

LECTURE 19

Discrete Event Simulation

19.1. Stochastic simulation, the big picture	1
19.2. Simulating continuous random variables	2
19.3. Simulating Poisson processes	3
19.4. Simulating an M/M/1 queue	5
19.5. Simulating an M/M/k queue	9

19.1. Stochastic simulation, the big picture

In many probabilistic systems we can derive a mathematical model from basic principles. The key result of such a model is a probability distribution for the quantity of interest, from which expected values, variances, and other important quantities may be calculated. When analysis fails, we turn to simulation. For example, earlier we examined the robustness of an LC circuit by the following simulation procedure:

- (1) Generate random values for the two components L and C .
- (2) Calculate the resonant frequency $\omega_0 = \frac{1}{\sqrt{LC}}$ with these values.
- (3) Repeat (1) and (2) a large number of times, collecting an ensemble of ω_0 values.
- (4) Compute a histogram of ω_0 to estimate the probability density function $f(\omega_0)$. Calculate sample mean and sample variance of ω_0 to estimate the mean and variance of the resonant frequency.

All simulation procedures break down into these three parts: generating random input values of appropriate distribution, processing the inputs to obtain an ensemble of output values, and analyzing the outputs statistically to draw conclusions about the process. When the process can be expressed as a simple function of the input $Y = g(X)$, Monte Carlo simulations like the one above work well. Other kinds of systems, notably queues, have inputs which are streams of events rather than numbers. A different simulation approach is taken for these systems.

Before introducing discrete event simulation *per se*, we return to a topic that we didn't get to earlier in the course.

19.2. Simulating continuous random variables

Let X be a random variable with cumulative distribution function F_X . We will show that the random variable $U = F_X(X)$ is uniformly distributed on $[0, 1]$.

The cumulative distribution function of U is

$$F_U(u) = P(U \leq u) = P(F_X(X) \leq u) .$$

Now, assuming that F_X is monotonically increasing (a stronger requirement than nondecreasing) as well as continuous, the inverse function F_X^{-1} exists, and the event $(F_X(X) \leq u)$ is equivalent to the event $(X \leq F_X^{-1}(u))$. So,

$$F_U(u) = P(X \leq F_X^{-1}(u)) = F_X(F_X^{-1}(u)) = u, \quad 0 \leq u \leq 1 .$$

Now differentiate F_U to get the probability density function f_U .

$$f_U(u) = \frac{dF_U}{du} = 1, \quad 0 \leq u \leq 1 ,$$

which is uniform.

We can use this to generate continuous random variables from a desired PDF f_X . Use a random number generator to obtain a number U uniformly distributed on $(0, 1)$. Then, the random variable $X = F_X^{-1}(U)$ will be distributed according to f_X . This is called the *inverse transform method* for generating random numbers. It works when a mathematical expression exists for F_X^{-1} .

For example, to generate a value from an exponential distribution, begin with the CDF,

$$F(x) = 1 - e^{-x/\theta} .$$

The inverse function is obtained by solving $u = 1 - e^{-x/\theta}$ for x :

$$x = -\theta \log(1 - u) .$$

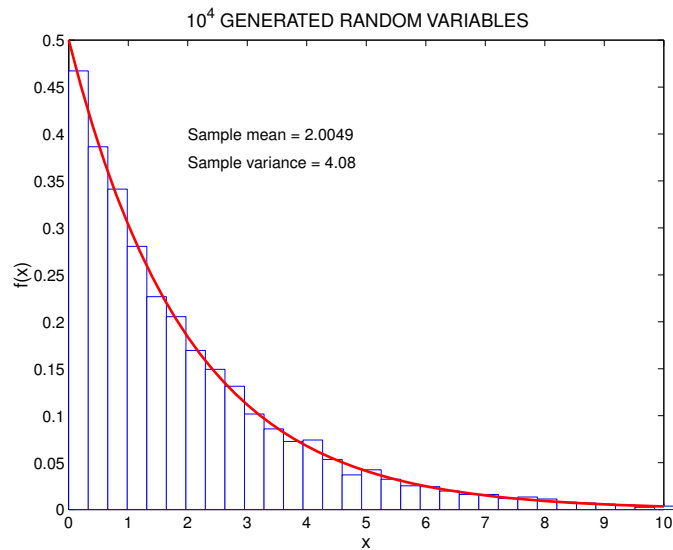
If U is uniformly distributed on $(0, 1)$, then so is $1 - U$, so we may also write

$$X = -\theta \log U \tag{19.1}$$

In Matlab, one line of code will generate a vector of N exponential random numbers:

```
x = -theta * log( rand(1,N) );
```

Here is a sample result: 10^4 samples from an exponential distribution with $\mu = 2$.



19.3. Simulating Poisson processes

When simulating a Poisson process, you may want to simulate how many counts (events) occur in a given interval of length t . Or, you may want to simulate the actual event stream, by generating a sequence of interarrival times.¹

Simulating the number of events in an interval

For a Poisson process with rate λ , we know that the interarrival times are distributed exponentially with mean $1/\lambda$. To simulate the number of events in an interval T , sum exponential random variables until the sum exceeds the length of the interval, T . A particularly efficient way of doing this follows from the earlier observation that an exponential random variable is proportional to the logarithm of a uniform random variable: $X = -\frac{1}{\lambda} \log U$. Let T_k be the k^{th} interarrival time. Then,

$$T \leq \sum_{k=1}^n T_k = -\frac{1}{\lambda} \sum_{k=1}^n \log U_k = -\frac{1}{\lambda} \log \left(\prod_{k=1}^n U_k \right)$$

or, equivalently,

$$e^{-\lambda T} \leq \prod_{k=1}^n U_k. \quad (19.2)$$

Summing exponential variates until the sum exceeds T is equivalent to multiplying uniform variates until the product exceeds $e^{-\lambda T}$. Then the desired Poisson variate is one less than this n . Here is Matlab code that does the calculation (as usual, $\mu = \lambda T$):

```
p = exp(-mu);
n = 1;
```

¹S. Ross, *Simulation*, third edition (Academic Press, 2001), Sections 4.2 and 5.4.

```

u = rand;
while(u >= p),
    u = u * rand;
    n = n+1;
end
X = n-1;

```

Simulating a sequence of events

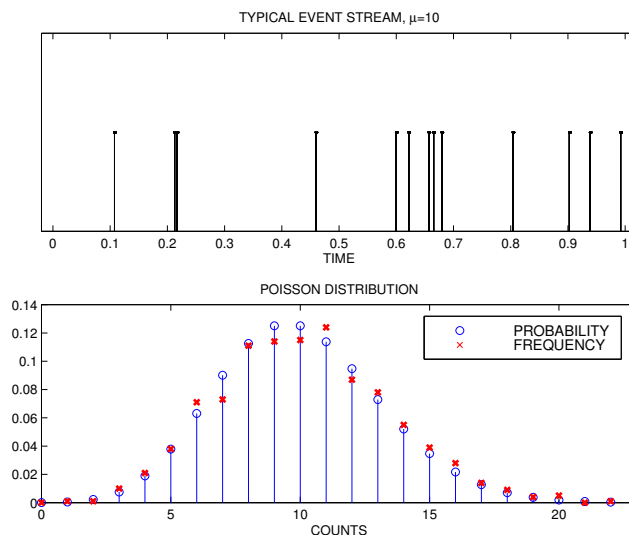
If instead we want to simulate a sequence of events, *i.e.*, arrival times, we can generate a sequence of exponential random variables, stopping when their cumulative sum exceeds the length of the interval, T . The following Matlab code does the job.

```

t1 = []; % Set of event times.
t = 0; % Running sum of interarrival times
while (t < T),
    t = t - log(rand)/mu; % Exponential r.v. with mean 1/mu
    t1 = [t1, t];
end
X = length(t1) - 1; % We also get the number of counts in T

```

Shown below is an example of an event stream with $\mu = 10$, generated by this method. The lower plot compares the histogram of event counts, for 1000 trials, with the Poisson probability function.



19.4. Simulating an M/M/1 queue

Consider an $M/M/1$ queue with arrival rate λ and service rate θ . To simulate the system we will keep track of the time, t and the number of customers in the system, $X(t)$. The simulation will run from $t = 0$ to $t = T$. These quantities will change when an event — arrival or departure — occurs.²

With $t = 0$ initially, we generate an arrival time (an exponential random variable with mean $1/\lambda$), update the time pointer t to this time and increment the number of customers. Then we generate a departure time, since there is a customer waiting to be served. We also generate the next arrival time. This gives us a pair $\{t_A, t_D\}$ called the *event list*. If $t_A < t_D$, it means that a new customer has arrived before the first customer has departed, and the number of customers must be incremented. On the other hand, if $t_D < t_A$, the first customer leaves the system before the second customer enters. Whichever occurs first, arrival or departure, a new arrival or departure time is generated, replacing the old one in the event list. If a departure leaves the queue empty, then a new departure time is not generated until a customer has arrived. In this way we proceed through time, event by event, until at last an arrival time is greater than the limit T . That customer is not added to the system, and the remaining customers empty out, one departure at a time.

A sample code is shown below. It can also be easily modified for finite-capacity queues, tracking how many customers are turned away, etc.

```

%% DISCRETE EVENT SIMULATION of M/M/1 QUEUE
%% E.W. Hansen, Engs 27 02W
%% Ref: S. Ross, "Simulation", Sect. 6.2

ra = 1;                % Arrival and service rates (per minute)
rs = 1.2;
T = 12*60;            % Simulation time (12 hours)

% Event times
A = [];               % Series of arrival times
D = [];               % Series of departure (service completion) times
S = [];               % Series of service times

% Initial conditions
n = 0;                % Current number in system
t = 0;                % Elapsed time
y = -log(rand)/ra;    % Exponential r.v., first interarrival time
ta = y;               % First arrival time
td = inf;             % Initialize departure time

```

²Ross, Sections 6.1-6.2; for a different approach, J.J. Higgins and S. Keller-McNulty, *Concepts in Probability and Stochastic Modeling* (Duxbury, 1995), pp. 318-320

```

while(min(ta, td) <= T),
% Case 1: Not time for departure yet, so generate an arrival
    if(ta <= td & ta <= T),
        t = ta; % Update elapsed time
        if(n<C)
            n = n+1; % If queue isn't full, add customer
            if (n==1), % If only one customer, generate service time
                y = -log(rand)/rs;
                td = t + y; % Next departure time
                S = [S, y]; % Update service time record
            end
            A = [A, t]; % Update arrival time record
        end
        y = -log(rand)/ra; % Next interarrival time
        ta = t + y; % Next arrival time
    end

% Case 2: Departure time has come
    if(td < ta),
        t = td; % Update elapsed time
        n = n-1; % Remove customer from queue
        D = [D, t]; % Update departure time record
        if (n==0), % If queue is empty
            td = inf; % set td to "infinity"
        else
            y = -log(rand)/rs; % Next service time
            td = t + y; % Next departure time
            S = [S, y]; % Update service time record
        end
    end

end

% Case 3: End of simulation, no new arrivals, flush queue
while(n > 0)
    t = td; % Update elapsed time
    n = n-1; % Remove customer from queue
    if(n > 0) % If queue isn't empty
        y = -log(rand)/rs; % Next service time
        td = t + y; % Next departure time
        S = [S, y]; % Update service time record
    end
    D = [D, t]; % Update departure time record
end
Tp = max(t-T, 0); % Time past T that last customer leaves queue

```

The great advantage of discrete event simulation is flexibility, and the power to examine systems that do not have nice analytical models. For example,

- Arrivals and departures can be made to follow any distribution, not just Poisson.
- Arrival and service rates can be time-varying.
- Servers can be connected in sequence (*e.g.*, wait for your food, wait to pay for it).

Post-simulation analysis

During the simulation, we keep track of each customer's arrival, departure, and service times. These are absolute times. Interarrival times are calculated by differencing the arrival vector,

```
tA = diff(A);
```

The time each customer spends in the queue is the difference of his departure and arrival time, and the time each customer spends waiting for service is the time in queue minus the service time,

```
tQ = D-A;
tW = tQ-S;
```

Means and standard deviations can be computed from `tA`, `tQ` and `tW` using the `mean` and `std` functions.

A running count of the queue size, $X(t)$, can be reconstructed from the arrival and departure times. An approximate way to do this is to divide the simulation time $[0, T_s]$ into intervals of equal length, Δt . Then, for each time $t_m = m\Delta t$, check to see how many arrivals and departures have occurred. The number of customers $X(t_m)$ is the number of arrivals before t_m minus the number of departures before t_m . Equivalently, suppose that a number of customers have arrived before time t_m . Some of these customers will also have departed by time t_m . The length of the queue is the number of customers which *havenot* departed by time t_m , those whose departure times are greater than t_m . Here is a way to do the calculation with MATLAB.

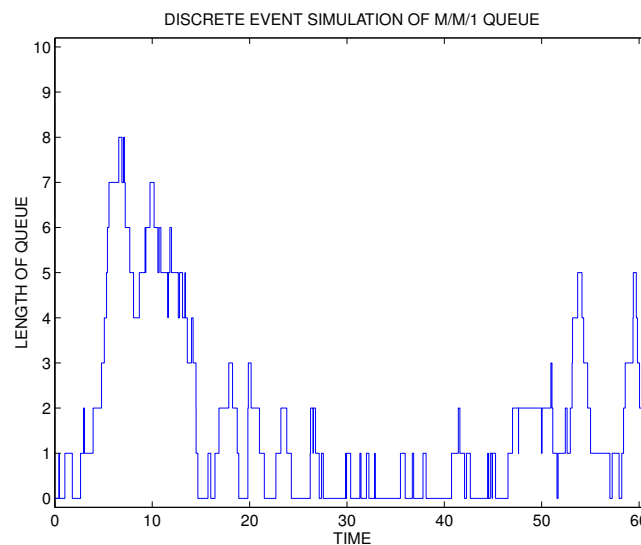
```
% Number of customers in system, vs t
% Divide the simulation time into intervals of equal length. Number of
% customers in system at time t = number of customers for which D > t > A.
% You have to rebin the customers in this way to get correct statistics.
dt = 0.1/rs; % 10% of the average service time
t = 0:dt:(T+Tp);
M = length(t);
X = zeros(1,M);
for m = 1:M,
    tm = t(m); % For the mth time,
    indx = find( A <= tm ); % Find the arrivals which have occurred
    X(m) = sum( D(indx) >= tm ); % Count the departures which haven't happened yet
end
```

Here is another way to calculate the running queue length which is more efficient, because it does not require subdividing the simulation interval $[0, T_s]$. Begin by merging the arrival and

departure times into a sorted vector and creating a vector `updown` which is `+1` for each arrival time and `-1` for each departure time. Then the queue length `N` is the cumulative sum of `updown`. The vector `N` can be plotted vs the merged event times.

```
tevent = [0, A, D];           % Merge
[tevent, indx] = sort(tevent); % and sort the event times
updown = [0, ones(size(A)), -1*ones(size(D))];
updown = updown(indx);      % Merge and sort 1 and -1 in the same way
N = cumsum(updown);         % Running sum
```

A sample graph is shown below.



To obtain a probability distribution for N , we need to know how frequently the queue is of a particular length. Let T_n be the amount of time the queue is of length n , then $P(N = n) \approx \frac{T_n}{T_s}$, where T_s is the duration of the simulation run. To calculate T_n , let t_m be an event time for which $N = n$. At this time, the queue length has just increased from $n - 1$ to n or decreased from $n + 1$ to n . At the next event time, t_{m+1} , the queue length will change. The difference $t_{m+1} - t_m$ is the amount of time the queue was length n for this one t_m . Summing over all the t_m gives the time T_n .

```
Ts = tevent(end);           % Ts = the last event time
Nq = max(N);
Pn = zeros(1, Nq+1);       % Make a vector to hold the probability
for n=1:Nq,                % For each queue length
    indx = find(N==n);     % Find all events for which N=n
    if(max(indx) == length(tevent)), % Don't count the last event
        indx = indx(1:end-1);
    end
    Tn = sum(tevent(indx+1)-tevent(indx)); % Sum the differences
```

```

    Pn(n+1) = Tn / Ts;           % Estimate the probability P(N=n)
end
Pn(1) = 1 - sum(Pn(2:end));     % P(N=0) is special

```

Now that we have the PMF for the queue length, we can calculate quantities like the mean queue length over the run, $\sum_n nP(N = n)$.

```

Navg = sum((0:Nq) .* Pn);

```

19.5. Simulating an M/M/k queue

The M/M/1 simulation can be extended to more than one server.³ The key change is in how we handle departures from the queue. If a customer arrives and a server is available, then she begins service immediately. Otherwise, she waits. When she reaches the head of the line, she waits until any one of the k customers being served departs, then begins service.

³Ross, Sections 6.3-6.4; for a different approach, Higgins and Keller-McNulty, pp. 320-322