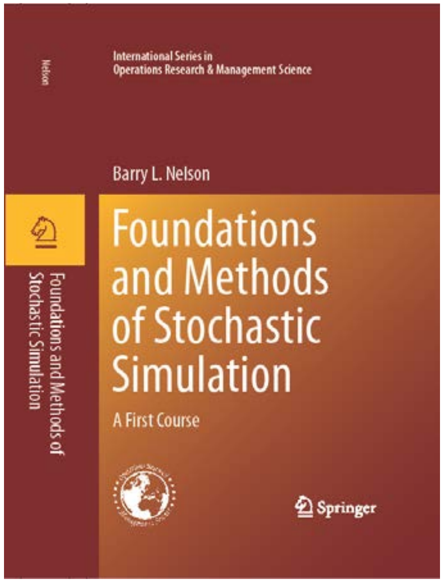


Chapters 1, 2 & 3: A Brief Introduction

©Barry L. Nelson
Northwestern University

July 2017

Why do we simulate?

- We typically choose to simulate a dynamic, stochastic system when the performance measure we want…
  - Is not analytically tractable.
  - Is not computationally tractable.
  - Cannot be approximated with a bound on the error.
- Let's start with a very simple example to remind us what we are doing and why.

Discrete-event simulation example

Two components work as an active and spare, so the system fails if both are failed.

The lifetime of a component is equally likely 1, 2, 3, 4, 5 or 6 days. Repair takes exactly 2.5 days (only one at a time). What can we say about the time to failure (TTF) for this system?

The *state* of the system is the number of functional components.
The *events* are the failure of a component and the completion of a repair.

	System	Event Calendar	
Clock	State	Next Failure	Next Repair
0	2		

	System	Event Calendar	
Clock	State	Next Failure	Next Repair
0	2	$0 + \mathbf{5} = 5$	∞
5	1	$5 + \mathbf{3} = 8$	$5 + 2.5 = 7.5$
7.5	2	8	∞
8	1	$8 + \mathbf{6} = 14$	$8 + 2.5 = 10.5$
10.5	2	14	∞
14	1	$14 + \mathbf{1} = 15$	$14 + 2.5 = 16.5$
15	0	∞	16.5

Features

- Simulated time (the simulation clock) jumps from event time to event time; this is called *next-event time advance*.

- The current state of the system, the event logic, and the list of future events are all we need to advance the system to the next state change.

- The simulation ends when a particular system state occurs. In other simulations termination may occur at a fixed time or event count.

- The system state over time is a *sample path* (output) from which we may extract performance measures. It is one realization of a stochastic process.

Outputs

Replications are statistically independent repetitions of the same model. We distinguish between *within-replication* and *across-replication* output data.

The time of system failure Y and the number of functional components $\{S(t); t \geq 0\}$ are *within-replication* outputs.

The times of system failure $Y_1, Y_2, \ldots, Y_n$, and the average number of functional components, $\bar{S}_1, \bar{S}_2, \ldots, \bar{S}_n$, from n replications are *across-replication* outputs.

Across-replication outputs should be independent (we made independent rolls of the dice) and identically distributed (we all followed the same rules).

Time averages

Notice that $\bar{S}$ is a *time-average* because $S(t)$ is a continuous-time output variable.

$$\bar{S} = \frac{1}{Y}\int_0^Y S(t)\,dt = \frac{1}{e_N - e_0}\sum_{i=1}^{N} S(e_{i-1}) \times (e_i - e_{i-1})$$

where $e_0, e_1, \ldots, e_N$ are the event times in a replication.

$$\begin{aligned}\bar{S} \;&=\; \frac{1}{15-0}\,[2(5-0) + 1(7.5-5) + 2(8-7.5) + 1(10.5-8)\\ &\quad +2(14-10.5) + 1(15-14)] = \frac{24}{15}\end{aligned}$$

Performance measures

We run simulations to *estimate* system performance, often to compare alternative designs for a system.

We justify this based on some version of the *strong law of large numbers*, either as the number of replications increases

$$\lim_{n \to \infty} \frac{1}{n} \sum_{i=1}^{n} Y_i = \mu \quad \text{(with probability 1)}$$

...or as the length of the replication goes to infinity (this would make sense if we did not stop the simulation when both components are failed)

$$\lim_{T \to \infty} \frac{1}{T} \int_0^T S(t)\, dt = \theta \quad \text{(with probability 1)}$$

This book

Building Simulations: Modeling and programming

Doing Simulations Well: Experimental design and output analysis

Simulation "Theory:" Why simulations work

Simulation for Research: Get you ready to use simulation in your research, even if your research area is not simulation

Programming: Book uses VBA for Excel, but Java and Matlab versions are available on the book web site.

Structure of a discrete-event simulation

- **Main program**
  - Initialize state variables, clock, statistics
  - Schedule at least one future event
  - Simulation loop: Until ending condition met
    - Call Timer to find next event
    - Call event subprogram
  - Final update of statistics and report results

- **Timer subprogram**
  - Remove next event from event calendar
  - Advance clock
  - Return event type

- **Event subprograms**
  - Update state variables
  - Schedule any future events
  - Update statistics

Programming quick start

```
Dim Clock As Double              ' simulation clock
Dim NextFailure As Double        ' time of next failure event
Dim NextRepair As Double         ' time of next repair event
Dim S As Double                  ' system state
Dim Slast As Double              ' previous value of the system state
Dim Tlast As Double              ' time of previous state change
Dim Area As Double               ' area under S(t) curve

Private Function Timer() As String
     Const Infinity = 1000000

' Determine the next event and advance time
    If NextFailure < NextRepair Then
        Timer = "Failure"
        Clock = NextFailure
        NextFailure = Infinity
    Else
        Timer = "Repair"
        Clock = NextRepair
        NextRepair = Infinity
    End If
End Function
```

Global declarations

```
Private Sub TTF()
' Program to generate a sample path for the TTF example
    Dim NextEvent As String
    Const Infinity = 1000000
    Rnd (-1)
    Randomize (1234)

' Initialize the state and statistical variables
    S = 2
    Slast = 2
    Clock = 0
    Tlast = 0
    Area = 0

' Schedule the initial failure event
    NextFailure = WorksheetFunction.Ceiling(6 * Rnd(), 1)
    NextRepair = Infinity

' Advance time and execute events until the system fails
    Do Until S = 0
        NextEvent = Timer
        Select Case NextEvent
        Case "Failure"
            Call Failure
        Case "Repair"
            Call Repair
        End Select
    Loop

' Display output
    MsgBox ("System failure at time " _
            & Clock & " with average # functional components " & Area / Clock)
    End
End Sub
```

```
Private Sub Failure()
' Failure event
' Update state and schedule future events
    S = S - 1
    If S = 1 Then
        NextFailure = Clock + WorksheetFunction.Ceiling(6 * Rnd(), 1)
        NextRepair = Clock + 2.5
    End If

' Update area under the S(t) curve
    Area = Area + Slast * (Clock - Tlast)
    Tlast = Clock
    Slast = S
End Sub

Private Sub Repair()
' Repair event
' Update state and schedule future events
    S = S + 1
    If S = 1 Then
        NextRepair = Clock + 2.5
        NextFailure = Clock + WorksheetFunction.Ceiling(6 * Rnd(), 1)
    End If

' Update area under the S(t) curve
    Area = Area + Slast * (Clock - Tlast)
    Tlast = Clock
    Slast = S
End Sub
```

Important Computer Simulation Concepts

$$U \xrightarrow{\text{algorithm}} X \xrightarrow{\text{logic}} Y$$

Source of randomness, usually assumed i.i.d. U(0,1)

Input random variables with known distributions that we specify; ex: component TTF is DU(1,2,3,4,5,6)

Output random variables whose properties we want to estimate; ex: system TTF

Random-number generation

Even if we could get random numbers, we don't want them. We prefer that our results be repeatable, but appear to be random: *pseudorandom*.

Think of this as a long list with period P that repeats if we reach the end.

$$U_1, U_2, \ldots, U_P, U_1, U_2, \ldots$$

U's are consumed in order from wherever we start.

With a good pseudorandom number generator it does not matter where in the list we start (usually specified by a "seed" or "stream").

Random-variate generation

We specify an input distribution F_X we want.

Variate generation means finding an algorithm such that if $X = \texttt{algorithm}(U)$ and $U \sim U(0,1)$, then

$$\Pr\{X \leq x\} = F_X(x)$$

An algorithm that always works is

$$X = F_X^{-1}(U) = \min\{x : F_X(x) \geq U\}$$

Proof for continuous case:
$\Pr\{X \leq x\} = \Pr\{F_X^{-1}(U) \leq x\} = \Pr\{U \leq F_X(x)\} = F_X(x)$

Replications

We will typically make multiple replications to get better estimators and compute a measure of error (e.g., confidence interval).

Results across replications are (pseudo) i.i.d.

- "independent" because different (pseudo)random numbers are used if we initialize the generator OUTSIDE the replication loop.
- "identically distributed" because initial conditions and logic are always the same INSIDE the loop (e.g, 2 functional components at the beginning).

Thus standard large-sample statistics apply:
$Y_1, Y_2, \ldots, Y_{100}$ system times to failure: use $\bar{Y} \pm 1.96S/\sqrt{100}$

Replication version of TTF

```
Private Sub TTFRep()
' Program to generate a sample path for the TTF example
    Dim NextEvent As String
    Const Infinity = 1000000
    Rnd (-1)
    Randomize (1234)
' Define and initialize replication variables
    Dim Rep As Integer
    Dim SumS As Double, SumY As Double
    SumS = 0
    SumY = 0
    For Rep = 1 To 100
' Initialize the state and statistical variables
        S = 2
        Slast = 2
        Clock = 0
        Tlast = 0
        Area = 0
' Schedule the initial failure event
        NextFailure = WorksheetFunction.Ceiling(6 * Rnd(), 1)
        NextRepair = Infinity
' Advance time and execute events until the system fails
        Do Until S = 0
            NextEvent = Timer
            Select Case NextEvent
                Case "Failure"
                    Call Failure
                Case "Repair"
                    Call Repair
            End Select
        Loop
' Accumulate replication statistics
        SumS = SumS + Area / Clock
        SumY = SumY + Clock
    Next Rep
' Display output
    MsgBox ("Average failure at time " _
            & SumY / 100 & " with average # functional components " & SumS / 100)
    End
End Sub
```

Initialize random number generator (annotation pointing to Rnd (-1) / Randomize (1234))

Canonical examples

- To illustrate things as we go along we will use 4 ½ small examples that are
  - Easy to simulate.
  - Don't actually need to be simulated (they are pretty tractable).
- This will allow us to see the issues that arise in general without getting buried in details.
- These examples are also useful for testing new ideas in simulation.

$M(t)/M/\infty$ queue

- A queue with "ample" (infinite) servers. Used to design systems that should have adequate capacity nearly always.
- Arrivals are described by a nonstationary Poisson arrival process with rate $\lambda(t)$.
- Service times are independent and identically distributed exponential with finite mean τ.
- Examples: parking lot for large mall; cell phone system.

More on *M(t)/M/∞*

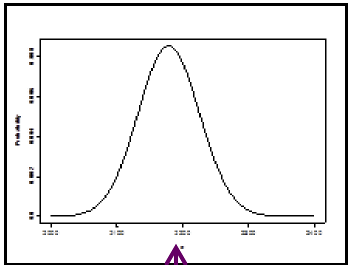
If $N(t)$ is the number of cars in the parking garage at time $t \geq 0$, then $N(t)$ has a Poisson distribution with mean $m(t)$, where $m(t)$ solves

$$\frac{d}{dt}m(t) = \lambda(t) - \frac{m(t)}{\tau}$$

with an appropriate initial condition like $m(0) = 0$ if the garage starts empty.

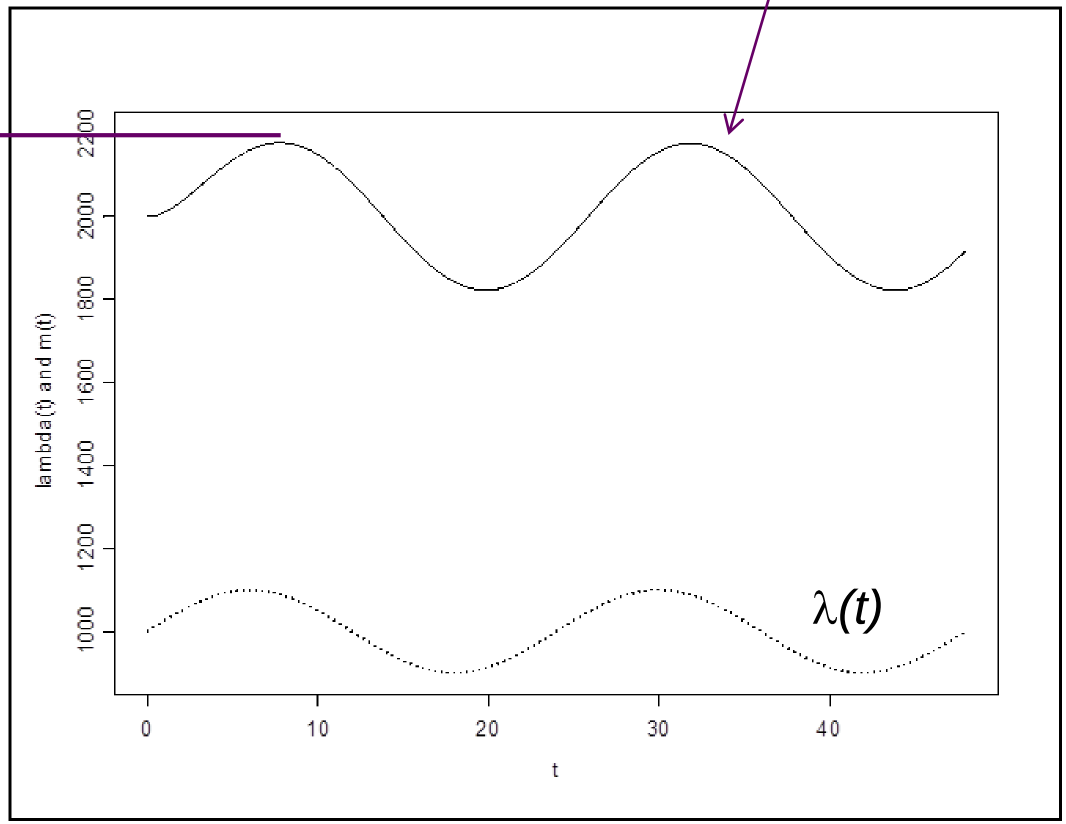
For capacity planning, solve for $m^\star = \max_t m(t)$ and choose c to control

$$\Pr\{\text{number of cars in the garage } \leq c\} = \sum_{n=0}^{c} \frac{\mathrm{e}^{-m^\star}(m^\star)^n}{n!}.$$



Distribution of number in the garage at mean *m**

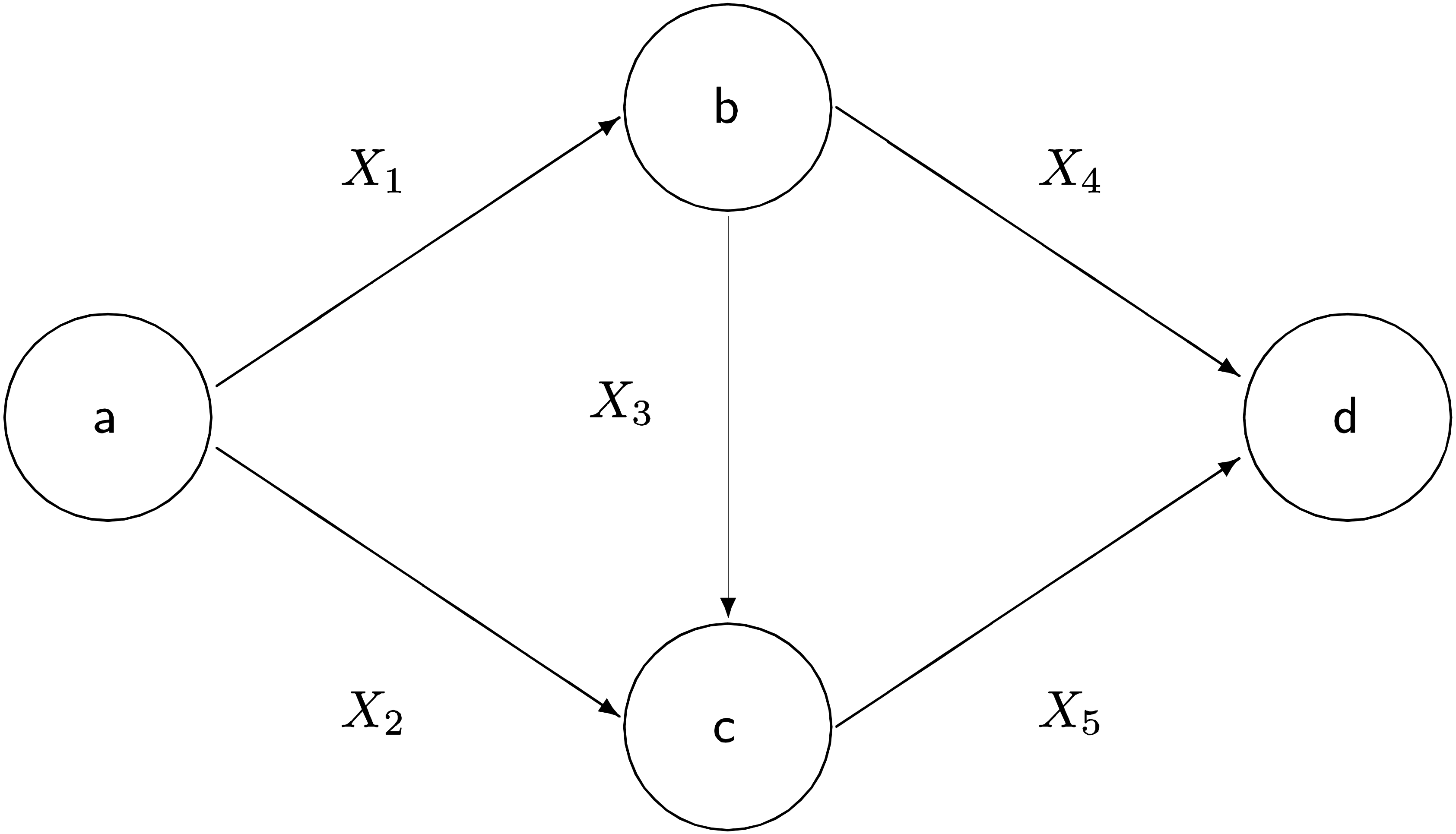
Mean number of cars in the garage *m(t)*



Simulation issues

- How do we simulate a nonstationary arrival process?
- How do we program a simulation that could (conceptually) have nearly an infinite number of "car departure" events?
- What relevant performance measure can we actually get out of a simulation?

Stochastic activity network (SAN)



More on the SAN

Let Y represent the time to complete the project, where X_i are the (random) activity times:

$$Y = \max\{X_1 + X_4, X_1 + X_3 + X_5, X_2 + X_5\}.$$

The project planners are interested in performance measures such as $\theta = \Pr\{Y > t_{\text{promised}}\}$ and $\mu = \mathrm{E}(Y)$.

If all of the activity times are exponential 1 then $\Pr\{Y \leq t_p\} =$

$$\left(\frac{1}{2}t_p^2 - 3t_p - 3\right)\mathrm{e}^{-2t_p} + \left(-\frac{1}{2}t_p^2 - 3t_p + 3\right)\mathrm{e}^{-t_p} + 1 - \mathrm{e}^{-3t_p}.$$

Simulation issues

The SAN illustrates

- Performance measures such as means, probabilities and quantiles (if we want to know how to set a promise date) are all relevant.

  Note that

  $$\theta = \mathrm{E}[I(Y > t_p)]$$

  so a probability is also a mean, but a quantile is different.

- A situation where the natural design is to make replications, so the question is how many to make.

M/G/1 queue

- A single-queue, first-come-first-served service system with one server.
- Arrivals are described by a stationary Poisson arrival process (interarrivals exponential $1/\lambda$)
- Service times are independent and identically distributed with finite mean τ and standard deviation σ.
- Examples: video kiosk, ticket window, ATM machine, receptionist.

M/G/1 & Lindley's equation

$A_1, A_2, \ldots$ i.i.d. exponential interarrival times with mean $1/\lambda$.

$X_1, X_2, \ldots$ i.i.d. service times with mean τ and standard deviation σ.

If $Y_1, Y_2, \ldots$ are the successive waiting times in queue then

$$Y_i = \max\{0, Y_{i-1} + X_{i-1} - A_i\}, \; i = 1, 2, \ldots$$

where $Y_0 = X_0 = 0$.

Notice that we expect the $\{Y_i\}$ to be *neither independent nor identically distributed.*

What is known?

Provided $\lambda < \infty$, $\rho = \lambda\tau < 1$ and $\sigma < \infty$ then

1. The outputs $Y_1, Y_2, \ldots$ converge in distribution to a random variable Y.
2. The sample mean $\bar{Y}(m) = m^{-1} \sum_{i=1}^{m} Y_i$ converges with probability 1 to a constant μ.
3. The $\mathrm{E}(Y)$ and μ are equal, and are given by

$$\mu = \mathrm{E}(Y) = \frac{\lambda(\sigma^2 + \tau^2)}{2(1 - \lambda\tau)}.$$

What if our goal is to estimate μ (or some other property of Y)?

Simulation issues

- The quantities of interest are defined in the limit; easy (sometimes) mathematically; hard (nearly always) in simulation.
- The experiment design is not so clear: One very long run or multiple replications?
- As $\rho \to 1$ both the mean and the variability of Y explode (e.g., the standard deviation of Y grows as $1/(1-\rho)$).

The AR(1) surrogate model

While a lot is known (at least in the limit) about the $M/G/1$, the $\max$ operator makes transient behavior difficult to derive.

A nice stand-in that shares similar behavior is the AR(1)

$$Y_i = \mu + \varphi(Y_{i-1} - \mu) + X_i, \; i = 1, 2, \ldots$$

with $X_1, X_2, \ldots$ i.i.d. $(0, \sigma^2)$ and $|\varphi| < 1$.

The AR(1) is easy to analyze because

$$Y_i = \mu + \varphi^i(Y_0 - \mu) + \sum_{j=0}^{i-1} \varphi^j X_{i-j}.$$

Asian option

A "call" option is a contract giving the holder the right to purchase a stock for a fixed "strike price" at some time in the future.

If the stock's value increases well above the strike price, then the option is a good deal provided the purchase price for the option was not too high.

If offered a call option with strike price K and maturity T on a stock that is currently trading at $X(0)$, how much should one be willing to pay for this option?

For an Asian option it is

$$\nu = \mathrm{E}\left[\mathrm{e}^{-rT}\left(\bar{X}(T) - K\right)^{+}\right]$$

Simulation issues

Usually $X(t)$ is modeled as GBM, a continuous-time, continuous-state stochastic process, and

$$\bar{X}(T) = \frac{1}{T}\int_0^T X(t)\,\mathrm{d}t.$$

We cannot directly simulate this.

The natural approximation

$$\widehat{\bar{X}(T)} = \frac{1}{m}\sum_{i=1}^{m} X(i\Delta t)$$

with $\Delta t = T/m$ introduces bias, but we need highly accurate and precise estimates in financial applications.