Chapter 9

Modeling Stochastic Processes

Dart throwing computer

Imagine you are standing in front of a large dartboard with a horizontal line drawn on it. The total length of the line is L. The line is divided into two segments – segment A and segment B.

You have thousands of darts, and you are throwing those darts blindly to the board. The darts can hit only the line on the dartboard, nowhere else.

You have thrown N darts, where N is very large. Let N_A and N_B are the number of darts on segment A and segment B of the line. Can we calculate the length of segment A and B from this dart count?

Yes, we can. As you are throwing the darts blindly, all the points on the line have the same chance of being hit by a dart. The probability that a dart will hit anywhere on segment A should be equal to the relative size of this line segment,

$$p_A = \frac{N_A}{N} = \frac{L_A}{L}$$

here, L_A is the length of segment A.

So the length of segment A is,

$$L_A = \frac{N_A}{N} L = p_A L$$

We want to simulate this dart throw experiment on a computer. We tell the computer that the probability of hitting segment A is p_A. Throw the dart N times and count how many times (N_A) the dart hits segment A.

The computer does not need to throw real darts for this experiment. Instead, we will take help of random numbers. Draw an imaginary line for the interval [0, 1]. The length of this line is L =1. As $L_A = p_A L$, consider 0 to p_A as segment A (blue part in Figure 1).

Dart throwing is unbiased. Two line segments of the same length have the same probability of a hit by a dart. So take the help of uniformly distributed random numbers. Ask the computer to generate random numbers one after another following $U(0, 1)$. Each random number is one dart.

Suppose the first random number is u_1 and $u_1 \leq p_A$ (Figure 1). So this random number or dart is in segment A. The computer generates the second random number u_2. This time $u_2 > p_A$. So this dart is in segment B. Keep iterating this procedure for N number of random numbers. Keep a count of random numbers or darts in segment A. That count is N_A.

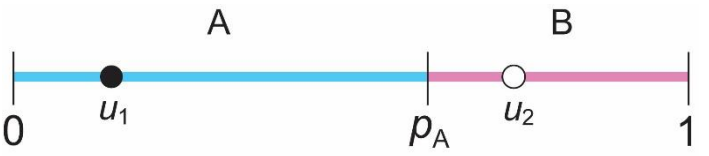


Figure 1: The strategy of simulation of dart throw. A line of unit length is divided into two segments – A (blue) and B (pink). The length of segment A is equal to p_A. u_1 and u_2 are two independent random numbers drawn from $U(0, 1)$.

Figure 2 shows the result of throwing ten darts, where the probability of hitting segment A is p_A = 0.65. On repetition, the result of our simulation will change. That also happens in real dart throwing. Dart throwing is a stochastic process, and results vary; so is the result of its simulation.

Simulating coin toss

Using the binomial distribution, we can calculate the probability of a particular outcome in a sequence of coin tosses. For example, we can calculate the probability of 6 *heads* in 10 tosses of a fair coin. However, suppose you do not want to use the Binomial distribution but want to calculate the probability from experimental data.

u	$u \le p_A$	A or B
0.5021	Yes	A
0.004	Yes	A
0.447	Yes	A
0.1559	Yes	A
0.907	No	B
0.7723	No	B
0.1916	Yes	A
0.6876	No	B
0.5631	Yes	A
0.5154	Yes	A

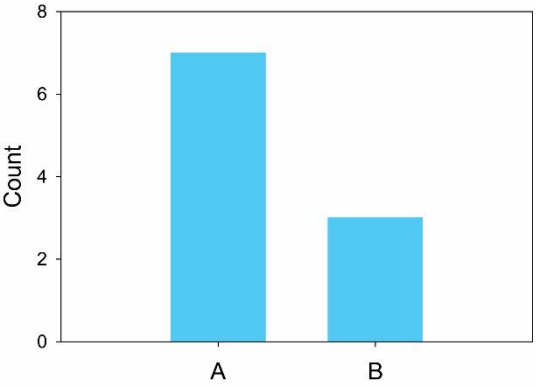


Figure 2: Simulation of the dart throw. The table shows the random numbers $u \sim U(0, 1)$ generated sequentially for the simulation. $p_A = 0.65$. The graph shows the count of A and B in 10 dart throws.

For that, you have to repeat the 10 tosses thousands of times and calculate the frequency of 6 heads in 10 tosses in those thousands of repetitions. Considering that frequency is equivalent to probability for a large number of observations, this experiment will provide the probability 6 *heads* in 10 tosses. One can draw a frequency histogram from this experiment, which would be the same as the PMF of the binomial distribution.

However, will you agree to repeat ten coin tosses thousands of times? Maybe, only if you are stranded on a desolated island with nothing else to kill your time.

This is where a computer is helpful. After all, computers are good at repetitive tasks. Moreover, if it can throw virtual darts, why not coin toss?

Suppose the probability of H is $p_H = 0.3$. The coin will be tossed $n = 10$ times, and the number of *heads* will be counted.

Repeat 10 coin tosses $N = 10000$ times. From this data, calculate the frequency of N_H number of *heads* in 10 tosses. $N_H = 0, 1, 2, \cdots, 10$. Plot the frequency histogram.

To simulate a coin toss, we will use a strategy similar to the one used for dart throw. Take a line of length 1 representing the interval [0, 1]. Mark p_H on it. The line segment from zero to p_H is the region for *head*. The rest of the line is for *tail*.

Generate, $u \sim U(0, 1)$. If $u \le p_H$, u is in the line segment for *head*. So we say that we have got a *head*. Otherwise, it is a *tail*.

Repeat this another nine times. Each random number is a coin toss. Count how many *heads* we got in 10 tosses. Repeat the whole process 10000 times. Calculate the frequency for $N_H = 0, 1, 2, \cdots, 10$.

We can write this method as an algorithm:

```
for i = 1 to N
      #Array for count of heads in n tosses
      H = [H0, H1,⋯, Hn] = [0, 0,⋯, 0]

      # One set of n coin tosses
      Number of heads NH = 0
      for j = 1 to n
            Generate u ~ U(0, 1)
            if u ≤ pH
            NH = NH + 1
      end

      # Update the heads count array
      H[NH]= H[NH] + 1

      # Calculate frequency of heads in N set of n tosses
      f = [f0, f1,⋯, fn] = [H0, H1,⋯, Hn]/N
end
```

Notice that the computer does not need to draw a line and then divide it into two segments. We are using the line to understand the logic of the algorithm. The outcome of a toss can be simply decided based on the logical test of $u \leq p_H$.

The blue-colored bar plot in Figure 3 shows the result of our simulation. Pink circles show the probabilities calculated manually using the PMF of the binomial distribution. The computer has done an excellent job.

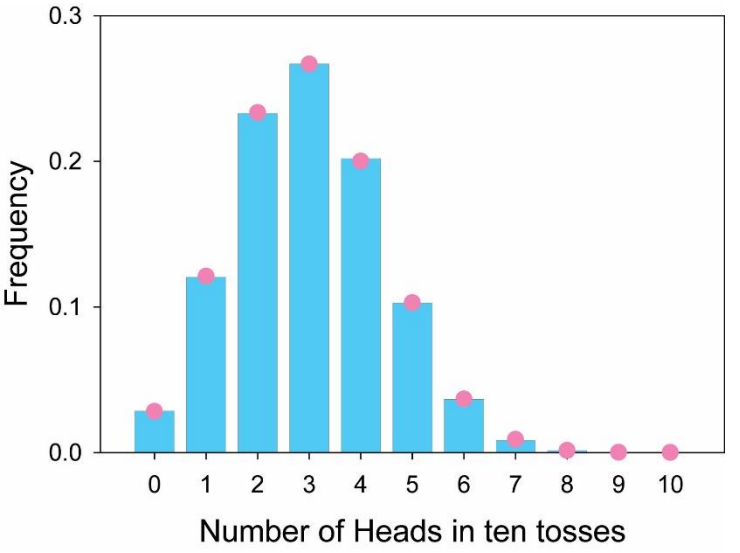


Figure 3: Simulation of the coin toss. p_H = 0.3, number tosses n = 10, number of repetition N = 10000. The blue bar plot shows the result of the simulation. Pink circles are for the corresponding probabilities calculated using the PMF of the binomial distribution.

We can use the same approach for simulation of a multinomial problem, like dice throw. Suppose the probabilities of six faces of a die are p_1, p_2, p_3, p_4, p_5, and p_6. Divide the interval from 0 to 1 in 6 segments, each having a length equal to these probabilities (Figure 4).

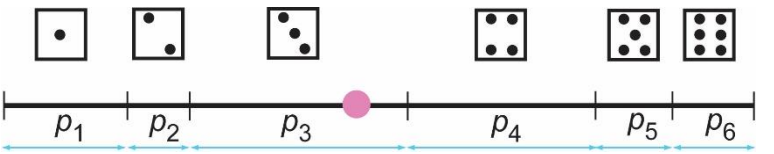


Figure 4: Strategy for simulation of a six-sided die throw. The interval [0, 1] is divided into segments with lengths equal to the probabilities of six sides, p_i, i = 1 to 6. $u \sim U(0, 1)$. The pink circle shows the position of u.

Generate $u \sim U(0, 1)$. The position of u on the interval [0, 1] decides the die throw outcome. For example, in Figure 4, the random number u is in the interval of face 3. So we consider that the outcome of this die throw is 3. Mathematically this logic can be represented as $p_1 + p_2 < u \leq p_1 + p_2 + p_3$.

In general, we will consider that the i^{th} face has appeared, if

$$\sum_{j=0}^{i-1} p_j < u \leq \sum_{j=0}^{i} p_j$$

here, $p_0 = 0$ and $i = 1$ to 6

From dart throwing to dice throw, we are using random numbers to simulate discrete events. This approach belongs to a broad class of algorithms known as Monte Carlo methods, where random sampling is done to solve a problem numerically. Monte Carlo is used in various problems starting from simulation of stochastic processes to integration of complicated functions.

Earlier, we have considered that transcription of a gene is equivalent to a coin toss. Therefore we can simulate transcription using the Monte Carlo method for the coin toss. We can map many other dynamical processes in biology to a coin toss or dice throw. However, time is continuous. Discretization of time may simplify a problem but will add errors in our estimates. So we will refrain from such a discretization of time. We will use random sampling but using a completely different strategy.

Poisson Process

Suppose we are counting mRNAs produced by transcription of a gene. The rate of transcription is λ. The mRNA count at time t is $N(t)$. Take a small time interval $(t, t + \Delta t]$. What would be the probability that n number of mRNAs will be produced in this interval?

To answer this question, let us make some assumptions. First, we assume that at any moment of time, either an mRNA is produced or nothing happens. More than one mRNA are not produced simultaneously.

Our second assumption is that the rate of transcription does not change with time. We call this the assumption of time homogeneity.

Our third assumption is that number of mRNAs produced in a time interval is independent of the number of mRNAs produced earlier. That is the assumption of independent increment.

With these three assumptions, let's calculate the probability. If we consider that transcription is a deterministic process, the number of mRNAs produced in this interval would be $\lambda \Delta t$.

However, transcription is a stochastic process. The average behavior of a stochastic process is equivalent to the deterministic behavior. So, the mean number of mRNA produced in the interval $(t, t + \Delta t]$ will be $\mu = \lambda \Delta t$.

Now, we use the analogy of coin toss. We divide the interval $(t, t + \Delta t]$ in N small interval h such that $N \to \infty$, and $h \to 0$. Suppose p is the probability of production of one mRNA in one such infinitesimally small interval h.

Earlier, we have shown that with $N \to \infty$ binomial distribution becomes a Poisson distribution. So the probability that n number of mRNA will be produced in the interval $(t, t + \Delta t]$ is,

$$P\left(N\left(t+\Delta t\right)-N(t)=n\right)=\frac{\mu^n}{n!}e^{-\mu}=\frac{\left(\lambda\Delta t\right)^n}{n!}e^{-\lambda\Delta t} \tag{1}$$

So in our simplified model of transcription, the probability distribution of the number of mRNAs produced in an interval follows the Poisson distribution. Many other dynamical processes have this property. All such processes are called Poisson processes. We will take the clues from our model for transcription and defines a generalized Poisson process.

A stochastic process is called a counting process $\{N(t), t \geq 0\}$ such that the count is non-negative, integer, and non-decreasing:

$$N\left(t\right)\geq 0$$

$$N(t)\in \square_{\geq 0}$$

$$N\left(t\right)\geq N\left(s\right), \text{ when } t \geq s$$

Here, $N(t)$ is the count till time t, and $N(t) - N(s)$ is the count in the interval $(s, t]$. By count, we mean count of a stochastic event – like count of mRNAs produced or count of molecules moved to nucleus from cytoplasm or count of cells died upon drug treatment.

A Poisson process is a counting process with the rate λ for $\lambda > 0$, such that

1. $N\left(0\right)=0$

2. It has independent increments.

3. The count for an interval t follows Poisson distribution,

$$P\left(N\left(t+s\right)-N(s)=n\right)=\frac{\left(\lambda t\right)^{n}}{n!}e^{-\lambda t}$$

for $n = 0, 1, 2, \ldots$; $s, t \geq 0$.

When λ is independent of time, the Poisson process is called a homogeneous Poisson process. We will deal with only time homogenous systems. So we drop the term 'homogenous' while discussing homogeneous Poisson processes.

Following the definition of the Poisson process, our model for transcription is a Poisson process. Suppose we are counting the mRNAs as they are produced. Just now, I have seen one new copy of the mRNA. How long do I have to wait for the next one?

As transcription or, in general, any Poisson process is a stochastic process, the time interval between two events or counts must also be a random variable. Let T_n be the time interval between the n-th event and the $(n-1)$-th event, for $n > 0$. T_n is called the inter-arrival time. What would be the probability distribution of T_n for a Poisson process?

We will calculate the probability distribution for T_1. T_1 is the first arrival time. Before T_1 count is zero. So, the probability that T_1 is greater than an arbitrary time t must be equal to the probability that no events have happened by time t:

$$P\left(T_1 > t\right) = P\left(N(t)=0\right)$$

As this is a Poisson process with rate λ,

$$\begin{aligned} P\left(T_1 > t\right) &= P\left(N(t)=0\right) \\ &= \frac{(\lambda t)^0}{0!}e^{-\lambda t} \\ &= e^{-\lambda t} \end{aligned}$$

We know $P\left(T_1 \leq t\right) + P\left(T_1 > t\right) = 1$. Therefore, $P\left(T_1 \leq t\right) = 1 - e^{-\lambda t}$.

By definition $P(T_1 \le t)$ is the CDF of T_1, and as we calculated, its CDF is the same as the CDF of the exponential distribution. Therefore, T_1 must be following the exponential distribution Exp(λ) with the PDF $f_{T_1}(t) = \lambda e^{-\lambda t}$.

T_2 is the inter-arrival time for the second event/count. Remember that in a Poisson process, events are independent. Once the first event has happened, we can reset our clock to zero and wait for the second event. As we have reset the clock, T_2 is equivalent to T_1. So T_2 should also follow the exponential distribution Exp(λ).

So, in general, the inert-arrival time T_n of a Poisson process with the rate λ will follow the exponential distribution Exp(λ).

T_n is the inter-arrival time of the n-th count/event. Let S_n be the waiting time for the n-th event. So $S_n = T_1 + T_2 + \cdots + T_n$. Each of the inter-arrival times is an independent random number following the exponential distribution Exp(λ).

It can be shown that summation of n independent random variables following an exponential distribution Exp(λ), has the Gamma distribution Gamma(n, λ). So the waiting time for the n-th count will follow Gamma(n, λ) with the PDF,

$$f_{S_n}(t) = \frac{\lambda}{(n-1)!}(\lambda t)^{n-1} e^{-\lambda t}$$

The mean and variance of the waiting time are $E(S_n) = \frac{n}{\lambda}$ and $\text{var}(S_n) = \frac{n}{\lambda^2}$, respectively.

Simulation of a Poisson process

We can use the Monte Carlo method to simulate a Poisson process. In the simulation of a coin toss, we used random numbers to decide the outcome of a toss. For the Poisson process simulation, we will use a random number to decide when an event will happen.

Take the example of transcription again. Consider transcription is a Poisson process. At any moment, either an mRNA is produced, or nothing happens. The rate of transcription is λ. We have to simulate the process till time T. At t = 0 and total number of mRNA $N(0)$ = 0. Suppose the first mRNA is produced at time t_1. The second mRNA is produced at the

time ($t_1 + t_2$). Similarly, the n-th mRNA is produced at ($t_1 + t_2 + \cdots + t_n$). Here t_1, t_2, …, t_n are the inter-arrival times, and these are independent random numbers that follow Exp(λ).

To simulate transcription, generate a random number $\tau \sim$ Exp(λ). That is t_1. Change the clock from zero to t_1 and increase the count of mRNA to one from zero. Again generate a random number $\tau \sim$ Exp(λ). This random number is t_2. Change the time from t_1 to ($t_1 + t_2$) and increase the mRNA count to two from one. Keep iterating these steps as long as the total time is less-equal to T.

We can write this method as an algorithm:

```
#Initial values
t = 0
N = 0

#Transcription
while t ≤ T
      Generate τ ~ Exp(λ)
      t = t + τ
      N = N + 1
End

#Discard the time-point beyond T
t = T
N = N -1
```

We simulated transcription for λ = 20 per hr. Total time T = 2 hr. The result is shown in Figure 5. The table in Figure 5a shows the data for every step of the simulation. The total number of mRNA produced in 2 hr is 32.

We are doing a stochastic simulation. Therefore repetition of this simulation will yield a different result. Figure 6 shows the results of four independent simulations of the same problem. Like a stochastic process, the result of a stochastic simulation will also have fluctuation around the average behavior. We can repeat the simulation many times (usually thousands of times) and then calculate the average behavior. The average temporal dynamics will be equivalent to the expected deterministic behavior. The gray line in Figure 6 shows the deterministic result for this problem – $N = \lambda t$.

a)

$\tau \sim Exp(\lambda)$	Total time (t)	Total Number of mRNA (N)	
	0	0	
0.06369528	0.063695275	1	
0.1212625	0.184957779	2	= Last N + 1
0.10058467	0.285542451	3	
0.00224856	0.287791016	4	= Last t + τ
0.0103069	0.298097913	5	
⋮	⋮	⋮	
0.01339332	1.799519222	30	
0.01586438	1.815383606	31	t > T
0.16537007	1.980753678	32	Stop & discard
0.08193517	2.062688843	33	this step

b)

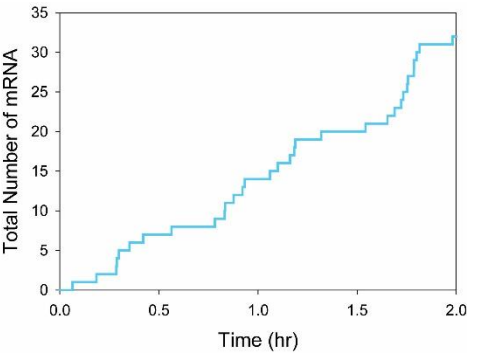


Figure 5: Simulation of transcription as a Poisson process. Rate $\lambda = 20$ per hr. Total time $T = 2$ hr. a) shows the data of simulation. The first coulm has random numbers following the exponential distribution Exp(λ). The rule for calculations in two other coumns are explained in the figure. Data of the last row was discarded as time $t > T$. b) shows the data as a step plot.

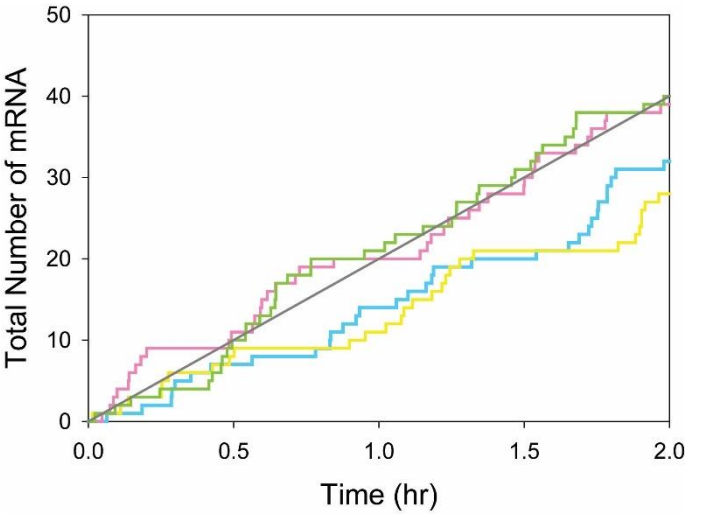


Figure 6: Multiple independent simulations of transcription as a Poisson process. Rate $\lambda = 20$ per hr. Total time $T = 2$ hr. The colored lines are for stochastic simulations. The grey line is the deterministic result.

Generating random numbers

Random numbers are at the heart of any Monte Carlo method. Thankfully, on a computer, we can easily generate random numbers of many probability distributions. Simulation of coin-toss or dice throw requires uniformly distributed random numbers. Common programming languages have library functions to generate uniformly distributed random number $u \sim U(0, 1)$. You can even generate $u \sim U(0, 1)$ in MS Excel using the rand() function.

There are several well-known algorithms to generate $u \sim U(0, 1)$, like Linear Congruential Generators Multiple Recursive Generators and Subtractive Generator. These algorithms are deterministic. They start with a 'seed' number and generate a sequence of random numbers. If we use the same seed, the algorithm will generate the same sequence of numbers.

A sequence of numbers generated by a random number generator apparently lacks any pattern. However, if we use the algorithm to generate a long sequence of numbers, the same numbers will be repeated with a specific period.

Therefore, random numbers generated by the computer algorithms are not 'real' random numbers but are pseudo-random numbers. However, if we judiciously choose the algorithm and the seed values, the pseudo-random numbers are adequate for most of our purposes.

Uniformly distributed random numbers between zero and one are widely useful in Monte Carlo. However, certain applications will require us to generate random numbers from non-uniform distribution. We cannot use the default pseudo-random number generators directly for those probability distributions.

Suppose we want to generate random numbers following exponential distribution with rate λ. We need these random numbers for the simulation of a Poisson process. We will use the Inverse Transformation method to generate these random numbers.

x is a random variable following the exponential distribution $\text{Exp}(\lambda)$. Remember that the CDF of any random variable must be less-equal to 1. So, $F_X(x)$ must be a value in the interval [0, 1].

Let u be a random number from the uniform distribution $U(0, 1)$, generated by a pseudo-random number generator. Then $0 \leq u < 1$. We want to find the value of x for which $F_X(x)$

= *u*. As *u* is a random number, *x* will also be a random number but having exponential distribution.

As $x \sim Exp(\lambda)$,

$$F_X(x) = 1 - e^{-\lambda x} = u$$

$$\therefore \quad x = -\frac{1}{\lambda}\ln(1-u) \tag{2}$$

(1 – *u*) itself is a uniformly distributed random number in the interval zero to one. Therefore, Equation 2 can be simplified to $x = -\frac{1}{\lambda}\ln(u)$.

The inverse transformation method can be used for any distribution where the CDF's inverse function $F_X^{-1}(\cdot)$ can be calculated easily. For example, we can use this method to generate uniformly distributed random numbers in the interval *a* to *b*.

The CDF of uniform distribution *U*(*a*, *b*) is (*x* – *a*)/(*b* -*a*). Using the inverse transformation, we get

$$x = a + (b-a)u \tag{3}$$

here *u* is a random number from *U*(0, 1)

Unfortunately, the inverse transformation is not possible for all CDFs. The Acceptance-rejection method is useful for such probability distributions. This method is based upon the concept of random sampling in Monte Carlo. In the dart-throwing example, we have considered a line where the darts hit. Now consider a two-dimensional dartboard.

Suppose the blue line in Figure 7a is the PDF $f_X(x)$ of a probability distribution, for $a \le x \le b$. We want to generate random numbers from this distribution. We enclose this PDF with a rectangle (the pink rectangle *abdc* in Figure 7a). The width of this rectangle is (*b* -*a*), and the height is *M*. *M* is equal to the maximum value of $f_X(x)$.

This rectangle is our dartboard. We throw thousands of darts without any bias. Some of those hit the blue area under $f_X(x)$, and some hits the pink region. We accept only the darts that are in the blue region and remove the rest.

Consider two values of x – x_1 and x_2. As the $f_X(x = x_1) > f_X(x = x_2)$, the number of accepted darts on the vertical line at x_1 would be more than that on the vertical line for x_2 (Figure 7a). Darts are thrown without a bias. Therefore, the frequency of accepted darts on the line for x_1 will be equivalent to the density of x_1 defined by the PDF.

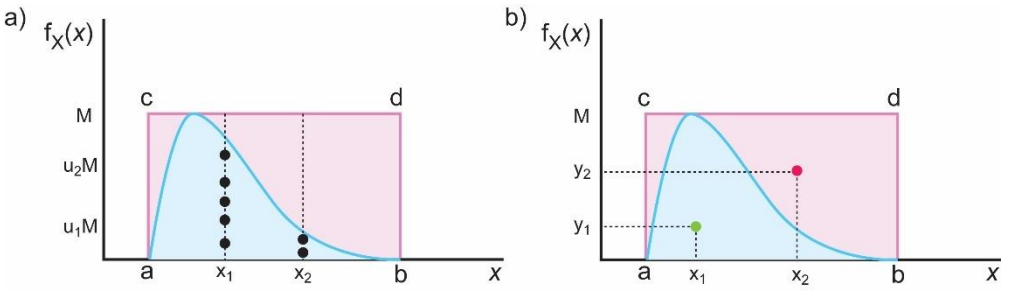


Figure 7: Acceptance-rejection method for random number generation. The blue line represents the target PDF $f_X(x)$, for $a \leq x \leq b$. (a) Results of multiple dart throw. The black circles represent darts. Darts in the pink regions are removed. For clarity, only the darts on the vertical lines for x_1 and x_2 are shown. (b) shows acceptance and rejection of a random number. The red circle is a rejected sample, and the green circle represents the accepted random number x_1.

That is what we want. Our random number generator should generate these two numbers with frequencies proportional to their densities defined by the PDF. If we generate a sequence of random numbers, x_1 should have a higher chance of appearing in that sequence than x_2.

Each accepted dart represents one random number that follows the given PDF. The acceptance-rejection algorithm for generating N number of random numbers following the PDF $f_X(x)$ is given below. As the dartboard is two-dimensional, we need two uniformly distributed random numbers – one for the horizontal axes and the other for vertical axes.

```
i = 0
while i < N
      Generate x' ~ U(a, b)
      Generate y ~ U(0, M)
      if y ≤ f_X(x = x')
            accept x' and report it as a random number
            i = i + 1
      end
end
```

In Figure 7b, x_1 is an accepted random number. However, the red circle for x_2 is rejected as $y_2 > f_X(x = x_2)$. Depending upon the shape of the PDF, this algorithm may reject a huge number of random numbers. That makes the algorithm slow and inefficient.

In this algorithm, we are sampling uniformly in both horizontal and vertical axes. The PDF in Figure 7 is very shallow for values of x near b. Therefore, most of the samples generated near b will get rejected. This wastage can be avoided if we sample lesser near b and sample more where the PDF is high.

That is achieved by using a proposal distribution. The proposal distribution will guide us in random sampling. The shape of $f_P(p)$, the PDF of the proposal distribution, should be similar to the target PDF $f_X(x)$. Also, the inverse function $F_P^{-1}(\cdot)$ for the CDF of the proposal distribution must be known.

This second requirement is essential as we will be generating random numbers using the proposal distribution. For a two-dimensional dart throw, we need two random numbers. In this modified algorithm, the random number for vertical axes will be generated from a uniform distribution; but the random number for the horizontal axes will be generated from the proposal distribution.

The third requirement is that there should be a constant c, such that $f_X(x) \leq cf_P(p = x)$, for all x. An optimum c is $\max\{f_X(x) / f_P(p = x)\}$. The algorithm for this method is given below.

```
i = 0
while i < N

      Generate u ~ U(0, 1)
      x' = F_P^-1(u)

      Generate y ~ U(0, cf_P(p = x'))

      if y ≤ f_X(x = x')
            accept x' and report it as a random number
            i = i + 1
      end
end
```

Exact stochastic simulation of chemical reactions

It is easy to simulate transcription of a gene or a zero-order reaction as Poisson processes. However, it gets complicated when multiple reactions occur in the system. One molecule can participate in multiple reactions, and reactions can be coupled. In 1977, Daniel T. Gillespi presented a numerical algorithm for exact stochastic simulation of a system of coupled chemical reactions. This algorithm is called Gillespi's algorithm. This algorithm and many variants of it are widely used in stochastic modeling in Systems Biology.

We start with the definition of the problem. Suppose we have well stired system with a fixed volume V and the system is at a thermal equilibrium. There are M reactions, $R_\mu \in \{R_1, R_2, \cdots, R_M\}$, happening in the system. These reactions involves N molecular species $S_i \in \{S_1, S_2, \cdots, S_N\}$.

Note that we are not defining reactants or products separately as the product of a reaction can react to another. At time t the number of S_i is X_i. We define the state of the system at t as $\mathbf{X}(t) = \left(X_1(t), X_2(t), \cdots, X_N(t)\right)$.

We have the stoichiometric information for each reaction. Based on the stoichiometry, we define a state change vector $\boldsymbol{\nu}_j = (x_{1j}, x_{2j}, \ldots, x_{Nj})$ for $j = 1$ to M. Here, x_{ij} is the change in number of S_i in the j^{th} reaction.

With all these information, we ask for a given $\mathbf{X}(t=0)$ what is $\mathbf{X}(t)$ at time t?

To predict the future state of the system, we need to answer two questions:

1) When will the next reaction happen?
2) Which of the M reaction will happen?

As the problem is stochastic, we will have probabilistic answers for both the question. Let us start with a few assumptions and definitions. We assume that in an infinitesimally small time interval dt, only one of the M reactions can happen, and successive reactions are mutually independent.

Each of the M reactions must have a probability to happen in dt. Let there is reaction where one molecule of S_1 reacts with one molecule of S_2. Let c be a constant such that the probability of one molecule of S_1 reacting with one molecule of S_2 in small time dt is cdt.

If we have X_1 number of S_1 and X_2 number of S_2, the reaction can happen in ${}^{X_1}C_1 \, {}^{X_2}C_1 = X_1 X_2$ ways. As each of these combinations are independent, the overall probability of having one reaction in dt is $X_1 X_2 c dt$.

c has the unit of 1-by-time. So it is a rate constant, and we call it the Mesoscopic Rate Constant. It is different from the usual rate constant for a reaction that we estimate in a laboratory experiment and use in dynamical models. For the time being, consider that we can calculate the mesoscopic rate constants for all the reactions in the system.

Let us now generalize the reaction scheme. Consider that p molecules of S_1 react with q molecules of S_2.

$$pS_1 + qS_2 \rightarrow S_3$$

Let c be a constant such that the probability of one set of p molecule of S_1 reacting with one set of q molecule of S_2 in small time dt is cdt.

If there are X_1 and X_2 numbers of S_1 and S_2, respectively, then the reaction can happen in ${}^{X_1}C_p \, {}^{X_2}C_q$ ways. So the probability of having one reaction in dt is,

$${}^{X_1}C_3 \, {}^{X_2}C_2 c dt = h c dt = a dt$$

Here $h = {}^{X_1}C_p \, {}^{X_2}C_q$ is a stoichiometric factor and $a = hc$ is called the reaction propensity. Mesoscopic rate constant c is a constant, but h depends upon the number of molecules of S_1 and S_2. Therefore, the reaction propensity changes with the number of molecules of S_1 and S_2.

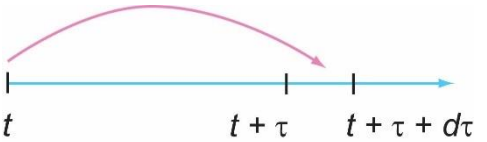
Generalizing this formulation, let c_μ and h_μ are the mesoscopic rate constant and stoichiometric factor for the μ-th reaction, respectively. Then, $a_\mu dt$ is the probability that the μ-th reaction R_μ will happen in the small interval dt. a_μ is the reaction propensity of R_μ and $a_\mu = h_\mu c_\mu$.

We need another definition. Given the state at time t, $\mathbf{X}(t)$,

$P(\mu, \tau) d\tau \equiv$ Probability, a reaction will take place in volume V, in the interval of $(t+\tau, t+\tau+d\tau)$ and the reaction will be R_μ.

Here $P(\mu,\tau)$ is a PDF. It is called the Reaction Probability Density Function.

Pay attention to the definition of this PDF. We are at time t, and we are asking when will the next reaction happen and which reaction would happen. As defined in the PDF, the next reaction will happen after a period of τ (see the following diagram). From t to t + τ, no reaction happens. So τ is called the waiting time and $0<\tau<\infty$.



$P(\mu,\tau)d\tau$ is a product of two probabilities,

$$P(\mu,\tau)d\tau = P_0(\tau)\left(a_\mu d\tau\right) \tag{4}$$

Here $P_0(\tau)$ is the probability that no reaction will happen in the interval $(t,t+\tau)$ and $a_\mu d\tau$ is the probability that the μ-th reaction R_μ will happen in the interval $(t+\tau,t+\tau+d\tau)$.

Let us calculate $P_0(\tau)$. There are M reactions, and $a_\mu dt$ is the probability that the μ-th reaction will happen in an infinitesimally small time interval *dt*. Therefore the probability that no reaction will happen in *dt* is $\left(1-\sum_{\mu=1}^{M} a_\mu dt\right)$.

To calculate $P_0(\tau)$, divide τ in *K* number of infinitesimally small intervals: $\varepsilon=\frac{\tau}{K}$, where $\varepsilon\to 0$ and $K\to\infty$. No reaction should happen in each of these small intervals. That would assure that no reaction happens in τ.

Therefore,

$$
\begin{aligned}
P_0(\tau) &= \lim_{K\to\infty}\prod_{i=1}^{K}\left(1-\sum_{\mu=1}^{M}a_\mu\varepsilon\right) \\
&= \lim_{K\to\infty}\left(1-\sum_{\mu=1}^{M}a_\mu\varepsilon\right)^K \\
&= \lim_{K\to\infty}\left(1-\sum_{\mu=1}^{M}a_\mu\frac{\tau}{K}\right)^K = e^{-\sum_{\mu=1}^{M}a_\mu\tau}
\end{aligned}
\tag{5}
$$

Now we can rewrite Equation 4 as,

$$
\begin{aligned}
P(\mu,\tau)\,d\tau &= e^{-\sum_{\mu=1}^{M}a_\mu\tau}\left(a_\mu\,d\tau\right) \\
&= a_\mu e^{-\sum_{\mu=1}^{M}a_\mu\tau}\,d\tau \\
&= \left(\frac{a_\mu}{\sum_{\mu=1}^{M}a_\mu}\right)\left(\sum_{\mu=1}^{M}a_\mu e^{-\sum_{\mu=1}^{M}a_\mu\tau}\,d\tau\right) \\
&= \underbrace{P(\mu\,|\,\tau)}_{\substack{\text{which reaction}\\ \text{happens}}}\quad \underbrace{P(\tau)}_{\substack{\text{when the}\\ \text{next reaction}\\ \text{happens}}}
\end{aligned}
\tag{6}
$$

On the right-hand side of Equation 6, $a_\mu / \sum_{\mu=1}^{M} a_\mu$ is the normalized reaction propensity of the μ-th reaction. $0 \le a_\mu / \sum_{\mu=1}^{M} a_\mu \le 1$. So, $a_\mu / \sum_{\mu=1}^{M} a_\mu$ is the probability that out of the M possible reactions, the μ-th reaction will happen.

You must have noticed that the second part of the right-hand side of Equation 6 has the PDF of an exponential distribution. That is the PDF of the waiting time τ and it follows the exponential distribution – $f(\tau) = \sum_{\mu=1}^{M} a_\mu e^{-\sum_{\mu=1}^{M}a_\mu\tau}$ with the rate parameter $\sum_{\mu=1}^{M} a_\mu$. Therefore, the mean waiting time is $E(\tau) = 1/\sum_{\mu=1}^{M} a_\mu$.

Equation 6 is at the heart of Gillespi's algorithm. We can implement it numerically. The algorithm is given below.

1. Initialize :
   a) The number of species N.
   b) The number of reactions M.
   c) Total time for simulation T.
   d) Mesoscopic rate constants: c_μ for μ = 1 to M.
   e) State change vector, $\mathbf{v}_j$, j = 1 to M.
   f) Stoichiometric information of all reactions.
   g) State of the system at t = 0: $\mathbf{X}(t) = (X_1, X_2, \ldots, X_N)$.
   h) Set t = 0.

2. If t < T, Calculate:
   a) Stoichiometric factor h_μ for μ = 1 to M, using $\mathbf{X}$(t) and stoichiometric information.
   b) Reaction propensity $a_\mu = h_\mu c_\mu$, for μ = 1 to M.
   c) $\sum_{\mu=1}^{M} a_\mu$
   d) $P(\mu) = \dfrac{a_\mu}{\sum_{\mu=1}^{M} a_\mu}$

3. Generate $u_1 \sim U(0,1)$.

   Calculate the time of next reaction, $\tau = \dfrac{1}{\sum_{\mu=1}^{M} a_\mu} \ln\left(\dfrac{1}{u_1}\right)$.

   Increase time by τ: t ← t + τ.

4. Generate $u_2 \sim U(0,1)$.

   If $\sum_{j=0}^{\mu-1} P(j) < u_2 \leq \sum_{j=0}^{\mu} P(j)$, where P(0)= 0, choose the μ-th reaction, $R_\mu \in (R_1, R_2, \cdots, R_M)$.

   Update the state of the system $\mathbf{X}$(t) as per R_μ: $\mathbf{X}(t) = \mathbf{X}(t) + \mathbf{v}_\mu$

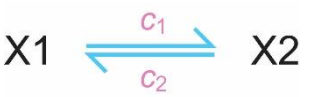
5. Go to step 2.
6. Report $\mathbf{X}$(t).

This algorithm uses two uniformly distributed random numbers. In step three, a random number is used to calculate the time of the next reaction. The waiting time τ follows an exponential distribution with rate parameter $\sum_{\mu=1}^{M} a_\mu$. However, we cannot directly generate an exponentially distributed random number. So, the method of Inverse Transformation is used.

Another random number is used in step four. At this step, we are choosing a reaction out of the M possible reactions. It is equivalent to throwing an M-faced dice. Therefore, the Monte Carlo method for simulation of a dice throw is used.

Once a reaction is randomly picked, the algorithm executes this reaction by updating the system $\mathbf{X}(t)$. Suppose in the chosen reaction, two molecules of S_4 react with three molecules of S_{10} to create four molecules of S_{25}. So the state change vector for this reaction will have $-2, -3, 4$ for S_4, S_{10}, S_{25}, respectively and rest of the entries will be zero. Following this vector, the state of the system is updated as, $X_4(t+\tau) = X_4(t) - 2$, $X_{10}(t+\tau) = X_{10}(t) - 3$, and $X_{25}(t+\tau) = X_{25}(t) + 4$. Counts of other molecules remain the same.

We used this algorithm to simulate the following reversible reaction:



The result of the simulation is shown in Figure 8a. We started with high X1. With time number of X1 drops, and X2 increases. However, these reactions are stochastic. Therefore, the data is not smooth and has abrupt changes in the number of X1 and X2.

We repeated the simulation, and the data for X1 from these simulations are shown in Figure 8b. Notice that the results of these simulations vary even though the parameters were the same. Such variations are expected for a stochastic algorithm and also for a real stochastic process. The pink line in Figure 8b shows the result of deterministic simulation of the same system using ordinary differential equation. You must have noticed that the deterministic behavior is close to the average of multiple stochastic simulations. Stochastic simulations are always repeated many times, and the average behavior is reported along with the variance.

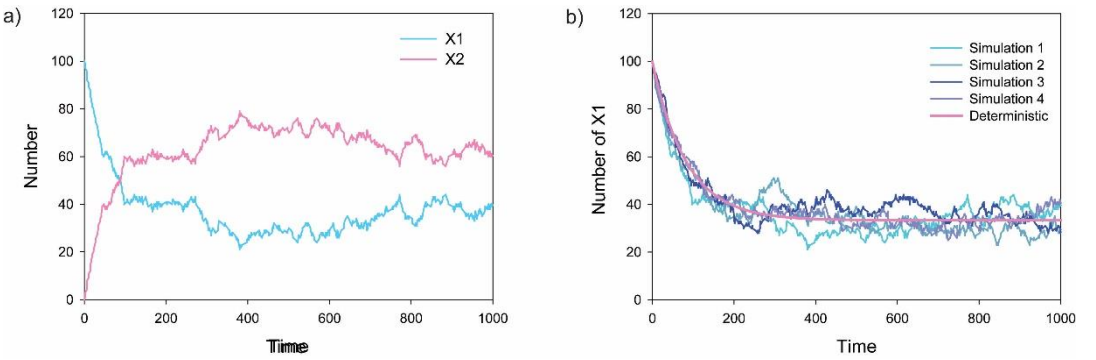


Figure 8: Stocahstic simulation of the reversible interconversion between X1 and X2, using Gillespie's algorithm. a) Shows the result of one simulation. b) Shows four independent stochastic simulations numbered 1 to 4. The pink line is for deterministic simulation of the same problem. In these simulations, c_1 = 0.008, c_2 = 0.004, $X1(t = 0)$ = 100 and $X2(t = 0)$ = 0.

How can we calculate mesoscopic rate constant from macroscopic rate constant?

We can do it by using dimensional analysis. Let us see 2 examples:

1. First order decay:

Here, dX/dt = -k.X, k being the rate constant. The unit of k is 1/s.

The stochastic rate = mesoscopic rate constant X number of molecules

Unit of stochastic rate is molecules per sec.

So, the unit of mesoscopic rate constant is 1/s.

As both the constant have same dimensions, we do not need any conversion.

2. Second order reaction (one product formed by the reaction of two reactants):

Here, unit of macroscopic rate constant is L/mol-s. But the dimension of mesoscopic rate constant is 1/s-number of molecules. So, to convert the macroscopic rate constant into mesoscopic one, you have to divide it by Avogadro number and reaction volume.

In general, we can convert the macroscopic rate constant into mesoscopic rate constant using the following relationship,

$$c = \frac{1}{(Na.V)^{m-1}}.k$$,

Here Na = Avogadro number, V = reaction volume, m = molecularity of the reaction. Molecularity of a reaction is obtained by summing the stochiometric coefficients of the reactants.

How to calculate h:

h represent the number of way the reactants can collide with each other. So it is nothing but all possible combinations of reactants.

For a simple reaction like the following,

$$A \rightarrow B$$

$h = n_A$, where n_A = number of A.

in other words,

$$h = \begin{pmatrix} n_A \\ 1 \end{pmatrix}$$

For,

$$A + 2B \rightarrow C$$

$$h = \binom{n_A}{1}\binom{n_B}{2}$$

So, for a reaction having m reactants,

$$h = \prod_{j=1}^{m} \binom{n_j}{s_j}$$

Here, s = stochiometric coefficients

Further notes:

The exact simulation method, discussed till now, is very time consuming as we have to calculate two random numbers and the time of computation is proportional to number of possible reactions. Various modified but fast methods have been developed for stochastic simulation of chemical reactions – e.g. "next reaction method", "τ-leap method".

Even then, stochastic simulation is problematic if we have a mixed system, i.e if some species are present in very large number and some others are in very low number. As a_μ depends upon number of molecules of a species, the computer will spend most of the time simulating reactions with very high number of molecules. Similarly as the number of molecules is very high for few species τ will also be very low. So the total time required for simulation is also high. This problem is sorted out in many recent algorithms, where these two types of reactions are simulated by separate algorithms. These are called hybrid algorithms.

Exercises

1. Pareto distribution is a type of power-law distribution widely used in economics and biology. The PDF of this distribution is

$$f_X(x) = \frac{bx_m^b}{x^{b+1}}$$

for $x_m \le x \le \infty$, $x_m > 0$, and $b > 0$

How will you generate random numbers that follow this distribution?

2. In a coin toss experiment the probability of getting a *head* is 0.6. Perform a simulation for 10 successive coin tosses and report the number of *heads* obtained. For simulation use the following uniformly distributed random numbers $U(0, 1)$ sequentially from left to right: 0.34, 0.01, 0.71, 0.84, 0.44, 0.04, 0.9, 0.5, 0.20, 0.96, 0.21, 0.3, 0.19, 0.96, 0.4.

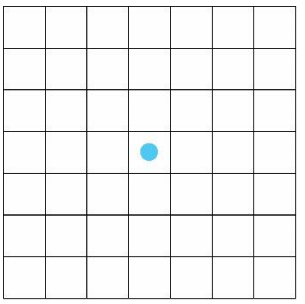
3. In a system, two independent reactions are happening. One reaction produces one molecule of A as a product, and the other produces one molecule of B as a product. Consider both the reactions as Poisson Processes with rate 1/3 per minute and 2/3 per minute, respectively. What is the probability that no molecules (neither A nor B) will be produced in the next two minutes?

4. Consider that protein expression is a Poisson process. The rate of expression of a protein is 2 copies per minute. What is the average time until the 100th copy is produced?

5. There are two copies of a gene G in a cell, under the control of two homologous promoters – P_1 and P_2. The transcription rate of G under the control of P_1 is λ_1 = 0.5 per minute. P_2 is a stronger promoter, and the rate of transcription of G under P_2 is λ_2 = 1.5 per minute. What is the probability that two copies of mRNA of G will be produced in the next two minutes?

6. Perform stochastic simulation of diffusion of a molecule (blue ball) in the square lattice shown below. Two sets of uniformly distributed random numbers $U(0, 1)$ are provided. Use random numbers from the first set to decide the direction of diffusion. Use the second set of random numbers to execute the diffusion. Use the

random numbers sequentially from left to right. Perform the simulation for three time-steps and calculate the displacement of the molecule in three time-steps. The dimension of each cell in the lattice is $1\times$ 1 unit^2. Consider the boundary of the lattice as periodic. The probability of diffusion is 0.6.



1st set of random numbers: 0.278058, 0.754627, 0.523845, 0.809792, 0.525448, 0.39481, 0.016055, 0.518212, 0.063255, 0.092969.

2nd set of random numbers: 0.729838, 0.937676, 0.14887, 0.660551, 0.378074, 0.748667, 0.235102, 0.506493, 0.450928, 0.516881.

7. Consider the following reaction as a Poisson process and simulate the reaction numerically. At t = 0, numbers of A, B, and C are 10, 20, and 0, respectively. The rate of reaction is 1 C per min.

$$A + 2B \rightarrow C$$

Show the data in a tabular format with one column for time and three other columns for numbers of A, B, and C. Do the simulation till 5 min. Report the numbers of A, B, and C at the fifth minute.

Use the following uniformly distributed random numbers $U(0, 1)$ sequentially from left to right: 0.099, 0.738, 0.155, 0.248, 0.392, 0.098, 0.276, 0.636, 0.353.