Simulation Modeling

Institute of Technology of Cambodia

Departerment of Applied Mathematics and Statistics.

Tepmony SIM & **Sokea LUEY**

January , 2023

Course outline

Introduction

- In many situations, analytic (symbolic) model is not available
- In many situations, the system for which the alternative procedures need to be tested may not even exist yet.
- The cost of conducting experiments may be prohibitive.
- In the case where the behavior cannot be explained analytically or data cannot be collected directly, the model might simulate the behavior indirectly in some manner and then test the various alternatives under consideration to estimate how each affects the behavior.
- Data can then be collected to determine which alternative is best.

Introduction

- We focus on **Monte Carlo Simulation**, a probabilistic model.
- Processes with an element of chance involved are called **probabilistic** or **stochastic**, as opposed to **deterministic**, process.
- The modeled behavior my be either deterministic or probabilistic.
- A principal advantage of Monte Carlo Simulation is the relative ease with which it can sometimes be used to approximate very complex probabilistic systems.

Introduction

- Monte Carlo simulation provides performance estimation over a wide range of conditions rather than a very restricted range as often required by an analytic model.
- On the negative side, simulation models are typically expensive to develop and operate. They may require many hours to construct and large amounts of computer time and memory to run (but not anymore a major problem)
- Another disadvantage is that the probabilistic nature of the simulation model limits the conclusions that can be drawn from a particular run unless a sensitivity analysis is conducted.

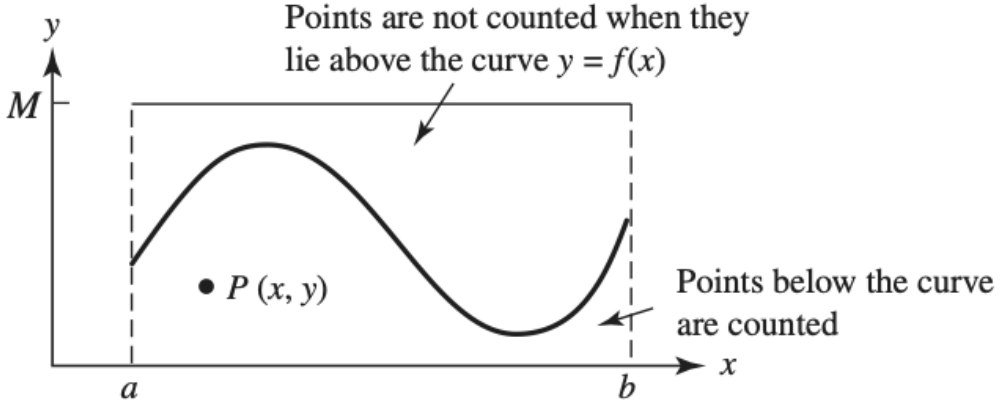
Simulating Deterministic Behavior: Area Under a Curve

Area Under a Curve Using Monte Carlo Simulation

- Let $f(x)$ be an "integrable" function defined on $a \leq x \leq b$
- Suppose for all $a \leq x \leq b$, $0 \leq f(x) \leq M$ for some $M > 0$
- Our goal is to calculate the area under the curve (and above x-axis) on $a \leq x \leq b$. That is, to calculate $\int_a^b f(x)\,dx$
- Let us draw a point $P(x,\ y)$ at random, where $a \leq x \leq b$ and $0 \leq y \leq M$.

Figure 5.2

The area under the nonnegative curve $y = f(x)$ over $a \leq x \leq b$ is contained within the rectangle of height M and base length $b - a$.



Simulating Deterministic Behavior: Area Under a Curve

Area Under a Curve Using Monte Carlo Simulation

- Let $A = \{(x,\ y)\ :\ 0 \leq y \leq f(x),\ a \leq x \leq b\}$
- We will **COUNT** as 1 if $P(x,\ y) \in A$, and as 0 if $P(x,\ y) \notin A$.
- Suppose we draw such point $P(x,\ y)$ at random for n times
- We then have the following approximation:

$$\frac{\text{Area of } A}{\text{Area of rectangle}} \approx \frac{\text{Number of points falling in } A}{n}$$

or

$$\text{Area of } A \approx M(b-a) \times \frac{\#\{P(x,\ y) \in A\}}{n}$$

Simulating Deterministic Behavior: Area Under a Curve

Monte Carlo Simulation: Algorithm for Computing Area

Monte Carlo Area Algorithm

Input	Total number n of random points to be generated in the simulation.
Output	AREA = approximate area under the specified curve $y = f(x)$ over the given interval $a \le x \le b$, where $0 \le f(x) < M$.
Step 1	Initialize: COUNTER = 0.
Step 2	For $i = 1, 2, \ldots, n$, do Steps 3–5.
Step 3	Calculate random coordinates x_i and y_i that satisfy $a \le x_i \le b$ and $0 \le y_i < M$.
Step 4	Calculate $f(x_i)$ for the random x_i coordinate.
Step 5	If $y_i \le f(x_i)$, then increment the COUNTER by 1. Otherwise, leave COUNTER as is.
Step 6	Calculate AREA $= M(b-a)$ COUNTER$/n$.
Step 7	OUTPUT (AREA)
	STOP

Remark.

- In Excel, the function **rand()** generates a uniform random number on $(0, 1)$. In **R**, we use **runif(1)**.
- If U = uniform on $(0, 1)$ and V = uniform on (a, b), then $V = a + (b - a) \times U$.

Simulating Deterministic Behavior: Area Under a Curve

Example

Using Monte Carlo Simulation, approximate the area beneath the curve $y = \cos x$ for $-\pi/2 \leq x \leq \pi/2$. In this example, we have $M = \max_{x \in \mathbb{R}} \cos x = 1$

Table 5.1 Monte Carlo approximation to the area under the curve $y = \cos x$ over the interval $-\pi/2 \leq x \leq \pi/2$

Number of points	Approximation to area	Number of points	Approximation to area
100	2.07345	2000	1.94465
200	2.13628	3000	1.97711
300	2.01064	4000	1.99962
400	2.12058	5000	2.01429
500	2.04832	6000	2.02319
600	2.09440	8000	2.00669
700	2.02857	10000	2.00873
800	1.99491	15000	2.00978
900	1.99666	20000	2.01093
1000	1.96664	30000	2.01186

Simulating Deterministic Behavior: Computing Integral

Computing $\int_a^b f(x)\,dx$ Using Monte Carlo Simulation

- Let $f(x)$ be an "integrable" function defined on $a \leq x \leq b$
- Suppose for all $a \leq x \leq b$, $m \leq f(x) \leq M$ for some values m and M
- Our goal is to calculate the integral of $f(x)$ over $a \leq x \leq b$.
  That is, to calculate $\int_a^b f(x)\,dx$
- Let us draw a point $P(x,\ y)$ at random, where $a \leq x \leq b$ and $m \leq y \leq M$.
- Let $A = \{(x,\ y)\ :\ 0 \leq y \leq f(x),\ a \leq x \leq b\}$ and $B = \{(x,\ y)\ :\ f(x) \leq y < 0,\ a \leq x \leq b\}$
- We will **COUNT** as 1 if $P(x,\ y) \in A$, as -1 if $P(x,\ y) \in B$, and as 0 otherwise.

Simulating Deterministic Behavior: Computing Integral

Computing $\int_a^b f(x)\,dx$ Using Monte Carlo Simulation

- Suppose we draw such point $P(x,\ y)$ at random for n times
- We then have the following approximation:

$$\int_a^b f(x)\,dx \approx$$

$$(M-m)(b-a) \times \frac{\#\{P(x,\ y) \in A\} - \#\{P(x,\ y) \in B\}}{n}$$

Simulating Deterministic Behavior: Computing Integral

Monte Carlo Computing Integral Algorithm

Input Total number n of random points to be generated in the simulation.

Output INTEGRAL = approximate value of $\int_a^b f(x)\,dx$

Step 1 Initialize: COUNTER $= 0$.

Step 2 For $i = 1, 2, \ldots, n$, do Steps 3–5.

Step 3 Calculate random coordinates x_i and y_i that satisfy $a \le x_i \le b$ and $m \le y_i \le M$.

Step 4 Calculate $f(x_i)$ for the random x_i coordinate.

Step 5 If $0 \le y_i \le f(x_i)$, then increment the COUNTER by 1. If $f(x_i) \le y_i < 0$, then increment the COUNTER by -1. Otherwise, leave COUNTER as is.

Step 6 Calculate INTEGRAL $= (M - m)(b - a)$COUNTER$/n$.

Step 7 OUTPUT (INTEGRAL)
STOP

Simulating Deterministic Behavior: Computing Volume

Computing Volume Using Monte Carlo Simulation

- Let the region $D \subset \mathbb{R}^k$.
- Let $B \subset \mathbb{R}^k$ be a box (or cube) such that $D \subset B$.
- Our goal is to calculate the volume of the region D, denoted by $V(D)$.
- Since B is a box, it is easy to generate a random point in B. Let us draw a point $x = (x^1, x^2, \ldots, x^k) \in B$ at random.
- We will **COUNT** as 1 if $x \in D$ and as 0 if $x \notin D$.
- Suppose we draw such point x at random for n times
- We then have the following approximation:

$$V(D) \approx V(B) \times \frac{\#\{x \in D\}}{n}$$

The box $B = \{(x^1, x^2, \ldots, x^k) \ : \ a^j \leq x^j \leq b^j, \ j = 1, 2, \ldots, k\}$ and $V(B) = \prod_{j=1}^{k} (b^j - a^j)$

Simulating Deterministic Behavior: Computing Volume

Computing Volume Using Monte Carlo Simulation

Input Total number n of random points to be generated in the simulation.

Output VOLUME = approximate value of the region D

Step 1 Initialize: COUNTER $= 0$.

Step 2 For $i = 1, 2, \ldots, n$, do Steps 3–4.

Step 3 Calculate random coordinates $x_i = (x_i^1, x_i^2, \ldots, x_i^k)$ that satisfy $x_i \in B$

Step 4 If $x \in D$, then increment the COUNTER by 1. Otherwise, leave COUNTER as is.

Step 5 Calculate VOLUME $= V(B) \times$ COUNTER$/n$.

Step 6 OUTPUT (VOLUME)
STOP

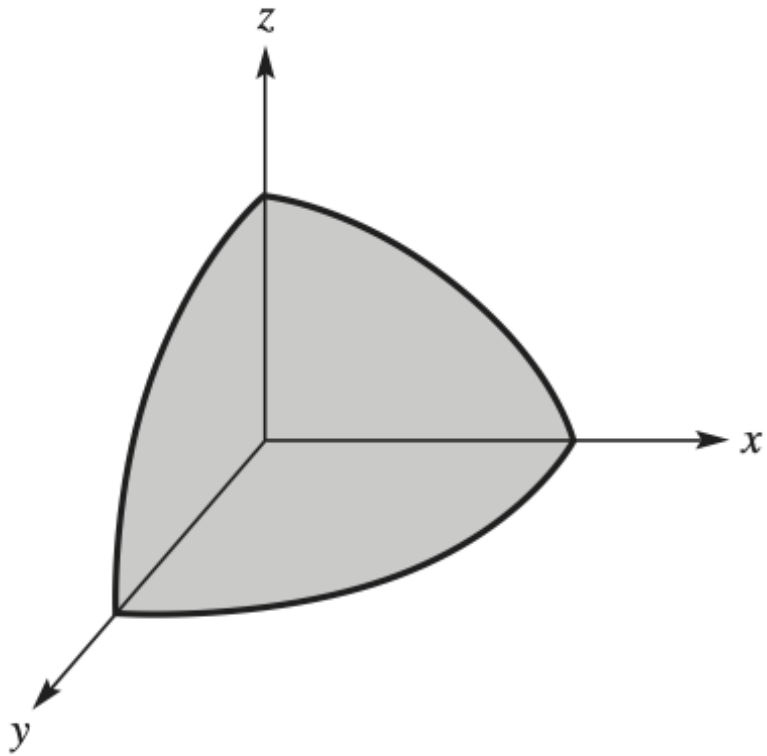
Simulating Deterministic Behavior: Calculating Volume

Example

Using Monte Carlo Simulation, approximate the volume of part of the sphere $x^2 + y^2 + z^2 \leq 1$ in the first octant (that is x, y, $z > 0$)

Figure 5.3

Volume of a sphere $x^2 + y^2 + z^2 \leq 1$ that lies in the first octant, $x > 0$, $y > 0$, $z > 0$



Simulating Deterministic Behavior: Calculating Volume

Example (Solution)

In this example, we choose $B = \{(x, y, z) \ : \ 0 < x \leq 1, 0 < y \leq 1, 0 < z \leq 1\}$, which is a cube. Hence, $V(B) = (1 - 0)(1 - 0)(1 - 0) = 1$.

n	$V(D)$	n	$V(D)$
100	0.5600	1500	0.5447
200	0.5700	2000	0.5375
300	0.5567	3000	0.5347
400	0.5675	4000	0.5310
500	0.5640	5000	0.5298
600	0.5533	6000	0.5293
700	0.5486	7000	0.5316
800	0.5500	8000	0.5263
900	0.5411	9000	0.5289
1000	0.5380	10000	0.5268

Simulating Deterministic Behavior: Computing Multiple Integral

Computing Integral $\int_D f(x)\,dx$ Using Monte Carlo Simulation

- Let $f(x)$ be an "integrable" function defined on a bounded region $x = (x^1, x^2, \ldots, x^k) \in D$. We assume the dimension of the region D is k.
- Suppose for all $x \in D$, $m \leq f(x) \leq M$ for some values m, M
- Our goal is to calculate the multiple integral of $f(x)$ over $x \in D$. That is, to calculate $\int_D f(x)\,dx$
- Let us draw a point $P(x,\ y)$ at random, where $x \in D$ and $m \leq y \leq M$.
- Let $A = \{(x,\ y)\ :\ 0 \leq y \leq f(x),\ x \in D\}$ and $B = \{(x,\ y)\ :\ f(x) \leq y < 0,\ x \in D\}$

Simulating Deterministic Behavior: Computing Multiple Integral

Computing $\int_D f(x)\,dx$ Using Monte Carlo Simulation

- We will **COUNT** as 1 if $P(x,\ y) \in A$, as -1 if $P(x,\ y) \in B$, and as 0 otherwise.
- Suppose we draw such point $P(x,\ y)$ at random for n times
- We then have the following approximation:

$$\int_D f(x)\,dx \approx$$

$$(M-m)V(D) \times \frac{\#\{P(x,\ y) \in A\} - \#\{P(x,\ y) \in B\}}{n},$$

where $V(D)$ is the volume or "content" of the region D.

Simulating Deterministic Behavior: Computing Multiple Integral

Monte Carlo Computing Integral Algorithm

Input Total number n of random points to be generated in the simulation.

Output INTEGRAL = approximate value of $\int_D f(x)\,dx$

Step 1 Initialize: COUNTER $= 0$.

Step 2 For $i = 1, 2, \ldots, n$, do Steps 3–5.

Step 3 Calculate random coordinates x_i and y_i that satisfy $x_i \in D$ and $m \leq y_i \leq M$.

Step 4 Calculate $f(x_i)$ for the random x_i coordinate.

Step 5 If $0 \leq y_i \leq f(x_i)$, then increment the COUNTER by 1. If $f(x_i) \leq y_i < 0$, then increment the COUNTER by -1. Otherwise, leave COUNTER as is.

Step 6 Calculate INTEGRAL $= (M - m)V(D)$COUNTER$/n$.

Step 7 OUTPUT (INTEGRAL)

STOP

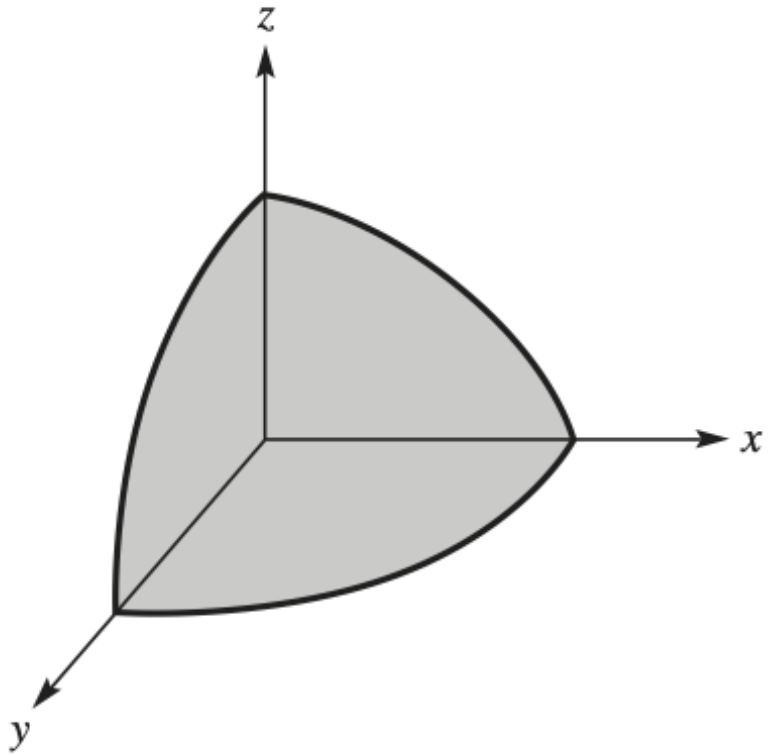
Simulating Deterministic Behavior: Computing Multiple Integral

Example. Volume Under Surface

Using Monte Carlo Simulation, approximate the volume of part of the sphere $x^2 + y^2 + z^2 \leq 1$ in the first octant (that is x, y, $z > 0$)

Figure 5.3

Volume of a sphere $x^2 + y^2 + z^2 \leq 1$ that lies in the first octant, $x > 0$, $y > 0$, $z > 0$



Simulating Deterministic Behavior: Computing Multiple Integral

Example. Volume Under Surface (Solution)

Input	Total number n of random points to be generated in the simulation.
Output	VOLUME = approximate volume enclosed by the specified function, $z = f(x, y)$ in the first octant, $x > 0$, $y > 0$, $z > 0$.
Step 1	Initialize: COUNTER = 0.
Step 2	For $i = 1, 2, \ldots, n$, do Steps 3–5.
Step 3	Calculate random coordinates x_i, y_i, z_i that satisfy $0 \leq x_i \leq 1, 0 \leq y_i \leq 1, 0 \leq z_i \leq 1$. (In general, $a \leq x_i \leq b, c \leq y_i \leq d, 0 \leq z_i \leq M$.)
Step 4	Calculate $f(x_i, y_i)$ for the random coordinate (x_i, y_i).
Step 5	If random $z_i \leq f(x_i, y_i)$, then increment the COUNTER by 1. Otherwise, leave COUNTER as is.
Step 6	Calculate VOLUME = $M(d - c)(b - a)$COUNTER$/n$.
Step 7	OUTPUT (VOLUME)
	STOP

Simulating Deterministic Behavior: Computing Multiple Integral

Example. Volume Under Surface (Solution)

Table 5.2 Monte Carlo approximation to the volume in the first octant under the surface $x^2 + y^2 + z^2 \leq 1$

Number of points	Approximate volume
100	0.4700
200	0.5950
300	0.5030
500	0.5140
1,000	0.5180
2,000	0.5120
5,000	0.5180
10,000	0.5234
20,000	0.5242

Generating Random Numbers

- Monte Carlo simulation algorithm requires random numbers.
- Random numbers have a variety of applications:
  - Gambling problems
  - Finding an area or volume
  - Modeling larger complex systems such as large-scale combat operations or air traffic control situations.
- A computer does not really generate random numbers, because computers employ deterministic algorithms. However, we can generate sequences of pseudorandom numbers that, for all practical purposes, may be considered random.
- There is no single best random number generator or best test to ensure randomness.
- Many programming languages, such as R, Pascal and Basic, and other software (e.g., Minitab, MATLAB, and EXCEL) have built-in random number generators for user convenience.

Generating Random Numbers

Middle-Square Method

- The middle-square method was developed in 1946 by John Von Neuman, S. Ulm, and N. Metropolis at Los Alamos Laboratories to simulate neutron collisions as part of the Manhattan Project. Their middle-square method works as follows:
  1. Start with a four-digit number x_0, called the **seed**.
  2. Square it to obtain an eight-digit number (add a leading zero if necessary).
  3. Take the middle four digits as the next random number.
- Continuing in this manner, we obtain a sequence that appears to be random over the integers from 0 to 9999. These integers can then be scaled to any interval a to b. For example, if we wanted numbers from 0 to 1, we would divide the four-digit numbers by $10,000$.

Generating Random Numbers

Middle-Square Method Cont.

- Let's illustrate the middle-square method. Pick a seed, say $x_0 = 2041$, and square it (adding a leading zero) to get 04165681. The middle four digits give the next random number, 1656. Generating 13 random numbers in this way yields

n	0	1	2	3	4	5	6
x_n	2041	1656	7423	1009	0180	0324	1049

n	7	8	9	10	11	12
x_n	1004	80	64	40	16	2

- We can use more than 4 digits if we wish, but we always take the middle number of digits equal to the number of digits in the seed. For example, if $x_0 = 653217$ (6 digits), its square $426,692,449,089$ has 12 digits. Thus, take the middle 6 digits as the random number, namely, 692449.
- The middle-square method is reasonable, but it has a major drawback in its tendency to degenerate to zero (where it will stay forever). Example with the seed 2041.

Generating Random Number

Linear Congruence

- Linear Congruence Method is a class of Pseudo Random Number Generator (PRNG) algorithms used for generating sequences of random-like numbers in a specific range:

$$x_{n+1} = (ax_n + b) \mod (c)$$

  - where x is the sequence of pseudo-random value.
  - a, b, c : are positive integer.
  - c: modulus, a: multiplier, b: increment
  - x_0 : the initial seed (given number)
  - The qualifier mod(c) in the equation means to obtain the remainder after dividing the quantity $(ax_n + b)$ by c.

- For example: with $a = 1, b = 7$, and $c = 10$, we obtain:

$$x_{n+1} = (1 \times x_n + 7) \mod (10)$$

mean x_{n+1} is the integer remainder upon dividing $x_n + 7$ by 10. Thus, if $x_n = 115$, then x_{n+1} remainder $\frac{122}{10} = 2$.

Generating Random Number Cont.

Linear Congruence Cont.

- Before investigating the linear congruence methodology, we need to discuss **cycling**, which is a major problem that occurs with random numbers.
- Cycling means the sequence repeats itself, and, although undesirable, it is unavoidable.
- At some point, all pseudo-random number generators begin to cycle.
- Example: at $x_0 = 7$, we find $x_1 = (1 \times 7 + 7)$ mod(10) or 14 mod(10), which is 4.
- Repeating this same procedure, we obtain the sequence $7, 4, 1, 8, 5, 2, 9, 6, 3, 0, 7, 4, ...$
- Note that there is cycling after 10 numbers. The methodology produces a sequence of integers between 0 and $c - 1$ inclusively before cycling.
- Nevertheless, c can be chosen to be very large, and a and b can be chosen in such a way as to obtain a full set of c numbers before cycling begins to occur. Many computers use $c = 2^{31}$ for the large value of c.

Simulating Probabilistic Behavior

The term **probability** refers to the study of both randomness and uncertainty, as well as to quantifying of the likelihoods associated with various outcomes. Probability can be seen as a long-term average.

The probability of an event can be thought of as the ratio:

$$\frac{\text{number of favorable event}}{\text{total number of events}}$$

We examine three simple probabilistic models:

1. Flip of a fair coin
2. Roll of a fair die or pair of dice
3. Roll of an unfair die or pair of unfair dice.

Simulating Probabilistic Behavior Cont.

A Fair Coin

What happens if we actually start flipping a coin? Will one out of every two flips be a head?

Let's define $f(x)$ as follows, where x is a random number between $[0, 1]$:

$$f(x) = \begin{cases} \text{Head}, & 0 \leq x \leq 0.5 \\ \text{Tail}, & 0.5 \leq x \leq 1 \end{cases}$$

In the long run, we expect to find the following percent occurrences:

Random number interval	Cumulative occurrences	Percent occurrence
$x < 0$	0	0.00
$0 < x < 0.5$	0.5	0.50
$0.5 < x < 1.0$	1	0.50

Simulating Probabilistic Behavior Cont.

A Fair Coin Cont.

Monte Carlo Fair Coin Algorithm

Input Total number n of random flips of a fair coin to be generated in the simulation.

Output Probability of getting a head when we flip a fair coin.

Step 1 Initialize: COUNTER $= 0$.

Step 2 For $i = 1, 2, \ldots, n$, do Steps 3 and 4.

Step 3 Obtain a random number x_i between 0 and 1.

Step 4 If $0 \leq x_i \leq 0.5$, then COUNTER $=$ COUNTER $+ 1$. Otherwise, leave COUNTER as is.

Step 5 Calculate $P(\text{head}) = \text{COUNTER}/n$.

Step 6 OUTPUT Probability of heads, $P(\text{head})$.

STOP

Table 5.3 Results from flipping a fair coin

Number of flips	Number of heads	Percent heads
100	49	0.49
200	102	0.51
500	252	0.504
1,000	492	0.492
5,000	2469	0.4930
10,000	4993	0.4993

Simulating Probabilistic Behavior Cont.

Roll of a Fair Die

Rolling a fair die adds a new twist to the process. We must devise a method to assign six events because a die consists of the numbers $\{1, 2, 3, 4, 5, 6\}$. The s probability of a particular number occurring is defined to be

$$\frac{\text{number of occurrence of the particular number } \{1,\ 2,\ 3,\ 4,\ 5,6\}}{\text{total number of trials}}$$

We can use the following algorithm to generate our experiment for a roll of a die.

Monte Carlo Roll of a Fair Die Algorithm

Input	Total number n of random rolls of a die in the simulation.
Output	The percentage or probability for rolls $\{1, 2, 3, 4, 5, 6\}$.
Step 1	Initialize COUNTER 1 through COUNTER 6 to zero.
Step 2	For $i = 1, 2, \ldots, n$, do Steps 3 and 4.
Step 3	Obtain a random number satisfying $0 \leq x_i \leq 1$.
Step 4	If x_i belongs to these intervals, then increment the appropriate COUNTER.

Simulating Probabilistic Behavior Cont.

Roll of a Fair Die Cont.

Step 4 If x_i belongs to these intervals, then increment the appropriate COUNTER.

$$
\begin{array}{lll}
0 \leq x_i \leq \frac{1}{6} & \text{COUNTER 1} = \text{COUNTER 1} + 1 \\
\frac{1}{6} < x_i \leq \frac{2}{6} & \text{COUNTER 2} = \text{COUNTER 2} + 1 \\
\frac{2}{6} < x_i \leq \frac{3}{6} & \text{COUNTER 3} = \text{COUNTER 3} + 1 \\
\frac{3}{6} < x_i \leq \frac{4}{6} & \text{COUNTER 4} = \text{COUNTER 4} + 1 \\
\frac{4}{6} < x_i \leq \frac{5}{6} & \text{COUNTER 5} = \text{COUNTER 5} + 1 \\
\frac{5}{6} < x_i \leq 1 & \text{COUNTER 6} = \text{COUNTER 6} + 1
\end{array}
$$

Step 5 Calculate probability of each roll $j = \{1, 2, 3, 4, 5, 6\}$ by COUNTER$(j)/n$.

Step 6 OUTPUT probabilities.

STOP

Table 5.4 Results from a roll of a fair die (n = number of trials)

Die value	10	100	1000	10,000	100,000	Expected results
1	0.300	0.190	0.152	0.1703	0.1652	0.1667
2	0.00	0.150	0.152	0.1652	0.1657	0.1667
3	0.100	0.090	0.157	0.1639	0.1685	0.1667
4	0.00	0.160	0.180	0.1653	0.1685	0.1667
5	0.400	0.150	0.174	0.1738	0.1676	0.1667
6	0.200	0.160	0.185	0.1615	0.1652	0.1667

Simulating Probabilistic Behavior Cont.

Roll of an Unfair Die

Assume the die is loaded or biased according to the following empirical distribution:

Roll value	P(roll)
1	0.1
2	0.1
3	0.2
4	0.3
5	0.2
6	0.1

The cumulative occurrences for the function to be used in our algorithm would be:

Value of x_i	Assignment
$[0, 0.1]$	ONE
$(0.1, 0.2]$	TWO
$(0.2, 0.4]$	THREE
$(0.4, 0.7]$	FOUR
$(0.7, 0.9]$	FIVE
$(0.9, 1.0]$	SIX

Simulating Probabilistic Behavior Cont.

Roll of an Unfair Die Cont.

We model the roll of an unfair die using the following algorithm:

Monte Carlo Roll of an Unfair Die Algorithm

Input Total number n of random rolls of a die in the simulation.

Output The percentage or probability for rolls $\{1, 2, 3, 4, 5, 6\}$.

Step 1 Initialize COUNTER 1 through COUNTER 6 to zero.

Step 2 For $i = 1, 2, \ldots, n$, do Steps 3 and 4.

Step 3 Obtain a random number satisfying $0 \leq x_i \leq 1$.

Step 4 If x_i belongs to these intervals, then increment the appropriate COUNTER.

$$
\begin{array}{rcl}
0 \leq x_i \leq 0.1 & \text{COUNTER 1} = \text{COUNTER 1} + 1 \\
0.1 < x_i \leq 0.2 & \text{COUNTER 2} = \text{COUNTER 2} + 1 \\
0.2 < x_i \leq 0.4 & \text{COUNTER 3} = \text{COUNTER 3} + 1 \\
0.4 < x_i \leq 0.7 & \text{COUNTER 4} = \text{COUNTER 4} + 1 \\
0.7 < x_i \leq 0.9 & \text{COUNTER 5} = \text{COUNTER 5} + 1 \\
0.9 < x_i \leq 1.0 & \text{COUNTER 6} = \text{COUNTER 6} + 1
\end{array}
$$

Step 5 Calculate probability of each roll $j = \{1, 2, 3, 4, 5, 6\}$ by COUNTER$(j)/n$.

Step 6 OUTPUT probabilities.

STOP

Simulating Probabilistic Behavior Cont.

Roll of an Unfair Die Cont.

Table 5.5 Results from a roll of an unfair die

Die value	100	1000	5000	10,000	40,000	Expected results
1	0.080	0.078	0.094	0.0948	0.0948	0.1
2	0.110	0.099	0.099	0.0992	0.0992	0.1
3	0.230	0.199	0.192	0.1962	0.1962	0.2
4	0.360	0.320	0.308	0.3082	0.3081	0.3
5	0.110	0.184	0.201	0.2012	0.2011	0.2
6	0.110	0.120	0.104	0.1044	0.1045	0.1

Inventory Model: Gasoline and Consumer Demand

The relative importance of the various factors determining the average daily cost, we develop the following model:

Average daily cost = f(storage costs, delivery costs, demand rate)

A continuous submodel:

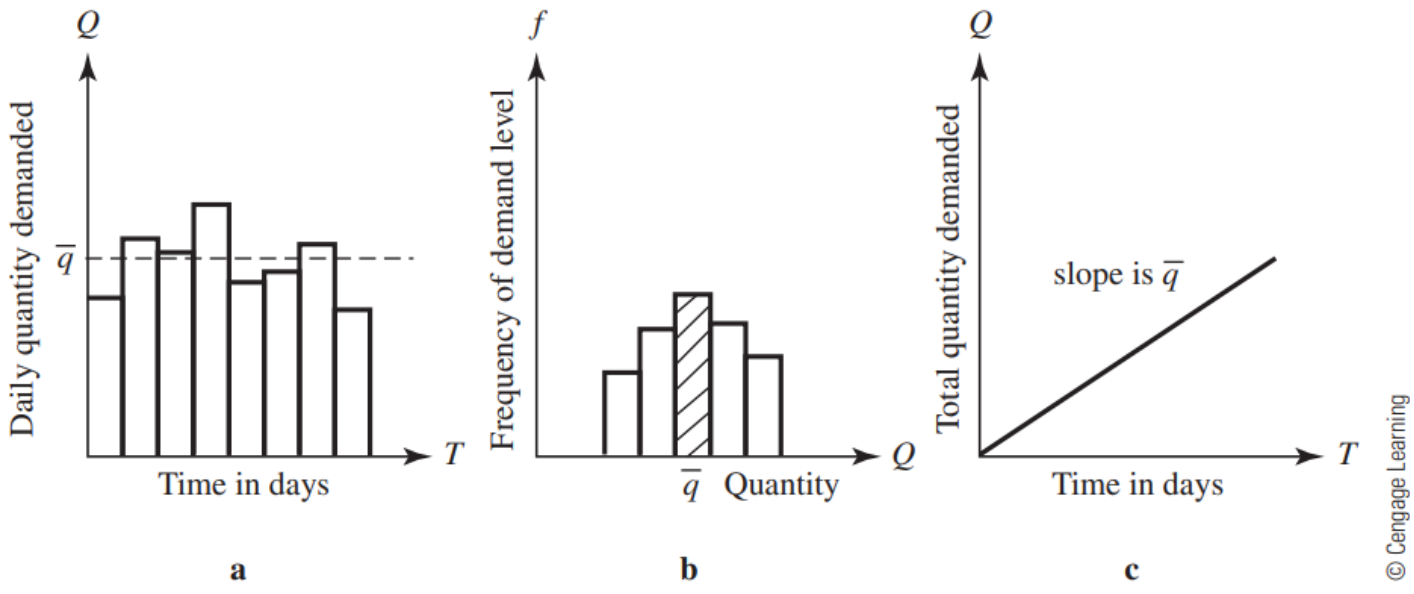


Figure 5.4

A constant demand rate

Inventory Model: Gasoline and Consumer Demand Cont.

From these assumptions we will construct, in Chapter 13, an analytic model for the average daily cost and use it to compute an optimal time between deliveries and an optimal delivery quantity:

$$T^* = \sqrt{\frac{2d}{sr}}$$
$$Q^* = rT^*$$

where

T^* = optimal time between deliveries in days
Q^* = optimal delivery quantity of gasoline in gallons
r = demand rate in gallons per day
d = delivery cost in dollars per delivery
s = storage cost per gallon per day

Inventory Model: Gasoline and Consumer Demand Cont.

Table 5.6 History of demand at a particular gasoline station

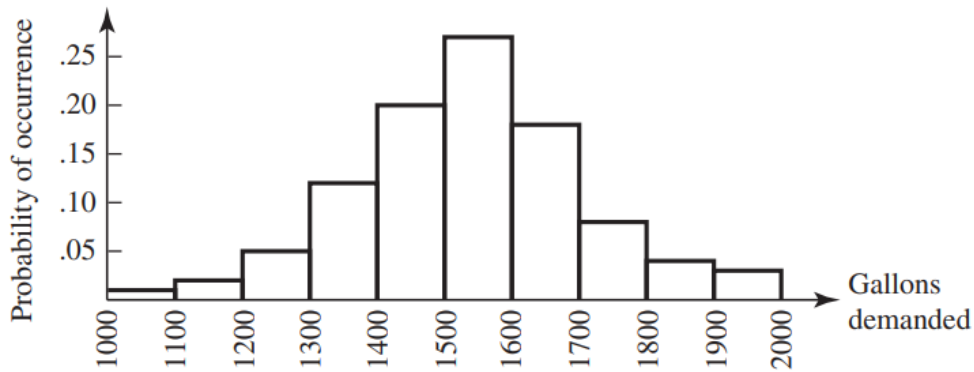
Number of gallons demanded	Number of occurrences (in days)
1000–1099	10
1100–1199	20
1200–1299	50
1300–1399	120
1400–1499	200
1500–1599	270
1600–1699	180
1700–1799	80
1800–1899	40
1900–1999	30
	1000

Table 5.7 Probability of the occurrence of each demand level

Number of gallons demanded	Probability of occurrence
1000–1099	0.01
1100–1199	0.02
1200–1299	0.05
1300–1399	0.12
1400–1499	0.20
1500–1599	0.27
1600–1699	0.18
1700–1799	0.08
1800–1899	0.04
1900–1999	0.03
	1.00

Figure 5.5

The relative frequency of each of the demand intervals in Table 5.7



Inventory Model: Gasoline and Consumer Demand Cont.

Summary of Monte Carlo Inventory Algorithm Terms

Q	Delivery quantity of gasoline in gallons
T	Time between deliveries in days
I	Current inventory in gallons
d	Delivery cost in dollars per delivery
s	Storage cost per gallon per day
C	Total running cost
c	Average daily cost
N	Number of days to run the simulation
K	Days remaining in the simulation
x_i	A random number in the interval [0, 1]
q_i	A daily demand
Flag	An indicator used to terminate the algorithm

Inventory Model: Gasoline and Consumer Demand Cont.

Monte Carlo Inventory Algorithm
Input Q, T, d, s, N
Output c
Step 1 Initialize

$$K = N$$
$$I = 0$$
$$C = 0$$
$$\text{Flag} = 0$$

Step 2 Begin the next inventory cycle with a delivery:

$$I = I + Q$$
$$C = C + d$$

Step 3 Determine if the simulation will terminate during this cycle:
If $T \geq K$, then set $T = K$ and Flag=1.

Step 4 Simulate each day in the inventory cycle (or portion remaining):
For $i = 1, 2, \ldots, T$, do Steps 5-9
Step 5 Generate the random number x_i.
Step 6 Compute q_i using the demand submodel.
Step 7 Update the current inventory: $I = I - q_i$
Step 8 Compute the daily storage cost and total running cost, unless the inventory has been depleted:
If $I \leq 0$, then set $I = 0$and GOTO Step 9.
Else $C = C + I * s$.
Step 9 Decrement the number of days remaining in the simulation:
$K = K - 1$
Step 10 If Flag= 0, then GOTO Step 2. Else GOTO Step 11.
Step 11 Compute the average daily cost: $c = C/N$.
Step 12 Output c.
STOP

Queuing Models

Example 1: A Harbor System

Consider a small harbor with unloading facilities for ships. Only one ship can be unloaded at any one time. Ships arrive for unloading of cargo at the harbor, and the time between the arrival of successive ships varies from 15 to 145 min. The unloading time required for a ship depends on the type and amount of cargo and varies from 45 to 90 min.

We seek answers to the following questions:

1. What are the average and maximum times per ship in the harbor?
2. If the waiting time for a ship is the time between its arrival and the start of unloading, what are the average and maximum waiting times per ship?
3. What percentage of the time are the unloading facilities idle?
4. What is the length of the longest queue?

Queuing Models Cont.

We have the following data for each ship:

	Ship 1	Ship 2	Ship 3	Ship 4	Ship 5
Time between successive ships	20	30	15	120	25
Unloading time	55	45	60	75	80

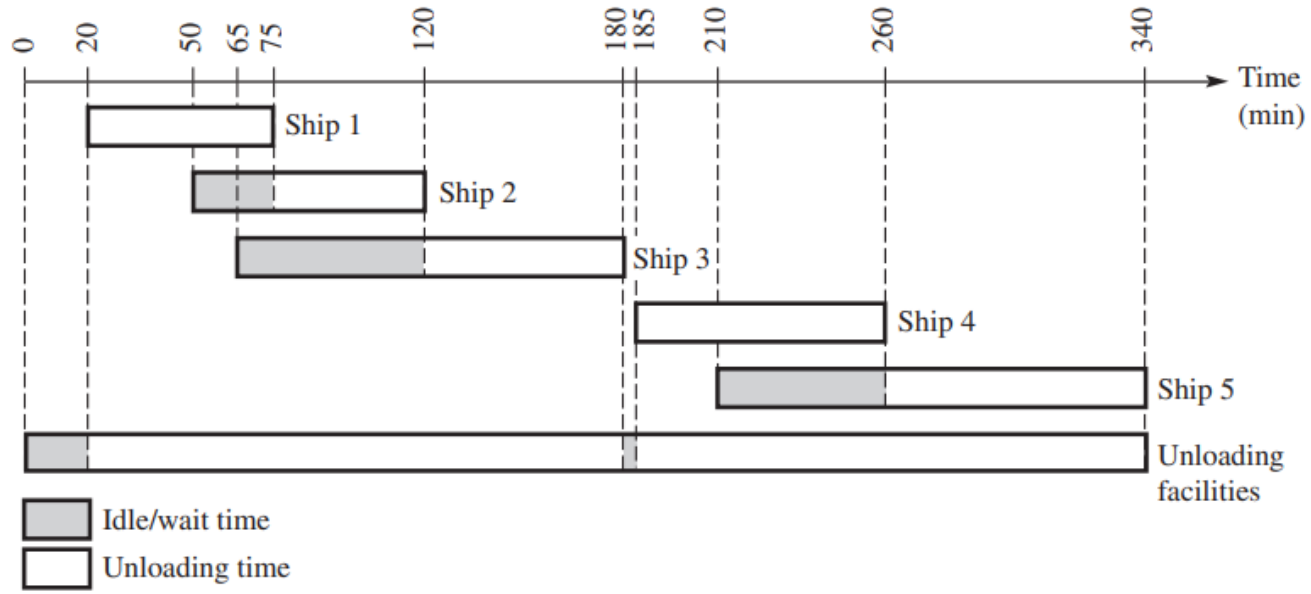


Figure 5.9

Idle and unloading times for the ships and docking facilities

Queuing Models Cont.

Table 5.14 Summary of the harbor system simulation

Ship no.	Random time between ship arrivals	Arrival time	Start service	Queue length at arrival	Wait time	Random unload time	Time in harbor	Dock idle time
1	20	20	20	0	0	55	55	20
2	30	50	75	1	25	45	70	0
3	15	65	120	2	55	60	115	0
4	120	185	185	0	0	75	75	5
5	25	210	260	1	50	80	130	0
Total (if appropriate):					130			25
Average (if appropriate):					26	63	89	

Summary of Harbor System Algorithm Terms

between$_i$	Time between successive arrivals of Ships i and $i-1$ (a random integer varying between 15 and 145 min)
arrive$_i$	Time from start of clock at $t = 0$ when Ship i arrives at the harbor for unloading
unload$_i$	Time required to unload Ship i at the dock (a random integer varying between 45 and 90 min)
start$_i$	Time from start of clock at which Ship i commences its unloading
idle$_i$	Time for which dock facilities are idle immediately *before* commencement of unloading Ship i
wait$_i$	Time Ship i waits in the harbor after arrival before unloading commences
finish$_i$	Time from start of clock at which service for Ship i is completed at the unloading facilities
harbor$_i$	Total time Ship i spends in the harbor
HARTIME	Average time per ship in the harbor
MAXHAR	Maximum time of a ship in the harbor
WAITIME	Average waiting time per ship before unloading
MAXWAIT	Maximum waiting time of a ship
IDLETIME	Percentage of total simulation time unloading facilities are idle

Queuing Models Cont.

Harbor System Simulation Algorithm

Input Total number n of ships for the simulation.
Output HARTIME, MAXHAR, WAITIME, MAXWAIT, and IDLETIME.
Step 1 Randomly generate between_1 and unload_1. Then set $\text{arrive}_1 = \text{between}_1$.
Step 2 Initialize all output values:
$\text{HARTIME} = \text{unload}_1, \quad \text{MAXHAR} = \text{unload}_1,$
$\text{WAITIME} = 0, \quad \text{MAXWAIT} = 0, \quad \text{IDLETIME} = \text{arrive}_1$
Step 3 Calculate finish time for unloading of Ship_1:
$\text{finish}_1 = \text{arrive}_1 + \text{unload}_1$
Step 4 For $i = 2, 3, \ldots, n$, do Steps 5–16.
Step 5 Generate the random pair of integers between_i and unload_i over their respective time intervals.
Step 6 Assuming the time clock begins at $t = 0$ min, calculate the time of arrival for Ship_i:
$\text{arrive}_i = \text{arrive}_{i-1} + \text{between}_i$
Step 7 Calculate the time difference between the arrival of Ship_i and the finish time for unloading the previous Ship_{i-1}:
$\text{timediff} = \text{arrive}_i - \text{finish}_{i-1}$
Step 8 For nonnegative timediff, the unloading facilities are idle:
$\text{idle}_i = \text{timediff} \quad \text{and} \quad \text{wait}_i = 0$
For negative timediff, Ship_i must wait before it can unload:
$\text{wait}_i = -\text{timediff} \quad \text{and} \quad \text{idle}_i = 0$
Step 9 Calculate the start time for unloading Ship_i:
$\text{start}_i = \text{arrive}_i + \text{wait}_i$
Step 10 Calculate the finish time for unloading Ship_i:
$\text{finish}_i = \text{start}_i + \text{unload}_i$
Step 11 Calculate the time in harbor for Ship_i:
$\text{harbor}_i = \text{wait}_i + \text{unload}_i$
Step 12 Sum harbor_i into total harbor time HARTIME for averaging.

Queuing Models Cont.

Step 13 If $\text{harbor}_i > \text{MAXHAR}$, then set $\text{MAXHAR} = \text{harbor}_i$. Otherwise leave MAXHAR as is.

Step 14 Sum wait_i into total waiting time WAITIME for averaging.

Step 15 Sum idle_i into total idle time IDLETIME.

Step 16 If $\text{wait}_i > \text{MAXWAIT}$, then set $\text{MAXWAIT} = \text{wait}_i$. Otherwise leave MAXWAIT as is.

Step 17 Set $\text{HARTIME} = \text{HARTIME}/n$, $\text{WAITIME} = \text{WAITIME}/n$, and $\text{IDLETIME} = \text{IDLETIME}/\text{finish}_n$.

Step 18 OUTPUT (HARTIME, MAXHAR, WAITIME, MAXWAIT, IDLETIME)
STOP

Queuing Models Cont.

Example 2 Morning Rush Hour

We consider a system with four elevators, illustrating multiple-server queues. We discuss the problem and present the algorithm in Appendix B.

Table 5.19 Linear segment submodels provide for the time between arrivals of successive ships as a function of a random number in the interval [0, 1].

Random number interval	Corresponding arrival time	Inverse linear spline
$0 \le x < 0.009$	$15 \le b < 20$	$b = 555.6x + 15.0000$
$0.009 \le x < 0.038$	$20 \le b < 30$	$b = 344.8x + 16.8966$
$0.038 \le x < 0.073$	$30 \le b < 40$	$b = 285.7x + 19.1429$
$0.073 \le x < 0.124$	$40 \le b < 50$	$b = 196.1x + 25.6863$
$0.124 \le x < 0.214$	$50 \le b < 60$	$b = 111.1x + 36.2222$
$0.214 \le x < 0.375$	$60 \le b < 70$	$b = 62.1x + 46.7080$
$0.375 \le x < 0.575$	$70 \le b < 80$	$b = 50.0x + 51.2500$
$0.575 \le x < 0.747$	$80 \le b < 90$	$b = 58.1x + 46.5698$
$0.747 \le x < 0.872$	$90 \le b < 100$	$b = 80.0x + 30.2400$
$0.872 \le x < 0.943$	$100 \le b < 110$	$b = 140.8x - 22.8169$
$0.943 \le x < 0.980$	$110 \le b < 120$	$b = 270.3x - 144.8649$
$0.980 \le x < 0.997$	$120 \le b < 130$	$b = 588.2x - 456.4706$
$0.997 \le x \le 1.000$	$130 \le b \le 145$	$b = 5000.0x - 4855$

Queuing Models Cont.

Table 5.20 Linear segment submodels provide for the unloading time of a ship as a function of a random number in the interval [0, 1].

Random number interval	Corresponding unloading time	Inverse linear spline
$0 \leq x < 0.017$	$45 \leq u < 47.5$	$u = 147x + 45.000$
$0.017 \leq x < 0.062$	$47.5 \leq u < 52.5$	$u = 111x + 45.611$
$0.062 \leq x < 0.157$	$52.5 \leq u < 57.5$	$u = 53x + 49.237$
$0.157 \leq x < 0.243$	$57.5 \leq u < 62.5$	$u = 58x + 48.372$
$0.243 \leq x < 0.373$	$62.5 \leq u < 67.5$	$u = 38.46x + 53.154$
$0.373 \leq x < 0.558$	$67.5 \leq u < 72.5$	$u = 27x + 57.419$
$0.558 \leq x < 0.766$	$72.5 \leq u < 77.5$	$u = 24x + 59.087$
$0.766 \leq x < 0.909$	$77.5 \leq u < 82.5$	$u = 35x + 50.717$
$0.909 \leq x \leq 1.000$	$82.5 \leq u \leq 90$	$u = 82.41x + 7.582$

Table 5.21 Harbor system simulation results for 100 ships

Average time of a ship in the harbor	108	95	125	78	123	101
Maximum time of a ship in the harbor	237	188	218	133	250	191
Average waiting time of a ship	38	25	54	9	53	31
Maximum waiting time of a ship	156	118	137	65	167	124
Percentage of time dock facilities are idle	0.09	0.09	0.08	0.12	0.06	0.10