9
Numerical simulations of master equations

9.1
The first reaction method.

Given the difficulties one encounters for the analytical treatment of master equations, it is common to resort to numerical simulations of the underlying stochastic process. We will expose in this chapter the basic principles of such a procedure. We first begin by the simple example of a two-state system with constant rates. If we denote by X and Y the possible states there will be jumps from X to Y at a rate $\omega_{1\to 2}$ and from Y to X at a rate $\omega_{2\to 1}$. Remember that, besides being both non-negative numbers, the rates $\omega_{1\to 2}$ and $\omega_{2\to 1}$ need not to have any relation amongst them. The process has been schematized in (8.1).

The stochastic process consists in a series of jumps from one of the two states to the other. Our numerical simulation in this simple case will consist precisely in the generation of these trajectories that jump from X to Y at the adequate times. Imagine that at the initial time t_0 we are at state X. We now need to find the time t_1 at which the particle will jump to Y for the first time. We have already proven in section 8.1 that this time t_1 of the first jump follows the distribution (8.21)

$$f^{\text{1st}}(2,t_1|1,t_0) = \omega_{1\to 2}\mathrm{e}^{-\omega_{1\to 2}(t_1-t_0)}. \tag{9.1}$$

In other words, $t_1 - t_0$ follows an exponential distribution. We learnt in section 2.4 how to generate values of a random variable distributed according to an exponential distribution. Simply generate a $\hat{\mathbf{U}}(0,1)$ random number u_0 uniformly distributed in the interval $(0,1)$ and use $t_1 - t_0 = -\dfrac{\ln u_0}{\omega_{1\to 2}}$, or

$$t_1 = t_0 - \frac{\ln u_0}{\omega_{1\to 2}}. \tag{9.2}$$

Now we are in state Y and want to determine the time t_2 of the next jump to state X. The time t_2 of the first jump follows the distribution (8.22):

$$f^{\text{1st}}(1,t_2|2,t_1) = \omega_{2\to 1}\mathrm{e}^{-\omega_{2\to 1}(t_2-t_1)}. \tag{9.3}$$

Therefore, we can generate t_2 by drawing a $\hat{\mathbf{U}}(0,1)$ random number u_1 from a uni-

form distribution in $(0,1)$ and obtain

$$t_2 = t_1 - \frac{\ln u_1}{\omega_{2\to 1}}. \tag{9.4}$$

Now we are again in state X. To find the time t_3 of the next jump we use the transition rate $\omega_{1\to 2}$ and set

$$t_3 = t_2 - \frac{\ln u_0}{\omega_{1\to 2}}, \tag{9.5}$$

and so on, alternating between the rates $\omega_{1\to 2}$ and $\omega_{2\to 1}$. In this way we can simulate a stochastic trajectory according to the rules of the rates of the process. The next program listing can be used to generate trajectories according to this basic algorithm up to a maximum time `tmax`. The program sets the initial state, 1 or 2, randomly with probability $1/2$, the call to the function `i_ran(2)`.

```
program rate2
  implicit double precision(a-h,o-z)
  tmax=50.0d0
  t=0.0d0
  w12=0.5d0
  w21=1.0d0
  i=i_ran(2)
  write(66,*) t,i
  do while (t < tmax)
    if (i == 1) then
      tn=-dlog(ran_u())/w12
      in=2
    else
      tn=-dlog(ran_u())/w21
      in=1
    endif
    t=t+tn
    write(66,*) t,i
    i=in
    write(66,*) t,i
  enddo
end program rate2
```

The program output (in file 66) is prepared such that a direct plot of this file will show the random trajectories. In figure 9.1 we plot a sample output of this program.

We now consider the more general case that there are M possible individual states labeled by $1, 2, \ldots, M$. Imagine that at time t_0 we are at state i_0. Now there can be jumps to $M-1$ different states with rates $\omega_{i_0\to j}$ for $j = 1, \ldots, M$, $j \neq i_0$. If $\omega_{i_0\to j} = 0$, then the corresponding jump $i_0 \to j$ is not permitted. The idea is to generate $M-1$ independent $\hat{\mathbf{U}}(0,1)$ random numbers u_0^j for $j = 1, \ldots, M$, $j \neq i_0$ and compute the jumping times to every one of these states as:

$$t_{i_0\to j} = \frac{-\ln u_0^j}{\omega_{i_0\to j}}, \qquad j = 1, \ldots, M, \quad j \neq i_0. \tag{9.6}$$

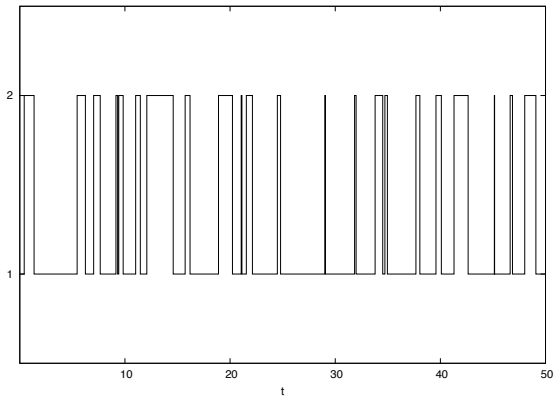

Figure 9.1 Typical trajectory of a stochastic system that jumps between 2 states using the rates $\omega_{1\to 2}=0.5$, $\omega_{2\to 1}=1.0$.

$t_{i_0\to j}$ is the time at which the transition $i_0 \to j$ would occur if there were no other states to which to jump to[1)]. Which transition would actually occur first? Simple: the one with the smallest time $t_{i_0\to j}$. This is called the "first reaction method". Let i_1 be the state with the minimum transition time: $t_{i_0\to i_1} = \min(t_{i_0\to 1}, t_{i_0\to 2}, \cdots, t_{i_0\to M})$. Then, at time $t_1 = t_0 + t_{i_0\to i_1}$ we jump from i_0 to i_1. Now that we are at state i_1 and to determine the state and the time of the next jump. For that, we generate $M-1$ independent $\hat{\mathbf{U}}(0,1)$ random numbers u_1^j for $j=1,\ldots,M$, $j\neq i_1$ and compute the times of possible jumps

$$t_{i_1\to j} = \frac{-\ln u_1^j}{\omega_{i_1\to j}}, \qquad j=1,\ldots,M, \quad j\neq i_1. \tag{9.7}$$

The actual jump $i_1 \to i_2$ is the one that occurs at the earliest of those times: $t_{i_1\to i_2} = \min(t_{i_1\to 1}, t_{i_1\to 2}, \cdots, t_{i_1\to M})$. Then, at time $t_2 = t_1 + t_{i_1\to i_2}$ the state jumps from i_1 to i_2. The process starts again at state i_2 at time t_2.

Here comes a program listing that implements this numerical method. We have decided to use as an example the rates $\omega(i\to j) = |i-j|$, which fulfill the necessary condition $\omega(i\to i)=0$. The rates are now stored in the vector `w(M,M)`.

```
program ratem
  implicit double precision(a-h,o-z)
  parameter (M=1000)
```

1) Note that if $\omega_{i_0\to j}=0$, the corresponding jumping time is $t_{i_0\to j}=\infty$, which is another way of saying that the jumping from i_0 to that particular value of j will never occur. In the programming of this algorithm one needs to avoid the calculation of jumping times for the transitions that are known not to occur, otherwise annoying overflow errors will appear.

```
dimension w(M,M)
do i=1,M
  do j=1,M
    w(i,j)=abs(i-j)
  enddo
enddo
tmax=10.0d0
  i=i_ran(M)
  t=0.0d0
  write(66,*) t,i
  do while (t < tmax)
    tn=1.0d16
      do j=1,M
        if (w(i,j) > 1.0d-8) then
           t1=-dlog(ran_u())/w(i,j)
           if (t1 < tn) then
             tn=t1
             in=j
           endif
         endif
       enddo
       t=t+tn
      write(66,*) t,i
      i=in
      write(66,*) t,i
   enddo
end program ratem
```

In this program, in order to avoid using rates $\omega(i \to j)$ which are equal to 0, the time of the next jump is not computed if $\omega(i \to j) < 10^{-8}$ (a reasonable small number which might need to be checked in other cases) and we set first a big time `tn`$= 10^{16}$ to determine the minimum jumping time in the assumption that it will be smaller than this big amount (again a reasonable assumption whose validity might need to be checked in one particular application). A simple output of this program for $M = 10$ is plotted in figure 9.2

Let us now go back to the case of $M = 2$ states. If we are interested in the numerical determination of $P_1(t)$, the probability that the particle is in state 1 at time $t \in (t_0, t_{\max})$, we would generate a large number N of those individual trajectories and determine in how many of them the particle is in state 1 at time t. As particles are independent, we can as well take a global point of view. Instead of running the above programs N times, we run a system consisting of N particles. In order to compute $P_1(t)$ we would count how many particles $n(t)$ there are in state 1 at time t and set $P_1(t) = n/N$.

There are two ways in which we could simulate the simultaneous trajectories of N particles. In the first one we determine for each one of the N particles the time of the next jump. So, if we start at $t = t_0$ with all particles at state 1 (because we have chosen the initial condition $P_1(t_0) = 1$), then we compute the times of the next jump t_1^k for $k = 1, \ldots, N$. Each of these times is computed as $t_1^k = t_0 - \dfrac{\log u_0^k}{\omega_{1\to 2}}$,

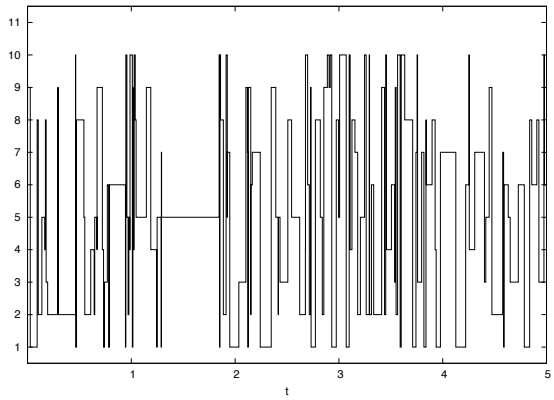

Figure 9.2 Typical trajectory of a stochastic system that jumps between 10 states using the rates $\omega_{i\to j} = |i-j|$.

where u_0^k are a set of N independent $\hat{\mathbf{U}}(0,1)$ uniform random numbers. Next we find the minimum of all these times $t_1 = \min(t_1^1, t_1^2, \ldots, t_1^N)$, say it is $t_1^{k_1}$. At this time $t_1 = t_1^{k_1}$ we place particle k_1 in state 2. We now proceed by computing again the jumping times t_2^k for all $k = 1, \ldots, N$ particles using $t_2^k = t_1 - \dfrac{\log u_1^k}{\omega_{i\to j}}$ where for each particle $k = 1, \ldots, N$ we have to set the correct rate $\omega_{1\to 2}$ or $\omega_{2\to 1}$ depending on which state this particle is at time t_1. Once we have determined the minimum time $t_2 = \min(t_2^1, \ldots, t_2^N)$, say $t_2 = t_2^{k_2}$ we let particle k_2 jump from the state it is to the other. The process continues until the time t_j reaches the desired maximum time.

We now describe the second, more convenient, way of generating trajectories for the N-particle system. As all we need for the calculation of $P_1(t)$ is the number of particles $n(t)$ in state 1 at time t, we adopt the "occupation numbers" point of view and characterize the ensemble not by the state every particle is in, but directly by the number n of particles in state 1 (and $N-n$ particles in state 2). From this alternative point of view, the variable n can take any of the $N+1$ values $n = 0, 1, \ldots, N$. So, we consider that the whole ensemble can be in any of $N+1$ states labeled by the value of n. The situation is formally similar to the M-states case explained before (program `ratem`), but now the $M = N+1$ possible states are collective states of the whole system of N particles, instead of individual state. Furthermore, the problem is simpler than the general M-state case explained before, as the only possible transitions allowed are those that increase (resp. decrease) in one unit the value of n, corresponding to transitions from one particle from 2 to 1 (resp. from 1 to 2). As found in section 8.1.2, eqs. (8.27-8.28) the rate of the transition from n to

$n+1$ is $\Omega(n \to n+1) = (N-n)\omega_{2\to 1}$ and the rate of the transition from n to $n-1$ is $\Omega(n \to n-1) = n\omega_{1\to 2}$. Then, if at time t_0 we are in a global state characterized by n particles in state 1, we have to compute only the time $t_{n\to n+1} = -\dfrac{\log u_0^1}{\Omega(n \to n+1)}$ of the next jump $n \to n+1$ and the time $t_{n\to n-1} = -\dfrac{\log u_0^2}{\Omega(n \to n-1)}$ of the next jump to $n \to n-1$, using two independent $\hat{\mathbf{U}}(0,1)$ uniform random numbers u_0^1, u_0^2, and implement the action implied by the minimum of these two times, $t_1 = t_0 + \min(t_{n\to n+1}, t_{n\to n-1})$, setting $n \to n+1$ or $n \to n-1$ accordingly. The process then repeats itself starting at time t_1 finding the time t_2 and the state of the next jump. We now give an example of a program listing that implements this numerical method. In this listing, we have used that if $n = 0$ (all particles are in state 2) then the only possible transition is towards $n = 1$ and that if $n = N$ (all particles in state 1) the only possible transition is to $n = N - 1$ (the number N is indicated in the program by the variable `N0` as capital and lower-case letters are be mistaken by the compiler) . We have also chosen the initial state with a random number of particles in state 1: the function `i_ran(N0+1)-1` returns a number between `0` and `N0`.

```
program rate2b
  implicit double precision(a-h,o-z)
  N0=1000
  tmax=100.0d0
  t=0.0d0
  w12=0.5d0
  w21=1.0d0
  n=i_ran(N0+1)-1
  write(66,*) t,n
  do while (t < tmax)
    if (n == 0) then
      tn=-dlog(ran_u())/(N0*w21)
      in=1
    elseif (n ==  N0) then
      tn=-dlog(ran_u())/(N0*w12)
      in=N0-1
    else
      tn1=-dlog(ran_u())/((N0-n)*w21)
      tn2=-dlog(ran_u())/(n*w12)
      if (tn1 < tn2) then
        tn=tn1
        in=n+1
      else
        tn=tn2
        in=n-1
      endif
    endif
    t=t+tn
    write(66,*) t,n
    n=in
    write(66,*) t,n
  enddo
```

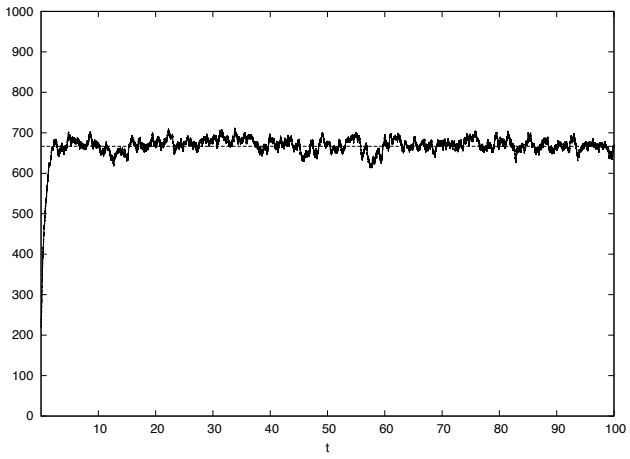

Figure 9.3 Number of particles in state 1 for an ensemble of $N = 1000$ particles that can jump between states 1 and 2 with rates $\omega_{1\to 2} = 0.5$, $\omega_{2\to 1} = 1.0$. The horizontal dashed line is the steady state value $\frac{n_{\rm st}}{N} = \frac{\omega_{2\to 1}}{\omega_{1\to 2}+\omega_{2\to 1}}$, as found in (8.11).

```
end program rate2b
```

In figure 9.3 we plot one output of this program. There is one obvious "practical" advantage of the simulation of the N particles running simultaneously, namely, that it is much easier to perform the averages over the N particles. For example, it is obvious from this figure that a steady state is reached in which $P_1(t)$ (given by the ratio of $n(t)$ to the number of particles, $N = 1000$) fluctuates around value equal to the theoretical prediction $\dfrac{\omega_{2\to 1}}{\omega_{1\to 2} + \omega_{2\to 1}}$.

When the N particles are not independent, i.e when the individual transition rates $\omega_{i\to j}$ depend on the state of the other particles, the simultaneous running of the N particles has another big advantage, namely, that the number of particles in each state is easily accessible. Consider, for example, the chemical reaction discussed in section 8.3. In that example the individual rate for a Na atom to react (to go from state 1, unbound or free, to state 2, bounded to the molecule) depends on the number n of Cl atoms (we take the simplifying assumption that the number n of Na atoms is equal to the number n of Cl atoms at all times): $\omega_{1\to 2}[n] = k_{12}V^{-1}n$. It would be very difficult to program this by finding the times at which individual atoms combine to form a molecule and keeping track of how many atoms n there are bounded at a given time t. Instead, we take the occupation numbers point of view and use the global rates given in (8.56-8.57): $\Omega(n \to n+1) = (N-n)\omega_{21}$ and $\Omega(n \to n-1) = k_{12}V^{-1}n^2$. In the previous program listing all we need is to modify the relevant lines to:

```
if (n ==  0) then
```

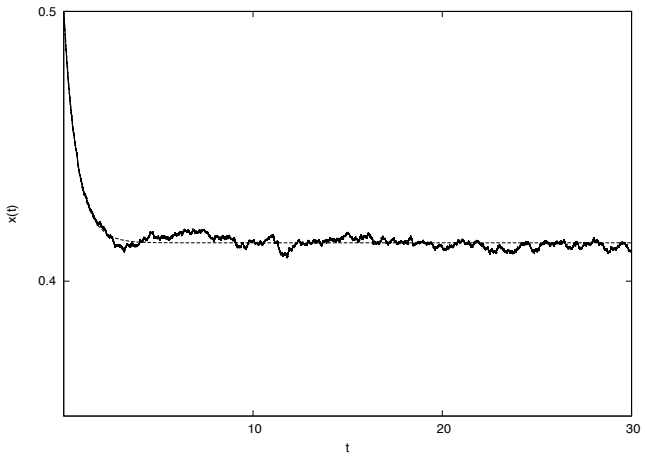

Figure 9.4 Fraction $x(t) = n(t)/V$ of unbound A atoms in the chemical reaction $A + B \rightleftarrows AB$. Parameters: $N = 10^4$, $k_{12} = 0.5$, $V/N \equiv v = 2$, $\omega_{21} = 1$. The initial condition is $n_0/V = 0.5$. The analytical line is the result of the mean-field theory, see exercise 8.6.

```
    tn=-dlog(ran_u())/(N0*w21)
    in=1
  elseif (n ==  N0) then
    tn=-dlog(ran_u())/(N0**2*k12/V)
    in=N0-1
  else
    tn1=-dlog(ran_u())/((N0-n)*w21)
    tn2=-dlog(ran_u())/(n**2*k12/V)
  ..............
```

and define `k12` and `V` somewhere at the beginning of the program. In figure 9.4 we plot the result of this program and compare with the predictions of the mean-field theory (see exercise 8.6).

A simple extension of this algorithm can be used in the case that an individual particle can be in $M > 2$ states. All we need to do is to specify the occupation numbers for particles in each state $(n_1, \ldots, n_M)$ and the possible rates $(n_1, \ldots, n_M) \to (n'_1, \ldots, n'_M)$. As the individual rate to go from state i to state j is $\omega_{i\to j}$, the rate at which the global transition $(n_1, \ldots, n_i, \ldots, n_j, \ldots, n_M) \to (n_1, \ldots, n_i - 1, \ldots, n_j + 1, \ldots, n_M)$ occurs is $n_i\omega_{i\to j}$ as any of the n_i particles in state i can make the jump to state j. The total number of possible transition rates $\Omega((n_1, \ldots, n_M) \to (n'_1, \ldots, n'_M))$ for the global system is $M(M-1)$, which can be a very large number. In practice, however, not all transitions are permitted by the

rules of the process. Instead of giving now a specific example, we will explain first a modification that leads to a much more efficient programming of the numerical simulations.

9.2 The residence time algorithm.

There is a very simple but very effective modification of the numerical algorithm to simulate a stochastic process and it associated master equation. It bears different names (residence time, kinetic Monte Carlo, n-fold way or Bortz-Kalos-Lebowitz, dynamic Monte Carlo or Gillespie algorithm, etc.) as it has been derived independently a number of times with minor variations and emphasizing different applications in each case.

We first take the point of view of considering only one particle. It can be in any of M states and we need to determine the state to which it will jump next and the time of jump. Assume that, as before, the particle is in the state i_0 at time t_0. First, one computes the rate of escape from this state i_0 to **any other state** $j \neq i_0$. This is nothing but $W_{i_0} = \sum_{j \neq i_0} \omega_{i_0 \to j}$. Then one computes the time interval to the next jump $t_{i_0 \to i_1}$ using this total rate:

$$t_{i_0 \to i_1} = \frac{-\ln u_0}{W_{i_0}}. \tag{9.8}$$

Once the time of the next jump has been determined as $t_1 = t_0 + t_{i_0 \to i_1}$ then we have to determine where to jump, or which is the final state i_1. Recall that $\omega_{i \to j} dt$ is the probability of jumping from i to j in the time interval $(t, t + dt)$ whereas $W_i dt$ is probability of jumping to any state during that same time interval. Therefore the probability $p_{i_0 \to j}$, of reaching state $j \neq i_0$ knowing that there has been a jump is the conditional probability

$$p_{i_0 \to j} = \frac{\omega_{i_0 \to j}}{W_{i_0}}. \tag{9.9}$$

In order to determine the final state i_1 according to these probabilities, we use the general technique explained in section 2.4 to generate the discrete distribution (1.18): draw a random number v_0 uniformly distributed in the interval $(0, 1)$ and find the smallest i_1 that satisfies $\sum_{j=1}^{i_1} p_{i_0 \to j} > v_0$, or equivalently, $\sum_{j=1}^{i_1} \omega_{i_0 \to j} > v_0 W_{i_0}$.

All these ideas can be implemented using the following program, where we use again, as an specific example, the rate $\omega_{i \to j} = |i - j|$.

```
program ratemg
  implicit double precision(a-h,o-z)
  parameter (M=10)
  dimension w(M,M),wt(M)
  do i=1,M
    wt(i)=0.0d0
    do j=1,M
```

```
      w(i,j)=abs(i-j)
      wt(i)=wt(i)+w(i,j)
    enddo
  enddo
  tmax=10.0d0
  t=0.0d0
  i=i_ran(M)
  write(66,*) t,i
  do while (t < tmax)
    tn=-dlog(ran_u())/wt(i)
    p=0.0d0
    j=0
    r=ran_u()*wt(i)
    do while (r > p)
    j=j+1
    p=p+w(i,j)
    enddo
    t=t+tn
    write(66,*) t,i
    i=j
    write(66,*) t,i
  enddo
end program ratemg
```

We store in vector `wt(i)` the total rate W_i to scape from state i. The results of this program look, as they should, similar to those displayed in figure 9.2. Again, in order to obtain meaningful statistics to compute, for example the probability $P_1(t)$ of being in state 1, we should run that program a large number N of times and average results. Alternatively, we could the residence time algorithm to run the N independent particles simultaneously, as we did before.

The second option is to consider the occupation numbers. This method has the advantage that it can also be used if particles are interacting such that the rates depend on the state of other particles. We take now the point of view that there are N (possibly interacting) particles and to consider the full set of occupation numbers variables $(n_1, \ldots, n_M)$ that give the number of particles n_k which are on each of the possible states $k = 1, \ldots, M$. These variables will change (typically by a small amount) and the rates of the transitions $(n_1, \ldots, n_M) \to (n'_1, \ldots, n'_M)$ will depend on the variables $(n_1, \ldots, n_M)$ themselves, although, typically, not many of these transitions will be allowed. We will give details of the method by using the example of the Lotka-Volterra prey-predator model introduced in section 8.3.

In the Lotka-Volterra model the required variables are the number n_1 of live prey and the number n_2 of live predators. In the space (n_1, n_2), and according to the rules of the model, there are three possible transitions whose rates are given in (8.74-8.76). For a given state (n_1, n_2) we compute the total scape rate

$$\begin{aligned} W(n_1,n_2) \quad = \quad & \Omega\left((n_1,n_2)\to(n_1+1,n_2)\right) \\ & +\Omega\left((n_1,n_2)\to(n_1,n_2-1)\right) \\ & +\Omega\left((n_1,n_2)\to(n_1-1,n_2+1)\right). \end{aligned} \tag{9.10}$$

As discussed before, the time to the next transition will be $-\log(u)/W(n_1,n_2)$ being u a $\hat{\mathbf{U}}(0,1)$ random number. Once this transition time has been found, next step is to decide which one of the three possible transitions will happen. Each one has a probability:

$$p_1 = \Omega\left((n_1,n_2)\to(n_1+1,n_2)\right)/W, \tag{9.11}$$

$$p_2 = \Omega\left((n_1,n_2)\to(n_1,n_2-1)\right)/W, \tag{9.12}$$

$$p_3 = \Omega\left((n_1,n_2)\to(n_1-1,n_2+1)\right)/W, \tag{9.13}$$

and we chose transitions 1, 2 or 3 according to these probabilities. Here comes a full program that implements this algorithm for the Lotka-Volterra model.

```
program LotkaVolterra
  implicit double precision(a-h,o-z)
  double precision k1
  tmax=100.0d0
  t=0.0d0
  V=1000
  w0=0.5d0
  k1=1.0d0
  a=k1/V
  w2=0.5d0
  dt=0.1d0
  n1=0.5d0*V
  n2=0.25d0*V
  write(66,*) t,dble(n1)/V,dble(n2)/V
  tw=0.0d0
  do while (t < tmax)
    omega1=w0*n1
    omega2=w2*n2
    omega3=a*n1*n2
    omega=omega1+omega2+omega3
    tn=-dlog(ran_u())/omega
    t=t+tn
    r=ran_u()*omega
    if (r < omega1) then
      n1=n1+1
    else if (r < omega1+omega2) then
      n2=n2-1
    else
      n1=n1-1
      n2=n2+1
    endif
    if (t-tw > dt) then
      write(66,*) t,dble(n1)/V,dble(n2)/V
      tw=t
    endif
  enddo
end program LotkaVolterra
```

In this particular code, the initial condition is set to $n_1(0)=0.5V$, $n_2(0)=0.25V$. Note the presence of the variable `dt` which controls the minimum time between

the writing of the values of the variables in unit `66`. The only reason to include this variable is that the time between transitions `tn` is very small, mainly for large values of V, and hence we end up with a lot of data that has too much detail. It is enough for most applications to have data spaced a larger time `dt`. The results of this program can be seen in figure 9.5 for increasing values of the volume V. Note how both variables giving the concentration of prey $x_1(t) = n_1(t)/V$ and predator $x_2(t) = n_2(t)/V$ oscillate, but there is a delay between them. This is because when the density of prey is very large, the predators find a lot of food and can increase their offspring by eating the prey. This decreases the density of prey and induces that some time later, the predator will find less food and will start decreasing in number. Then, as there are less predators, the prey can increase again. These oscillations in the numbers of prey-predators are well known in ecological populations.

The results of the simulation are compared against the result of the mean-field theory, Eqs. (8.107)-(8.108). These equations can not be solved explicitly in terms of simple functions to give $x_1(t)$ and $x_2(t)$, but it is not so difficult to prove that there is a constant of motion, see exercise 87, which allows one to obtain the closed curve in the (x_1, x_2) plane, as depicted in the right panels of figure 9.5. For large volume V the mean-field solution represents well the simulation results, but as V decreases the trajectories, while still oscillatory, depart more and more from the mean-field solution.

Further reading and references

The name "first reaction method" appears in [39]. The first versions of the residence time algorithm (not necessarily with that name) can be found in [40, 41, 42]. Kirman's model was first proposed in [43].

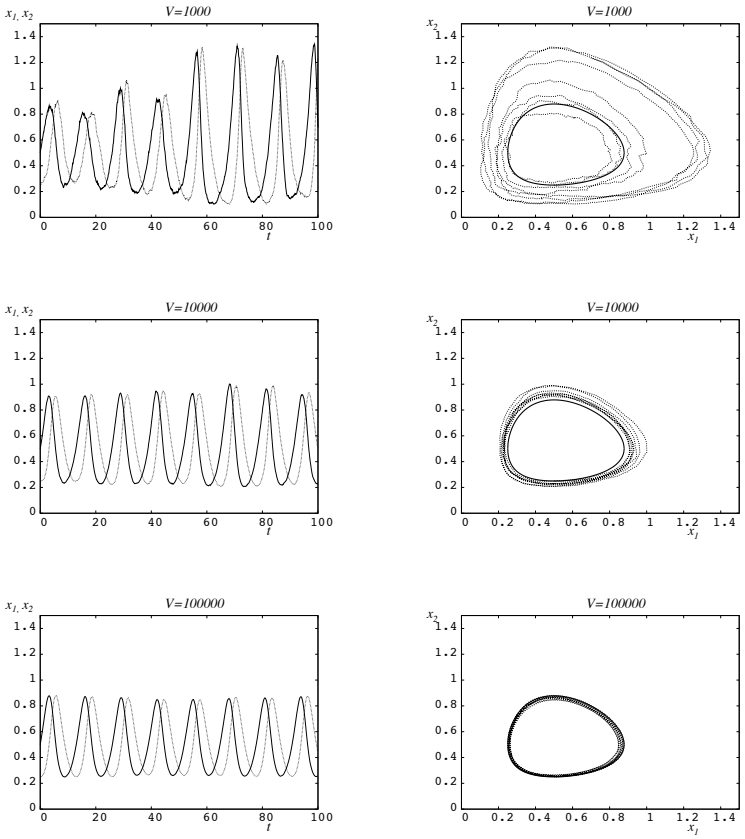

Figure 9.5 Trajectories obtained from the application of the residence time algorithm to the Lotka-Volterra model. In the left panels, we plot the prey variable $x_1(t)$, dashed line and the predator variable $x_2(t)$, solid line. In the right panels we plot the trajectories in the (x_1, x_2) phase space, indicating by a solid thick line the result of the mean-field theory. Top row $V = 10^3$, middle row $V = 10^4$, bottom $V = 10^5$. Parameters: $\omega_0 = 0.5$, $\omega_1 = 0.5$, $k_1 = 1$, initial condition $x_0 = 0.5$, $y_0 = 0.25$.

Exercises

1) Implement the first reaction method and the residence time algorithm to study the birth (from a reservoir) and death process of section 8.3. Start with $n = 0$ particles and compute the time evolution of the average value $\langle n(t)\rangle$ and its variance $\sigma^2[n(t)]$. Check with the theoretical results obtained by the use of the generating function. Compare the efficiency of both methods as a function of the rates Ω_A and ω.
2) Implement the residence time algorithm to compute the evolution of the average number of molecules in the chemical reaction $A + B \rightleftarrows AB$. Compare with the result of the mean field theory.
3) As a follow up to problem 8.8, use the residence time algorithm to compute the average and the variance of the number of intermediate molecule concentration and study under which circumstances (values of the rates) it can be considered a constant as assumed in the Michaelis and Mentis theory.
4) Implement the residence time algorithm to study the Lotka-Volterra prey-predator model. Check that the fluctuations in the trajectories scale as $V^{-1/2}$, being V the total area accessible to the population.
5) Use the results of exercise 8.2 to derive an algorithm that generates the times of the next reaction in the case of a time-dependent rate of the form $\omega(t) = \omega_0 \sin^2(t)$.
6) Consider the SIR model for epidemics spreading adding a spontaneous contagion rate $S \stackrel{\alpha}{\longrightarrow} I$ with a time dependent rate $\alpha(t) = \alpha_0 \sin^2(\omega t)$, modeling the seasonal variations of the presence of the virus causing the illness. Use the first reaction method and the results of the previous exercise and write a program to simulate this process. Study the variations of the number of infected people with time.
7) Consider a system composed by N particles. Each particle can be in each of two states and can make random jumps from one state to the other with a rate which is linear with the number of particles in the other state. Therefore, if there are n particles in state 1, the individual rate to jump from 1 to 2 is $a_1 + h_1(N - n)$ and the rate to jump from 2 to 1 is $a_2 + h_2 n$. Write down the evolution equations for the first two moments $\langle n(t)\rangle$, $\rangle n^2(t)\rangle$ and solve them (making the mean-field assumption if necessary). Use the residence time algorithm and compare the results for the moments and the probabilities $p_{\rm st}(n)$ in the steady state using $a_1 = a_2 = 1$ and $h_1 = h_2 = h$, analyzing separately the cases $h = 1$, $h < 1$ and $h > 1$. This is Kirman's model for herding behavior in financial markets, inspired by the behavior of ants who can chose between two paths when searching for food.