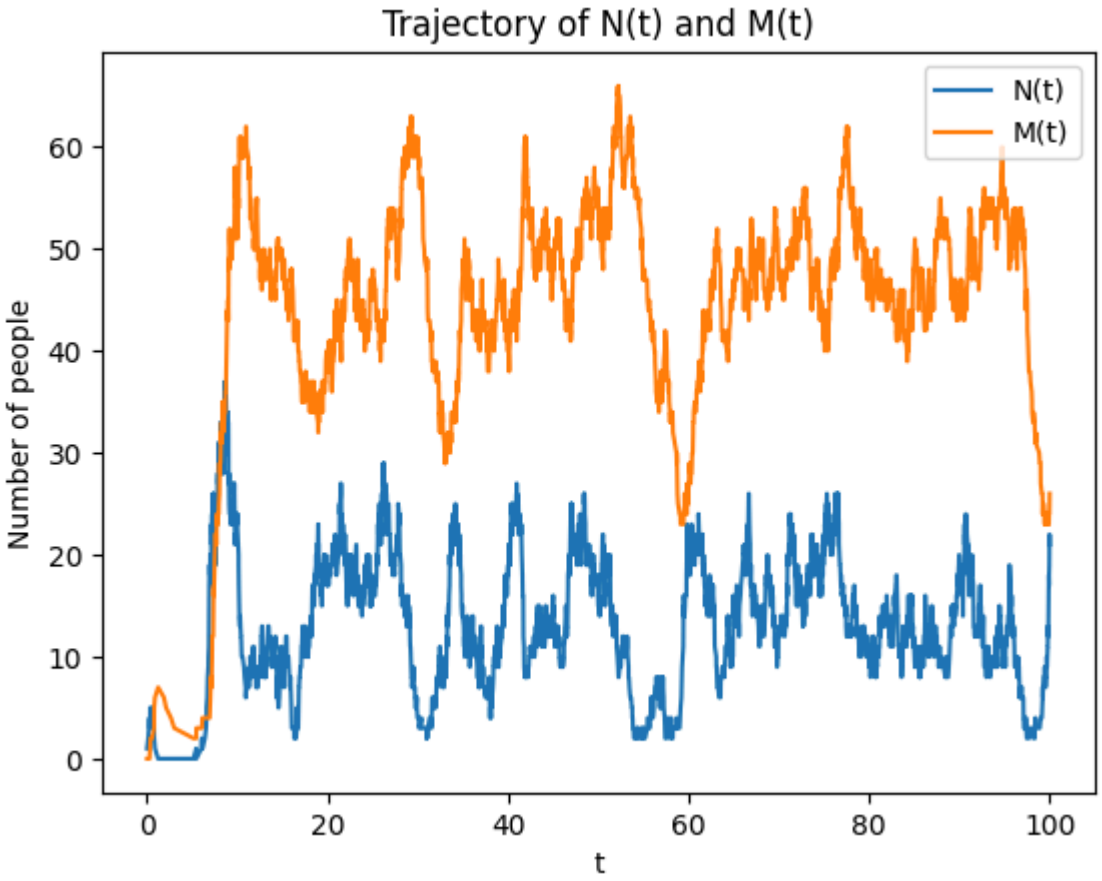
Also, we now need to keep track of two numbers to define the state of the system: N and M.

In [34]:
```
np.random.seed(0)

def run_gillespie2(Omega,alpha,epsilon,gamma,N0,M0,beta,tmax=100):
    #initialize time:
    t = 0
    times = [0]

    #initialize sick people:
    N = N0
    Nlist = [N0]

    #initialize immune people:
    M = M0
    Mlist = [M0]


    while True:
        # Calculate the rates of getting sick, getting cured and loosing imm

        rate_sick = alpha*N/Omega * (Omega-N-M) + epsilon*(Omega-N-M)
        rate_cure = gamma*N
        rate_loose_im = beta*M



        # We calculate the total rate:
        K = rate_sick + rate_cure + rate_loose_im

        # Select a random number and use it to calculate the time to the nex
```

```
        a = np.random.random()
        tau = -np.log(a)/K
        t+=tau

        #Append the time t to the list of times
        times.append(t)

        # If we go over the total time, stop:
        if t>tmax:
            break


        #Take a random number to choose which event to do.
        a2 = np.random.random()


        #If this random number is less than rate_sick/K, the sick event happ
        if a2 < rate_sick / K:
            N += 1

        # If a2 is bigger than rate_sick/K but smaller than (rate_sick + rat
        # the cure event happens.
        elif a2 < (rate_sick + rate_cure) / K:
            N -= 1
            M += 1

        #Else (a2 is bigger than (rate_sick + rate_cure) / K ), the loose im
        else:
            M -= 1



        Nlist.append(N)
        Mlist.append(M)


    return(times,Nlist,Mlist)
```

In [35]:

```
Omega = 100; alpha = 2.5; epsilon = 0.01; gamma = 1; N0 = 1; M0=0;beta=0.3



times,Nlist,Mlist = run_gillespie2(Omega,alpha,epsilon,gamma,N0,M0,beta,tmax


plt.plot(times[:-1],Nlist,label="N(t)")
plt.plot(times[:-1],Mlist,label="M(t)")
plt.title("Trajectory of N(t) and M(t)")
plt.legend()
plt.xlabel("t")
plt.ylabel("Number of people")
plt.show()
```



We can see that there is a lot of variation, but the steady state seems to be $N_{ss} \simeq 15$ and $M_{ss} \simeq 45$.

To check if this makes sense, we can solve the equations to find the steady state. In the steady state, the rate at which N increases by one should be equal to the rate at which it decreases by one, so we should have $\alpha \frac{N}{\Omega}(\Omega - N - M) + \epsilon(\Omega - N - M) = \gamma N$. This should also be true for M, so that $\gamma N = \beta M$. This gives us two equations with the two variables N, M and after substituting the values of the parameters, the solutions are $N_{ss} = 14.1008$ and $M_{ss} = 47.0026$. This seems to agree with our graph.

In [ ]: