Chapter 10

Monte Carlo Methods
Chapter Objectives

- Introduce Monte Carlo methods
- Introduce techniques for parallel random number generation
Outline

- Monte Carlo method
- Sequential random number generators
- Parallel random number generators
- Generating non-uniform random numbers
- Monte Carlo case studies
Monte Carlo Method

- Solve a problem using statistical sampling
- Name comes from Monaco’s gambling resort city
- First important use in development of atomic bomb during World War II
Applications of Monte Carlo Method

- Evaluating integrals of arbitrary functions of 6+ dimensions
- Predicting future values of stocks
- Solving partial differential equations
- Sharpening satellite images
- Modeling cell populations
- Finding approximate solutions to NP-hard problems
Example of Monte Carlo Method

\[
\frac{\text{Circle}}{\text{Square}} = \frac{\pi D^2 / 4}{D^2} = \frac{\pi}{4}
\]
Example of Monte Carlo Method

\[ \frac{16}{20} \approx \frac{\pi}{4} \Rightarrow \pi \approx 3.2 \]
Absolute Error

- Absolute error is a way to measure the quality of an estimate
- The smaller the error, the better the estimate
- $a$: actual value
- $e$: estimated value
- Absolute error $= |e-a|/a$
Increasing Sample Size Reduces Error

<table>
<thead>
<tr>
<th>$n$</th>
<th>Estimate</th>
<th>Error</th>
<th>$1/(2n^{1/2})$</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>2.40000</td>
<td>0.23606</td>
<td>0.15811</td>
</tr>
<tr>
<td>100</td>
<td>3.36000</td>
<td>0.06952</td>
<td>0.05000</td>
</tr>
<tr>
<td>1,000</td>
<td>3.14400</td>
<td>0.00077</td>
<td>0.01581</td>
</tr>
<tr>
<td>10,000</td>
<td>3.13920</td>
<td>0.00076</td>
<td>0.00500</td>
</tr>
<tr>
<td>100,000</td>
<td>3.14132</td>
<td>0.00009</td>
<td>0.00158</td>
</tr>
<tr>
<td>1,000,000</td>
<td>3.14006</td>
<td>0.00049</td>
<td>0.00050</td>
</tr>
<tr>
<td>10,000,000</td>
<td>3.14136</td>
<td>0.00007</td>
<td>0.00016</td>
</tr>
<tr>
<td>100,000,000</td>
<td>3.14154</td>
<td>0.00002</td>
<td>0.00005</td>
</tr>
<tr>
<td>1,000,000,000</td>
<td>3.14155</td>
<td>0.00001</td>
<td>0.00002</td>
</tr>
</tbody>
</table>
Mean Value Theorem

\[ \int_{a}^{b} f(x) \, dx = (b - a) \bar{f} \]
Estimating Mean Value

The expected value of \( \frac{1}{n}(f(x_0) + \ldots + f(x_{n-1})) \) is \( \bar{f} \)
Why Monte Carlo Works

\[
\int_a^b f(x) \, dx = (b - a) f \approx (b - a) \frac{1}{n} \sum_{i=0}^{n-1} f(x_i)
\]
Why Monte Carlo is Effective

- Error in Monte Carlo estimate decreases by the factor $1/n^{1/2}$
- Rate of convergence independent of integrand’s dimension
- Deterministic numerical integration methods do not share this property
- Hence Monte Carlo superior when integrand has 6 or more dimensions
Parallelism in Monte Carlo Methods

- Monte Carlo methods often amenable to parallelism
- Find an estimate about $p$ times faster
  OR
- Reduce error of estimate by $p^{1/2}$
Random versus Pseudo-random

- Virtually all computers have “random number” generators
- Their operation is deterministic
- Sequences are predictable
- More accurately called “pseudo-random number” generators
- In this chapter “random” is shorthand for “pseudo-random”
- “RNG” means “random number generator”
Properties of an Ideal RNG

- Uniformly distributed
- Uncorrelated
- Never cycles
- Satisfies any statistical test for randomness
- Reproducible
- Machine-independent
- Changing “seed” value changes sequence
- Easily split into independent subsequences
- Fast
- Limited memory requirements
No RNG Is Ideal

- Finite precision arithmetic $\Rightarrow$ finite number of states $\Rightarrow$ cycles
  - Period = length of cycle
  - If period $> \text{number of values needed}$, effectively acyclic
- Reproducible $\Rightarrow$ correlations
- Often speed versus quality trade-offs
Linear Congruential RNGs

\[ X_i = (a \times X_{i-1} + c) \mod M \]

- Sequence depends on choice of seed, \( X_0 \)
- Multiplier
- Additive constant
- Modulus
Period of Linear Congruential RNG

- Maximum period is $M$
- For 32-bit integers maximum period is $2^{32}$, or about 4 billion
- This is too small for modern computers
- Use a generator with at least 48 bits of precision
Producing Floating-Point Numbers

- $X_i$, $a$, $c$, and $M$ are all integers
- $X_i$'s range in value from 0 to $M-1$
- To produce floating-point numbers in range $[0, 1)$, divide $X_i$ by $M$
Defects of Linear Congruential RNGs

- Least significant bits correlated
  - Especially when $M$ is a power of 2
- $k$-tuples of random numbers form a lattice
  - Especially pronounced when $k$ is large
Lagged Fibonacci RNGs

\[ X_i = X_{i-p} \ast X_{i-q} \]

- \( p \) and \( q \) are lags, \( p > q \)
- \( \ast \) is any binary arithmetic operation
  - Addition modulo \( M \)
  - Subtraction modulo \( M \)
  - Multiplication modulo \( M \)
  - Bitwise exclusive or
Properties of Lagged Fibonacci RNGs

- Require $p$ seed values
- Careful selection of seed values, $p$, and $q$ can result in very long periods and good randomness
- For example, suppose $M$ has $b$ bits
- Maximum period for additive lagged Fibonacci RNG is $(2^p - 1)2^{b-1}$
Ideal Parallel RNGs

- All properties of sequential RNGs
- No correlations among numbers in different sequences
- Scalability
- Locality
Parallel RNG Designs

- Manager-worker
- Leapfrog
- Sequence splitting
- Independent sequences
Manager-Worker Parallel RNG

- Manager process generates random numbers
- Worker processes consume them
- If algorithm is synchronous, may achieve goal of consistency
- Not scalable
- Does not exhibit locality
Leapfrog Method

Process with rank 1 of 4 processes
Properties of Leapfrog Method

- Easy modify linear congruential RNG to support jumping by $p$
- Can allow parallel program to generate same tuples as sequential program
- Does not support dynamic creation of new random number streams
Sequence Splitting

Process with rank 1 of 4 processes
Properties of Sequence Splitting

- Forces each process to move ahead to its starting point
- Does not support goal of reproducibility
- May run into long-range correlation problems
- Can be modified to support dynamic creation of new sequences
Independent Sequences

- Run sequential RNG on each process
- Start each with different seed(s) or other parameters
- Example: linear congruential RNGs with different additive constants
- Works well with lagged Fibonacci RNGs
- Supports goals of locality and scalability
Other Distributions

- Analytical transformations
- Box-Muller Transformation
- Rejection method
Analytical Transformation
Exponential Distribution

\[ F^{-1}(u) = -m \ln u \]

\[ F(x) = 1 - e^{-x/m} \]

\[ f(x) = \frac{1}{m} e^{-x/m} \]
Example 1:

- Produce four samples from an exponential distribution with mean 3
- Uniform sample: 0.540, 0.619, 0.452, 0.095
- Take natural log of each value and multiply by -3
- Exponential sample: 1.850, 1.440, 2.317, 7.072
Example 2:

- Simulation advances in time steps of 1 second
- Probability of an event happening is from an exponential distribution with mean 5 seconds
- What is probability that event will happen in next second?
  - 1/5
- Use uniform random number to test for occurrence of event
Box-Muller Transformation

- Cannot invert cumulative distribution function to produce formula yielding random numbers from normal (gaussian) distribution

- Box-Muller transformation produces a pair of standard deviates $g_1$ and $g_2$ from a pair of normal deviates $u_1$ and $u_2$
Box-Muller Transformation

repeat

\[ v_1 \leftarrow 2u_1 - 1 \]
\[ v_2 \leftarrow 2u_2 - 1 \]
\[ r \leftarrow v_1^2 + v_2^2 \]

until \( r > 0 \) and \( r < 1 \)

\[ f \leftarrow \sqrt{-2 \ln \frac{r}{r}} \]
\[ g_1 \leftarrow f v_1 \]
\[ g_2 \leftarrow f v_2 \]
Example

- Produce four samples from a normal distribution with mean 0 and standard deviation 1

<table>
<thead>
<tr>
<th>$u_1$</th>
<th>$u_2$</th>
<th>$v_1$</th>
<th>$v_2$</th>
<th>$r$</th>
<th>$f$</th>
<th>$g_1$</th>
<th>$g_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.234</td>
<td>0.784</td>
<td>-0.532</td>
<td>0.568</td>
<td>0.605</td>
<td>1.290</td>
<td>-0.686</td>
<td>0.732</td>
</tr>
<tr>
<td>0.824</td>
<td>0.039</td>
<td>0.648</td>
<td>-0.921</td>
<td>1.269</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.430</td>
<td>0.176</td>
<td>-0.140</td>
<td>-0.648</td>
<td>0.439</td>
<td>1.935</td>
<td>-0.271</td>
<td>-1.254</td>
</tr>
</tbody>
</table>
Different Mean, Std. Dev.

\[ g_1 \leftarrow s f v_1 + m \]

\[ g_2 \leftarrow s f v_2 + m \]
Rejection Method
Example

- Generate random variables from this probability density function

\[ f(x) = \begin{cases} 
\sin x, & \text{if } 0 \leq x \leq \pi/4 \\
(-4x + \pi + 8)/(8\sqrt{2}), & \text{if } \pi/4 < x \leq 2 + \pi/4 \\
0, & \text{otherwise}
\end{cases} \]
Example (cont.)

\[ h(x) = \begin{cases} 
1/(2 + \pi/4), & \text{if } 0 \leq x \leq 2 + \pi/4 \\
0, & \text{otherwise}
\end{cases} \]

\[ \delta = (2 + \pi/4)/(\sqrt{2}/2) \]

\[ \delta h(x) = \begin{cases} 
\sqrt{2}/2, & \text{if } 0 \leq x \leq 2 + \pi/4 \\
0, & \text{otherwise}
\end{cases} \]

So \( \delta h(x) \geq f(x) \) for all \( x \)
Example (cont.)

<table>
<thead>
<tr>
<th>$x_i$</th>
<th>$u_i$</th>
<th>$u_i \delta h(x_i)$</th>
<th>$f(x_i)$</th>
<th>Outcome</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.860</td>
<td>0.975</td>
<td>0.689</td>
<td>0.681</td>
<td>Reject</td>
</tr>
<tr>
<td>1.518</td>
<td>0.357</td>
<td>0.252</td>
<td>0.448</td>
<td>Accept</td>
</tr>
<tr>
<td>0.357</td>
<td>0.920</td>
<td>0.650</td>
<td>0.349</td>
<td>Reject</td>
</tr>
<tr>
<td>1.306</td>
<td>0.272</td>
<td>0.192</td>
<td>0.523</td>
<td>Accept</td>
</tr>
</tbody>
</table>

Two samples from $f(x)$ are 1.518 and 1.306
Case Studies (Topics Introduced)

- Neutron transport (Monte Carlo time)
- Temperature inside a 2-D plate (Random walk)
- Two-dimensional Ising model (Metropolis algorithm)
- Room assignment problem (Simulated annealing)
- Parking garage (Monte Carlo time)
- Traffic circle (Simulating queues)
Neutron Transport
**Example**

Monte Carlo Time

<table>
<thead>
<tr>
<th>$D$ (0-$\pi$)</th>
<th>Angle (0-1)</th>
<th>$u$ (0-1)</th>
<th>$L$ (-ln $u$)</th>
<th>$L\cos D$</th>
<th>Dist.</th>
<th>Absorb? (0-1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.00</td>
<td>0.0</td>
<td>0.20</td>
<td>1.59</td>
<td>1.59</td>
<td>1.59</td>
<td>0.41 (no)</td>
</tr>
<tr>
<td>1.55</td>
<td>89.2</td>
<td>0.34</td>
<td>1.08</td>
<td>0.01</td>
<td>1.60</td>
<td>0.84 (no)</td>
</tr>
<tr>
<td>0.42</td>
<td>24.0</td>
<td>0.27</td>
<td>1.31</td>
<td>1.20</td>
<td>2.80</td>
<td>0.57 (no)</td>
</tr>
<tr>
<td>0.33</td>
<td>19.4</td>
<td>0.60</td>
<td>0.52</td>
<td>0.49</td>
<td>3.29</td>
<td></td>
</tr>
</tbody>
</table>
Temperature Inside a 2-D Plate

Random walk
Example of Random Walk

\[ 0 \leq u < 1 \Rightarrow \lfloor 4u \rfloor \in \{0, 1, 2, 3\} \]
2-D Ising Model
Metropolis Algorithm

- Use current random sample to generate next random sample
- Series of samples represents a random walk through the probability density function
- Short series of samples highly correlated
- Many samples can provide good coverage
Metropolis Algorithm Details

- Randomly select site to reverse spin
- If energy is lower, move to new state
- Otherwise, move with probability $\rho = e^{-\Delta/kT}$
- Rejection causes current state to be recorded another time
### Room Assignment Problem

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>3</td>
<td>5</td>
<td>9</td>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>B</td>
<td>3</td>
<td>0</td>
<td>2</td>
<td>6</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>C</td>
<td>5</td>
<td>2</td>
<td>0</td>
<td>8</td>
<td>9</td>
<td>2</td>
</tr>
<tr>
<td>D</td>
<td>9</td>
<td>6</td>
<td>8</td>
<td>0</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>E</td>
<td>1</td>
<td>4</td>
<td>9</td>
<td>3</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>F</td>
<td>6</td>
<td>5</td>
<td>2</td>
<td>4</td>
<td>5</td>
<td>0</td>
</tr>
</tbody>
</table>

Pairing A-B, C-D, and E-F leads to total conflict value of 32.
Physical Annealing

- Heat a solid until it melts
- Cool slowly to allow material to reach state of minimum energy
- Produces strong, defect-free crystal with regular structure
Simulated Annealing

- Makes analogy between physical annealing and solving combinatorial optimization problem
- Solution to problem = state of material
- Value of objective function = energy associated with state
- Optimal solution = minimum energy state
How Simulated Annealing Works

- Iterative algorithm, slowly lower $T$
- Randomly change solution to create alternate solution
- Compute $\Delta$, the change in value of objective function
- If $\Delta < 0$, then jump to alternate solution
- Otherwise, jump to alternate solution with probability $e^{-\Delta/T}$
Performance of Simulated Annealing

- Rate of convergence depends on initial value of $T$ and temperature change function
- Geometric temperature change functions typical; e.g., $T_{i+1} = 0.999 \, T_i$
- Not guaranteed to find optimal solution
- Same algorithm using different random number streams can converge on different solutions
- Opportunity for parallelism
Starting with higher initial temperature leads to more iterations before convergence.
Parking Garage

- Parking garage has $S$ stalls
- Car arrivals fit Poisson distribution with mean $A$
- Stay in garage fits a normal distribution with mean $M$ and standard deviation $M/S$
# Implementation Idea

Times Spaces Are Available

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>101.2</td>
<td>142.1</td>
<td>70.3</td>
<td>91.7</td>
<td>223.1</td>
</tr>
</tbody>
</table>

Current Time | Car Count | Cars Rejected
-------------|-----------|---------------
64.2         | 15        | 2             
Traffic Circle
## Traffic Circle Probabilities

<table>
<thead>
<tr>
<th></th>
<th>F</th>
<th></th>
<th>D</th>
<th>N</th>
<th>E</th>
<th>S</th>
<th>W</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td>0.33</td>
<td></td>
<td>N</td>
<td>0.1</td>
<td>0.2</td>
<td>0.5</td>
<td>0.2</td>
</tr>
<tr>
<td>E</td>
<td>0.50</td>
<td></td>
<td>E</td>
<td>0.3</td>
<td>0.1</td>
<td>0.2</td>
<td>0.4</td>
</tr>
<tr>
<td>S</td>
<td>0.25</td>
<td></td>
<td>S</td>
<td>0.5</td>
<td>0.3</td>
<td>0.1</td>
<td>0.1</td>
</tr>
<tr>
<td>W</td>
<td>0.33</td>
<td></td>
<td>W</td>
<td>0.2</td>
<td>0.4</td>
<td>0.3</td>
<td>0.1</td>
</tr>
</tbody>
</table>
# Traffic Circle Data Structures

<table>
<thead>
<tr>
<th>N</th>
<th>W</th>
<th>S</th>
<th>E</th>
<th>Offset</th>
<th>Arrival</th>
<th>ArrivalCnt</th>
<th>WaitCnt</th>
<th>Queue</th>
<th>QueueAccum</th>
<th>Circle</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>4</td>
<td>8</td>
<td>12</td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>23</td>
<td>22</td>
<td>38</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>13</td>
<td>11</td>
<td>26</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>0</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>37</td>
<td>49</td>
<td>20</td>
<td>41</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **Iteration**: 71

<table>
<thead>
<tr>
<th>N</th>
<th>W</th>
<th>S</th>
<th>E</th>
<th>Offset</th>
<th>Arrival</th>
<th>ArrivalCnt</th>
<th>WaitCnt</th>
<th>Queue</th>
<th>QueueAccum</th>
<th>Circle</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>-1</td>
<td>8</td>
<td>8</td>
<td>12</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>12</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
</tr>
<tr>
<td>12</td>
<td>8</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>15</td>
</tr>
</tbody>
</table>
Summary (1/3)

- Applications of Monte Carlo methods
  - Numerical integration
  - Simulation
- Random number generators
  - Linear congruential
  - Lagged Fibonacci
Summary (2/3)

- Parallel random number generators
  - Manager/worker
  - Leapfrog
  - Sequence splitting
  - Independent sequences

- Non-uniform distributions
  - Analytical transformations
  - Box-Muller transformation
  - Rejection method
Summary (3/3)

- Concepts revealed in case studies
  - Monte Carlo time
  - Random walk
  - Metropolis algorithm
  - Simulated annealing
  - Modeling queues
How to measure the CPU time?

```c
#include <time.h>

clock_t cpu_start, cpu_end;
double cpu_time_used;

cpu_start = clock();
.... program here ..... 
cpu_end = clock();
cpu_time_used = ((double) (cpu_end - cpu_start)) /
                 CLOCKS_PER_SEC;
printf("* Total CPU Time : %f\n",cpu_time_used);
```
Summary (3/3)

- Concepts revealed in case studies
  - Monte Carlo time
  - Random walk
  - Metropolis algorithm
  - Simulated annealing
  - Modeling queues