

## Lösungen zum Übungsblatt 5

Johannes Dörr

Übungsgruppe 2, Jürgen Lampe

---

### Problem 1: Implementation of Gillespie's algorithm

#### Introduction

We consider a two-state system  $n \in \{-1, 1\}$ . The probability of finding the system in state -1 (respectively 1) at time  $t$  is denoted  $P_-(t)$  (respectively  $P_+(t)$ ). The master equation describing the evolution of the system has the form:

$$\dot{P}_-(t) = \gamma_{-+}P_+(t) - \gamma_{+-}P_-(t) \quad (1)$$

$$\dot{P}_+(t) = \gamma_{+-}P_-(t) - \gamma_{-+}P_+(t) \quad (2)$$

$\gamma_{+-}, \gamma_{-+} \geq 0$  are the transition rates. This system is analytical resolvable. So we'll find a solution for  $P_-(t)$  and  $P_+(t)$  in task a). Another way to solve such a problem is the implementation of the Gillespie's algorithm for concrete  $\gamma_{+-}$  and  $\gamma_{-+}$  in the computer. We'll do this in task b). We have two ways to solve the problem and we'll compare the results.

#### Tasks

- a) Solve the Master equation (eq. 1, 2) analytically with the initial conditions  $P_-(t=0) = 1$  and find the probability distribution  $P(t) = (P_-(t), P_+(t))$ , the respective moments  $\langle m \rangle_t$ ,  $\langle m^2 \rangle_t$  of the random variable  $m := \frac{n+1}{2}$  and the stationary solution  $\mathbf{P}^0 = (P_-^0, P_+^0)$ .

We also have to compute the respective moments  $\langle m \rangle_t$  and  $\langle m^2 \rangle_t$  of the random variable  $m := \frac{n+1}{2}$ , whereas  $n$  is the state defined above.

- b) Implement Gillespie's algorithm (chose  $\gamma_{-+} = 0, 1$  and  $\gamma_{+-} = 0, 5$ ) and explain the source code.
- Plot 3 sample trajectories for  $T \approx 50$ .
  - Estimate  $P_+(T)$  for  $T = 1, 10, 100$  from  $N = 10, 100, 1000$  trajectories.
  - Estimate  $P_+(t)$  and  $P_-(t)$  for  $N = 1000$  trajectories.
  - Plot the histograms of time periods, the system stays in one state.

## Methods

a) We know that the propability to find a particle in state 1 or -1 has to be 1 for all  $t$ :

$$P_-(t) + P_+(t) = 1 \quad (3)$$

Insertion of (eq. 3) in (eq. 1) gives us one easily solvable linear first order differential equation for  $P_-(t)$ . With (eq. 3) we obtain  $P_+(t)$ .

For the calculation of the moments  $\langle m \rangle_t$  and  $\langle m^2 \rangle_t$  we only used basic calculation rules of expectations.

- b) We implement Gillespie's algorithm in Java. At first, we program the *Trajectory* class, that computes and stores one trajectory of the system. Then, we can create many of them and analyze them in a second step. Since all trajectories are generated and stored, this is not the most cost-effective way, but it makes the implementing really straight forward.
- i) With the *Trajectory* class, we can simply generate and print as many trajectories as wanted.
  - ii) We generate  $N$  trajectories and count, how many of them are in state +1 at the time  $T$ .
  - iii) We do the same as in ii) for many times  $t = n \cdot \Delta t$  and plot the result, respectively for the state -1 too.
  - iv) The histogram of the times, the system persists in a certain state, is computed by analysing one trajectory and arranging all found time periods in an array, that stores the number of periods that apply to a corresponding time interval (0-1s, 1-2s, ...). This is illustrated in a graph.

## Results

a) The propability distribution of the considered two-state system is (eq. 6):

$$\mathbf{P}(t) = \begin{pmatrix} P_-(t) \\ P_+(t) \end{pmatrix} = \begin{pmatrix} \frac{\gamma_{-+}}{\gamma_{-+} + \gamma_{+-}} + \left(1 - \frac{\gamma_{-+}}{\gamma_{-+} + \gamma_{+-}}\right) e^{-(\gamma_{-+} + \gamma_{+-})t} \\ 1 - \frac{\gamma_{-+}}{\gamma_{-+} + \gamma_{+-}} + \left(\frac{\gamma_{-+}}{\gamma_{-+} + \gamma_{+-}} - 1\right) e^{-(\gamma_{-+} + \gamma_{+-})t} \end{pmatrix}$$

And the stationary state is (eq. 7):

$$\mathbf{P}^0 = \begin{pmatrix} P_-^0 \\ P_+^0 \end{pmatrix} = \begin{pmatrix} \frac{\gamma_{-+}}{\gamma_{-+} + \gamma_{+-}} \\ 1 - \frac{\gamma_{-+}}{\gamma_{-+} + \gamma_{+-}} \end{pmatrix}$$

We obtain for the moments  $\langle m \rangle_t$  and  $\langle m^2 \rangle_t$  (eq. 9, 11):

$$\langle m \rangle_t = \langle m^2 \rangle_t = 1 - P_-(t)$$

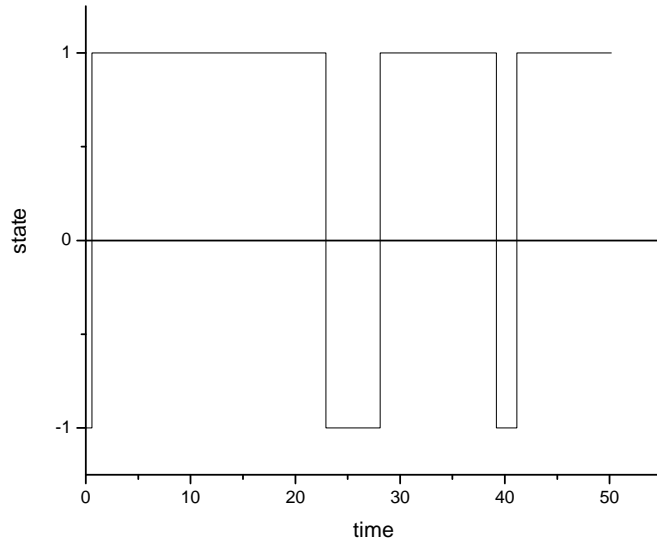


Figure 1: Sample trajectory (1.)

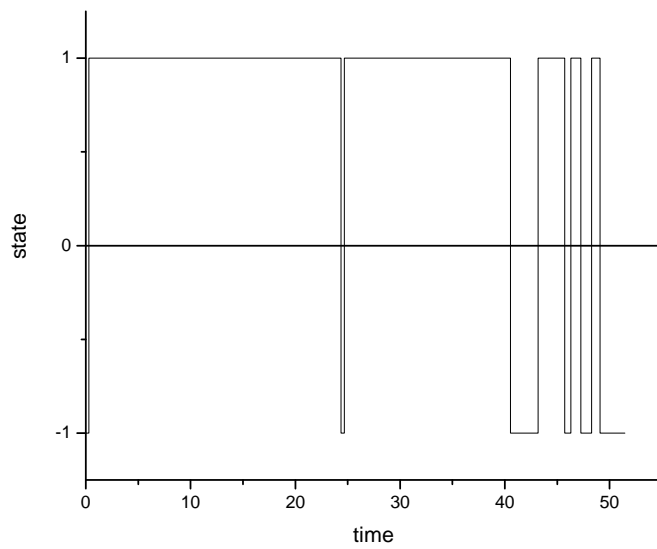


Figure 2: Sample trajectory (2.)

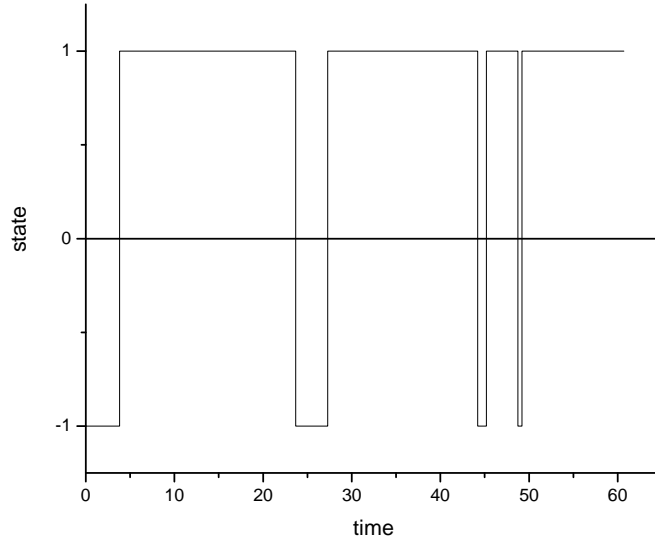


Figure 3: Sample trajectory (3.)

b) For the implementation of Gillespie's algorithm, see appendix.

i) The figures 1, 2 and 3 show three possible results computed by the simulator.

ii) The following table shows the results of  $P_+(T)$  depending on  $T$  and  $N$ :

$N = 10$	$P_+(1) = 0,4 \pm 0,3$	$P_+(10) = 0,8 \pm 0,2$	$P_+(100) = 0,8 \pm 0,2$
$N = 100$	$P_+(1) = 0,4 \pm 0,1$	$P_+(10) = 0,83 \pm 0,07$	$P_+(100) = 0,8 \pm 0,1$
$N = 1000$	$P_+(1) = 0,38 \pm 0,05$	$P_+(10) = 0,83 \pm 0,02$	$P_+(100) = 0,83 \pm 0,05$

See appendix for a detailed description of computing these values. To indicate the fluctuation of the result, we give an interval that encloses the values we got by repeating each computation 30 times.

iii) Figure 4 and 5 show the evolution over the time of the probability of finding the system in state  $+1$  respectively  $-1$ . We see that for times  $t > 10$  the system is approximately in a static state  $\mathbf{P}^0$ , where the probability doesn't depend on the time any more.

iv) The histograms of the time periods, in which the system is in one state, are shown in figure 6 and 7.

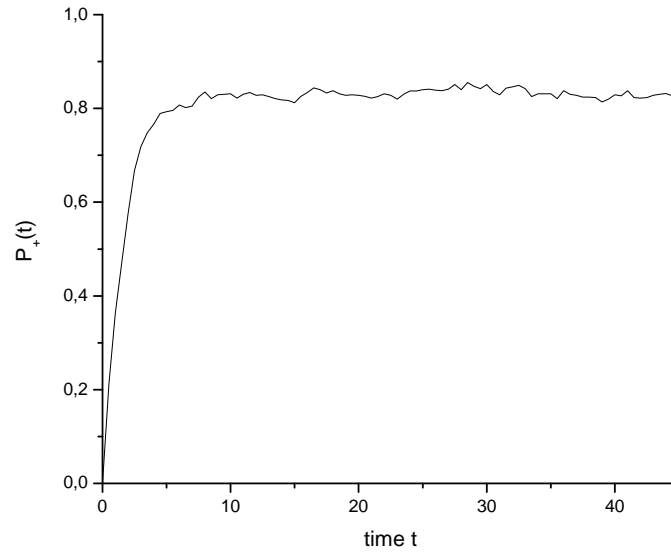


Figure 4: Probability of finding the system in state  $+1$  at time  $t$

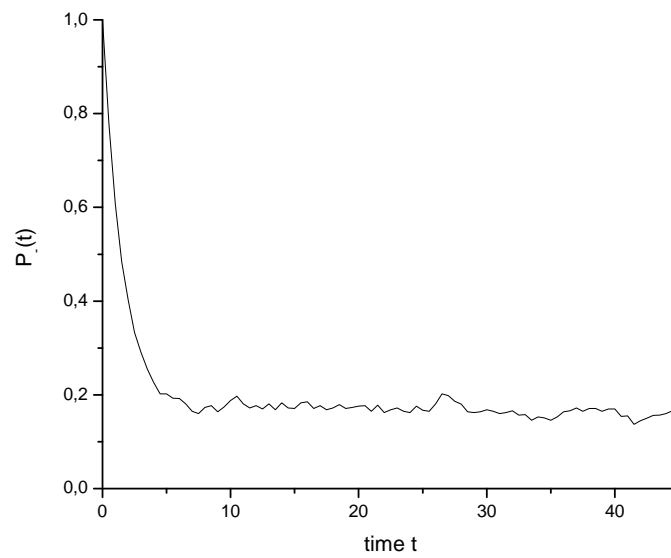


Figure 5: Probability of finding the system in state  $-1$  at time  $t$

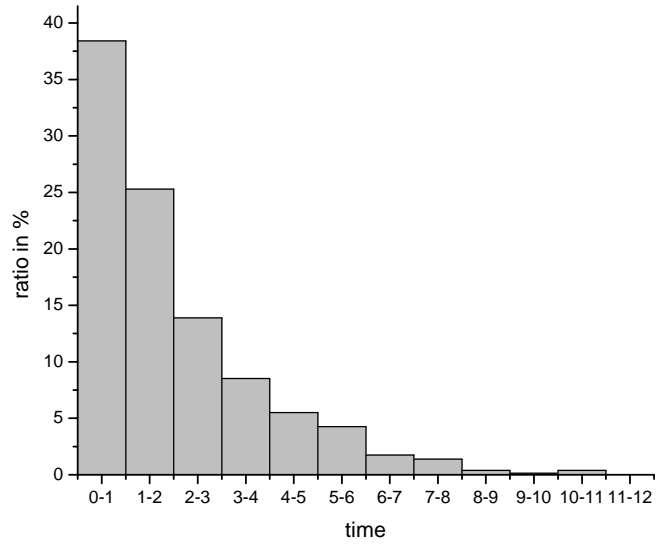


Figure 6: Histogram of the times in state +1

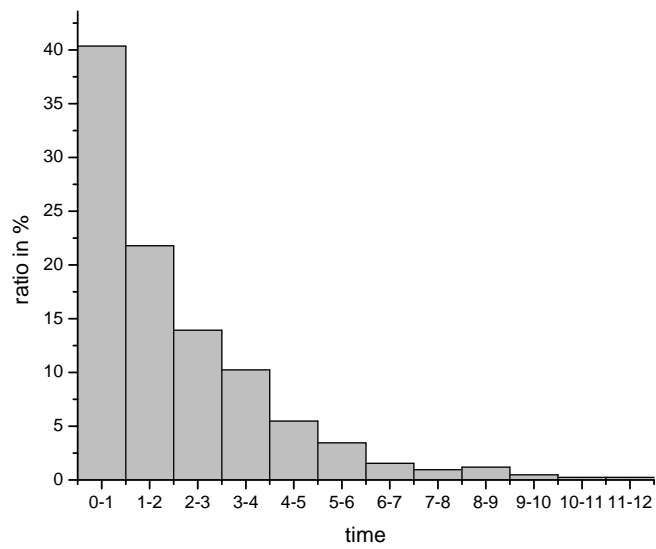


Figure 7: Histogram of the times in state -1

## Discussion

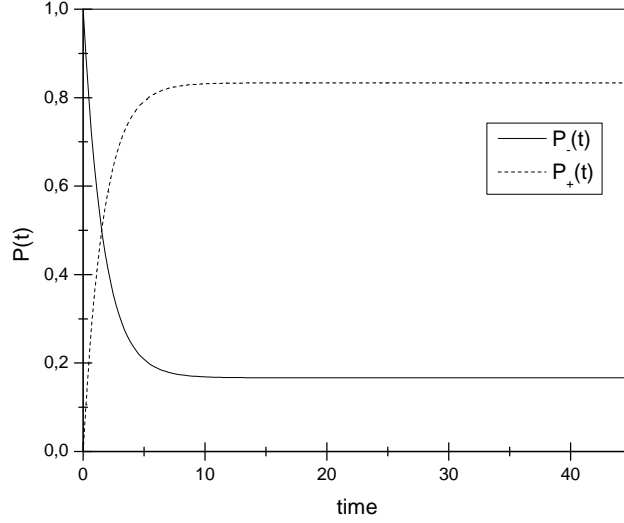


Figure 8: Analytical solution of  $P_+(t)$  and  $P_-(t)$  (see eq. 6) with  $\gamma_{-+} = 0,1$  and  $\gamma_{+-} = 0,5$

Comparing the figures 4, 5 and 8 exposes the good accordance of simulated and analytically computed values.

As mentioned in part b) ii), we executed the belonging process 30 times to get an idea of the reliability of the values. A better way is to implement a computation of the deviation during evaluation  $T$  trajectories. For this, we could use the formula for variance or standard deviation.

## Appendix

a) Insertion of (eq. 3) in (eq. 1) gives us:

$$\begin{aligned}\dot{P}_-(t) &= \gamma_{-+}(1 - P_-(t)) - \gamma_{+-}P_-(t) \\ \Rightarrow \dot{P}_-(t) + (\gamma_{-+} + \gamma_{+-})P_-(t) &= \gamma_{-+}\end{aligned}$$

The solution for  $P_-(t)$  consists of the homogeneous solution  $P_{-, \text{hom}}(t) = f e^{-(\gamma_{-+} + \gamma_{+-})t}$  and particular one  $P_{-, \text{part}}(t) = \frac{\gamma_{-+}}{\gamma_{-+} + \gamma_{+-}}$  as follows:

$$P_-(t) = \frac{\gamma_{-+}}{\gamma_{-+} + \gamma_{+-}} + f e^{-(\gamma_{-+} + \gamma_{+-})t}$$

Whereas  $f$  is the integration constant which has to be calculated on the basis of the initial

condition  $P_-(t=0) = 1$ :

$$\begin{aligned}
P_-(0) &= \frac{\gamma_{-+}}{\gamma_{-+} + \gamma_{+-}} + f = 1 \\
\Rightarrow f &= 1 - \frac{\gamma_{-+}}{\gamma_{-+} + \gamma_{+-}} \\
\Rightarrow P_-(t) &= \frac{\gamma_{-+}}{\gamma_{-+} + \gamma_{+-}} + \left(1 - \frac{\gamma_{-+}}{\gamma_{-+} + \gamma_{+-}}\right) e^{-(\gamma_{-+} + \gamma_{+-})t}
\end{aligned} \tag{4}$$

With (eq. 3) and (eq. 4) we can calculate  $P_+(t)$

$$\begin{aligned}
P_+(t) &= 1 - P_-(t) \\
\Rightarrow P_+(t) &= 1 - \frac{\gamma_{-+}}{\gamma_{-+} + \gamma_{+-}} + \left(\frac{\gamma_{-+}}{\gamma_{-+} + \gamma_{+-}} - 1\right) e^{-(\gamma_{-+} + \gamma_{+-})t}
\end{aligned} \tag{5}$$

The propability distribution is:

$$\mathbf{P}(t) = \begin{pmatrix} P_-(t) \\ P_+(t) \end{pmatrix} = \begin{pmatrix} \frac{\gamma_{-+}}{\gamma_{-+} + \gamma_{+-}} + \left(1 - \frac{\gamma_{-+}}{\gamma_{-+} + \gamma_{+-}}\right) e^{-(\gamma_{-+} + \gamma_{+-})t} \\ 1 - \frac{\gamma_{-+}}{\gamma_{-+} + \gamma_{+-}} + \left(\frac{\gamma_{-+}}{\gamma_{-+} + \gamma_{+-}} - 1\right) e^{-(\gamma_{-+} + \gamma_{+-})t} \end{pmatrix} \tag{6}$$

Now we're looking for the stationary state. It means, that  $\mathbf{P}(t)$  is constant for all  $t$ . For  $t \rightarrow \infty$  this constraint is satisfied and we obtain:

$$\mathbf{P}^0 = \begin{pmatrix} P_-^0 \\ P_+^0 \end{pmatrix} = \begin{pmatrix} \frac{\gamma_{-+}}{\gamma_{-+} + \gamma_{+-}} \\ 1 - \frac{\gamma_{-+}}{\gamma_{-+} + \gamma_{+-}} \end{pmatrix} \tag{7}$$

Now we want to calculate the respective moments of the random variable  $m := \frac{n+1}{2}$ :

$$\begin{aligned}
\langle m \rangle_t &= \left\langle \frac{n+1}{2} \right\rangle_t \\
&= \frac{1}{2} \langle n \rangle_t + \frac{1}{2}
\end{aligned}$$

What we need now is the mean of  $n$ . It's given by:

$$\begin{aligned}
\langle n \rangle_t &= P_-(t)n(-1) + P_+(t)n(+1) \\
&= P_+ - P_- \\
&= 1 - 2P_-(t)
\end{aligned} \tag{8}$$

The mean of  $m$  is:

$$\begin{aligned}
\langle m \rangle_t &= \frac{1}{2}(1 - 2P_-(t)) + \frac{1}{2} \\
&= 1 - P_-(t)
\end{aligned} \tag{9}$$

The same way we got (eq. 9) leads us to the other moment  $\langle m^2 \rangle_t$ :

$$\begin{aligned}
\langle m^2 \rangle_t &= \left\langle \frac{(n+1)^2}{4} \right\rangle_t \\
&= \frac{1}{4} \langle n^2 + 2n + 1 \rangle_t \\
&= \frac{1}{4} [\langle n^2 \rangle_t + 2 \langle n \rangle_t + 1]
\end{aligned} \tag{10}$$

Calculation of  $\langle n^2 \rangle_t$  (with eq. 3):

$$\begin{aligned}
\langle n^2 \rangle_t &= P_-(t)n^2(-1) + P_+(t)n^2(+1) \\
&= P_-(t) + P_+(t) \\
&= 1
\end{aligned}$$

Insertion of this result into (eq. 10) gives:

$$\begin{aligned}
\langle m^2 \rangle_t &= \frac{1}{4} [1 + 2 \langle n \rangle_t + 1] \\
&= \frac{1}{2} + \frac{1}{2} \langle n \rangle_t \\
&= \langle m \rangle_t
\end{aligned} \tag{11}$$

## b) i) Implementation of Gillespie's algorithm

The *Trajectory* class generates and stores a trajectory of the system. The constructor expects the abort time for the calculation and the beginning state. At instantiating a *Trajectory* object, the *Run* procedure, that implements Gillespie's algorithm, is called.

Firstly, an *ArrayList* is defined to store elements of the type *HopItem*. This type stores the time of the hop and the new state. Last thing is on the one hand not essential because in our case the system can only toggle between two states. On the other hand, some parts of the program become more slim as we don't have to worry about the starting state there.

In the whole process, the *current\_state* variable gets assigned the values  $-1$  or  $+1$ . So, defining the *rates* array as done in the code offers a simple way to get the transition rate for *current\_state* by reading *rates[current\_state + 1]*. Note, that the second element of the array is never used.

Since we want to limit the trajectory not by a certain amount of done hops but by a maximum time, define a while loop that breaks as soon as *current\_time* exceeds *maxtime*. Every iteration computes one system change. At first, the *waitingtime* is computed using the transition rate array and the internal random number generator. This specifies the time (from now on) till the system will perform the hop, so its absolute time results from adding *waitingtime* to *current\_time*, what becomes the new *current\_time*.

After this, Gillespie's algorithm normally has to decide, which is the new system state. But this is obvious in our case since it can only be the other remaining state. So, the *current\_state* is simply toggled.

Finally, *current\_time* and *current\_state* are stored in the ArrayList. After executing the *Run* procedure, the array of transitions can be accessed by the public *Result* variable.

In the same class, the function *getState* is defined, that returns the state at the given time. This is needed where the trajectories must be behold as a function of *t*.

```
import java.util.ArrayList;
import java.util.List;

/*
 * Represents a simulated experiment (trajectory)
 */
public class Trajectory {
    // stores the system changes (hops):
    public List<HopItem> Result;
    private int start_state;

    public Trajectory(double maxtime, int startstate) {
        this.start_state = startstate;
        this.Run(maxtime);
    }

    private void Run(double maxtime) {
        List<HopItem> res = new ArrayList<HopItem>();

        // stores transition rates
        double[] rates = { 0.5, 0, 0.1 };
        int current_state = this.start_state;
        double current_time = 0;
        do {
            // compute the time until
            // next system change:
            double waitingtime =
                - 1 / rates[current_state + 1]
                * Math.log(Math.random());
            current_time += waitingtime;

            // toggle state:
            if (current_state == -1)
                { current_state = 1; }
        }
    }
}
```

```

    else
        { current_state = -1; }

    // save this transition to array:
    HopItem i = new HopItem();
    i.time = current_time;
    i.state = current_state;
    res.add(i);
} while(current_time < maxtime);
// return array of transitions:
this.Result = res;
}

/*
 * Returns the state at given time
 * (This must be computed since only system changes
 * are saved but not the states at any time.)
 */
public int getState(double time) { /* ... */ }

public void print() { /* ... */ }

/*
 * Represents a transition. The time
 * and the new state are saved. All
 * these HopItems are saved in List<HopItem> Result.
 */
public class HopItem {
    public int state;
    public double time;
}
}

```

Now, to generate and plot a trajectory with initial state -1 for a time period of about 50s, we need the following code:

```

Trajectory t = new Trajectory(50, -1);
t.print();

```

## ii) Calculation of $P_+(T)$

To get  $P_+(T)$ , we use a for loop to create  $N$  trajectories and check, if the system is in state 1 at time  $T$ . The amount of trajectories, where this is true, divided by the total amount represents the wanted probability of finding the system in state 1 at time  $T$ . This is done by the following code:

```
public static double Task2(double T, int N) {  
    int statesum = 0;  
    for (int a=1; a<=N; a++) {  
        Trajectory t = new Trajectory(T+1, -1);  
        int state = t.getState(T);  
        if (state == 1) {  
            statesum += 1;  
        }  
    }  
    return (double) statesum / N;  
}
```

## iii) Plotting $P_+(t)$ and $P_-(t)$

To make a plot of  $P_+(t)$  and  $P_-(t)$ , the work of ii) must be done for many time steps. We firstly generate  $N$  trajectories ( $N = 1000$ ). Then, we count for every  $t$ , which we increase by  $\delta t \equiv \Delta t$  in a while loop, the trajectories, that are in state 1 respectively  $-1$  (defined by the function parameter *checkstate*) at this time. This is saved to an ArrayList, that contains  $P(t) = P(n \cdot \Delta t)$ .

```
public static List<Double> Task3(double deltaT, double endT,  
                                int N, int checkstate) {  
    List<Trajectory> Trajectories = new ArrayList<Trajectory>();  
    List<Double> Result = new ArrayList<Double>();  
    double current_time = 0;  
    for (int a=0; a<N; a++) {  
        Trajectories.add(new Trajectory(endT+1, -1));  
    }  
    do {  
        int statesum = 0;  
        for (int a=0; a<N; a++) {  
            Trajectory t = Trajectories.get(a);  
            int state = t.getState(current_time);  
            if (state == checkstate) {  
                statesum += 1;  
            }  
        }  
        Result.add((double) statesum / N);  
        current_time += deltaT;  
    } while (current_time < endT);  
}
```

```

    }
  }
  Result.add( (double) statesum / N );
  current_time += deltaT;
} while (current_time <= endT);
return Result;
}

```

#### iv) Plotting the histogram

For this, we generate a (long) trajectory with initial state  $-1$ . Then we go through the trajectory array and save all lengths of time periods in state 1 respectively  $-1$ . This is done by increasing the loop counter  $a$  by 2, so we always get to hops with the same direction.

If we start at  $a = 1$ , this is a  $1 \rightarrow -1$  hop. By subtracting the point in time of the previous hop ( $-1 \rightarrow 1$ ), we get the time period in state 1. In the next loop iteration,  $a$  is increased by 2 and the length of the next period is computed. If we start at  $a = 2$ , the same is calculated for time periods in state  $-1$ . The mentioned start value is given by the parameter *start*.

In fact, not the accurate time is saved. To get a histogram, we count the time periods, that are in a certain interval given by the parameter  $\text{delta}T \equiv \Delta t$ . In the end, the ArrayList *Histogram* contains the amount of periods in the interval  $[0, n \cdot \Delta t[$ , where  $n$  is the array index. By dividing this amount by the total number of considered time periods and multiplying this with 100, we get the ratio in percent. This is used to plot the histograms.

```

public static List<Integer> Task4(double T, double deltaT,
                                int start) {
    List<Integer> Histogram = new ArrayList<Integer>();
    Trajectory t = new Trajectory(T, -1);
    for (int a=start; a<t.Result.size(); a+=2) {
        double elapsedtime = t.Result.get(a).time
                             - t.Result.get(a-1).time;
        int index = (int)Math.floor((double) elapsedtime / deltaT);
        int val;
        // if the ArrayList has less elements than needed to
        // range in the current period, they are added.
        if (Histogram.size()-1 < index) {
            val = 1;
            for (int x=0; x<=index - Histogram.size(); a++) {
                Histogram.add(0);
            }
            Histogram.add(index, val);
        }
    }
}

```

```
    }  
    else {  
        val = Histogram.get(index) + 1;  
        Histogram.set(index, val);  
    }  
}  
return Histogram;  
}
```