07m - Extra

Multi-Stream Lehmer Random Number Generators

Multi-Stream Lehmer Random Number Generators are a powerful concept based on the classic **Lehmer (multiplicative) congruential method**, extended to enable **multiple independent streams of random numbers**. This approach is especially valuable in discrete-event simulation where different parts of the model (e.g., customer arrivals, service times, etc.) require independently controlled random sequences.

1. Lehmer Random Number Generator (LRNG) Overview

Lehmer introduced the **multiplicative congruential method**, a foundational random number generator defined by the recurrence relation:

$$X_{i+1} = (aX_i) \mod m$$

Where:

- X_0 is the seed.
- a is the multiplier.
- m is the modulus.

When properly chosen, a and m yield long-period sequences with excellent statistical properties.

2. Multi-Stream Generators

Multi-stream Lehmer generators are constructed to **support multiple independent random number streams**, each initialized with seeds far apart in the total sequence. This is crucial for simulations needing repeatability and control over randomness in different components.

Key Features:

- **Streams** are simply distinct sequences from the same generator, defined by seeds spaced far apart.
- A stream i starts from seed $x_{b(i-1)}$, where b is the spacing factor.
- Large b ensures streams are statistically independent.
- **Substreams** are similarly spaced subdivisions within a stream, used for finer control.

For instance, if $b = 10^{37}$, then:

$$\text{Seed for Stream } i = x_{b(i-1)}$$

This ensures **minimal overlap** between streams, simulating the effect of having multiple RNGs Discrete Event System S... .

3. Example from L'Ecuyer (1988)

To generate long independent sequences, L'Ecuyer proposed combining two Lehmer generators:

Parameters:

- Generator 1: $m_1 = 2,147,483,563$, $a_1 = 40,014$
- Generator 2: $m_2 = 2,147,483,399$, $a_2 = 40,692$

Algorithm:

```plaintext
1. Select seeds X1,0 and X2,0.
2. Compute:
   X1_j+1 = (a1 * X1_j) mod m1
   X2_j+1 = (a2 * X2_j) mod m2
3. Combine:
   X_j+1 = (X1_j+1 - X2_j+1) mod (m1 - 1)
4. Normalize:
   R_j+1 = X_j+1 / (m1 - 1)
```

The period is approximately:

$$\text{Period} = \frac{(m_1 - 1)(m_2 - 1)}{2} \approx 2 \times 10^{18}$$

This is further extended to support **multi-stream capability** by choosing seeds at regular intervals across the sequence Discrete Event System S... .

4. Applications

- Assign one stream per component (e.g., service time, inter-arrival time).
- Ensures **repeatable simulation experiments**.
- Allows **parallel and distributed simulations** without interference among random streams.

If you'd like a visual diagram of how multiple streams and substreams are derived and used in practice, I can find one or draw it for you. Would you like that?

Sample Statistics, Discrete-Data Histogram, Continuous-Data Histogram, Correlation

Here are detailed notes on **Sample Statistics**, **Discrete-Data Histogram**, **Continuous-Data Histogram**, and **Correlation** from *Discrete Event System Simulation (4th Edition)* by Jerry Banks:

---

1. Sample Statistics

Sample statistics are used to summarize and analyze data from simulations or experiments.

Sample Mean ($\bar{X}$):

For raw data $X_1, X_2, \ldots, X_n$:

$$\bar{X} = \frac{1}{n}\sum_{i=1}^{n} X_i$$

Sample Variance (S^2):

$$S^2 = \frac{1}{n-1}\sum_{i=1}^{n}(X_i - \bar{X})^2$$

Grouped Discrete Data:

If values are grouped with frequency f_j for value X_j, then:

$$\bar{X} = \frac{1}{n}\sum_{j=1}^{k} f_j X_j, \quad S^2 = \frac{1}{n-1}\left(\sum f_j X_j^2 - n\bar{X}^2\right)$$

Grouped Data in Class Intervals:

Use class midpoint m_j:

$$\bar{X} = \frac{1}{n}\sum_{j=1}^{c} f_j m_j, \quad S^2 = \frac{1}{n-1}\left(\sum f_j m_j^2 - n\bar{X}^2\right)$$

> These are approximations and used when raw data aren't available Discrete Event System S... .

2. Discrete-Data Histogram

Used when the variable is **discrete** (e.g., number of arrivals):

- Create a **cell (bar)** for each unique value.
- Plot **frequency** (or relative frequency) on the y-axis.
- The shape should resemble a **Probability Mass Function (PMF)**.

Example:

Histogram of vehicle arrivals over 5-minute intervals:

- Data: Number of vehicles per interval over several days.
- Histogram: Each bar represents frequency of a count from 0 to 11.
- Interpretation: Discrete and dense enough to justify per-value cells Discrete Event System S... .

3. Continuous-Data Histogram

Used when data is **continuous** (e.g., lifetime of components):

- Divide the range into **class intervals** (bins).
- Midpoint of each class is used to estimate value.
- Shape should resemble a **Probability Density Function (PDF)**.

Example:

Lifetimes of electronic components recorded:

- Data: Values range widely (e.g., 0.002 to 144.695 days).
- Most values lie in a small initial range.
- Histogram shows skewed distribution.
- Choose bin width carefully to avoid "ragged" or "coarse" histograms Discrete Event System S... .

4. Correlation

Correlation measures **linear dependence** between two variables X_1 and X_2.

Covariance:

$$\text{Cov}(X_1, X_2) = E[(X_1 - \mu_1)(X_2 - \mu_2)]$$

Correlation Coefficient (ρ):

$$\rho = \frac{\text{Cov}(X_1, X_2)}{\sigma_1 \sigma_2}$$

- $\rho = 0$ ⇒ No linear relationship.
- $\rho > 0$ ⇒ Variables increase together.
- $\rho < 0$ ⇒ One increases as the other decreases.
- Values range between **–1 and 1**.

Time-Series Autocorrelation:

$$\rho_h = \text{Corr}(X_t, X_{t+h})$$

- Measures dependence between variables at a **lag h**.
- If dependent only on lag, the time series is **covariance stationary** Discrete Event System S... .

Would you like a diagram showing the difference between discrete and continuous histograms?

Next-Event Simulation, Introduction, Terminology, Algorithmic approach, Examples

Here are detailed notes on **Next-Event Simulation**, covering its introduction, terminology, algorithmic approach, and examples, based on *Discrete Event System Simulation* by Jerry Banks:

---

1. Introduction to Next-Event Simulation

Next-event simulation, also known as **event-scheduling**, is a fundamental technique in **discrete-event simulation (DES)**. It simulates the behavior of a system by:

- Maintaining a **simulation clock**
- Executing **events** in time order from a **Future Event List (FEL)**
- Updating system **state variables** accordingly

It is well-suited for systems where **state changes occur at discrete time points** (e.g., customer arrivals and departures) Discrete Event System S... .

2. Terminology

Term	Definition
Entity	Objects of interest (e.g., customers, servers)
Attribute	Characteristics of entities (e.g., arrival time, service duration)
Event	Instantaneous occurrences that change the state (e.g., arrival, departure)
Activity	Duration-based operation (e.g., waiting, service)
State Variables	Define the current condition of the system (e.g., number in queue)
FEL (Future Event List)	A priority queue of upcoming events ordered by time
Simulation Clock	Tracks the current simulated time

A **system snapshot** is captured at each event time and used to update the FEL and statistics Discrete Event System S... .

3. Algorithmic Approach (Time-Advance Mechanism)

The **Next-Event Time-Advance Algorithm** proceeds through the following steps:

Step-by-Step Algorithm Discrete Event System S... :

1. **Initialize** simulation clock and system state; schedule initial events.
2. **Advance clock** to the next event time from FEL.
3. **Execute the event**, updating the system state and statistics.

4. **Schedule future events** as a result of this event (e.g., a new arrival after current arrival).
5. **Update statistics** (e.g., queue length, waiting times).
6. **Repeat** until a stopping condition is reached (e.g., end time or number of events).

Bootstrapping Example:

Used to generate arrival events recursively: after each arrival, schedule the next using random interarrival time Discrete Event System S... .

---

4. Example: Single-Server Queue

System Setup:

- One server, FIFO queue
- Events: **Arrival (A)**, **Departure (D)**, **End (E)**
- State: $L_Q(t)$ = queue length, $L_S(t)$ = server status (0 or 1)

Event Handling:

- **Arrival (A):**
  - If server idle $\rightarrow$ Start service and schedule **Departure**
  - Else $\rightarrow$ Add customer to queue
  - Schedule next **Arrival**
- **Departure (D):**
  - If queue non-empty $\rightarrow$ Serve next customer, schedule next **Departure**
  - Else $\rightarrow$ Set server to idle

Table Example:

Simulation table tracks:

- Clock time
- Event type
- System state
- Next scheduled events Discrete-Event System S... .

---

5. Advantages of Next-Event Simulation

- Precise modeling of asynchronous events
- Efficient time advancement (no unnecessary computations)
- Basis for most commercial simulation software

Would you like a flowchart or visual diagram showing the event-scheduling cycle and queueing example?

⬤Overview on a Network of Single-Server Service Nodes

Overview of a Network of Single-Server Service Nodes

A **network of single-server service nodes** refers to systems where customers or tasks are routed through multiple service stations, each with a single server. These types of networks are common in many real-world systems such as manufacturing lines, telecommunications, and computer systems.

Key Concepts

1. **Nodes and Service Centers**:
   - **Node**: A service point where customers (entities) arrive, wait, and are serviced.
   - **Single-Server Queue**: Each node has one server. The server may be idle or busy depending on the arrival of customers and the queue length.
2. **Arrival and Routing**:
   - Customers may arrive at different rates and can be routed between multiple nodes in the network.
   - For example, in a **two-node system**, customers arriving at node 1 may be serviced and then routed to node 2 for further processing.
3. **Service Characteristics**:
   - Each node might have different service rates or capacities, with a fixed number of servers (in this case, just one).
   - The **service rate** for each server is crucial for determining how long customers stay in the system.

4. **Queueing Behavior**:
    - Customers arriving at any node may either be served immediately if the server is idle or enter a queue if the server is busy.
    - The departure rate from each node is usually equal to the arrival rate, assuming no creation or destruction of customers within the queueing system .

Mathematical Modeling and Relationships

Several **fundamental principles** apply to networks of queues:

1. **Departure Rate**: The departure rate out of a queue equals the arrival rate into that queue over the long run, assuming stable conditions (no customer loss).
2. **Routing Between Nodes**: If customers arrive at node i at rate λ_i, and a fraction p_{ij} of them are routed to node j, then the arrival rate to node j from node i is $\lambda_i p_{ij}$.
3. **Overall Arrival Rate**: The total arrival rate to a node is the sum of all rates from other nodes in the system. If there are external arrivals λ_i, and internal arrivals from other nodes, the overall rate is the sum of all these inputs .

Stability and Utilization

- **Stability Condition**: For the system to be stable, the arrival rate at any service node should be less than the service rate. If $\lambda > \mu$, where λ is the arrival rate and μ is the service rate, the system becomes unstable, and the queue grows without bound.
- **Server Utilization**: The utilization of each server can be computed as the ratio of the arrival rate to the service rate. For a system with multiple servers, this concept is extended to measure how much each server is busy at any time .

Example of a Simple Network of Single-Server Queues

Consider a **three-node network** where customers first arrive at node 1, get serviced, and are then routed to node 2 and finally to node 3. The arrival rates at each node and the routing probabilities determine the overall performance of the system:

- At **node 1**, customers arrive at a rate of 80 per hour.
- 40% of the customers are routed to **node 2** and 60% to **node 3**.

The system can be modeled using the principles outlined, where each node can be treated as an **M/M/1 queue** (single server, exponential inter-arrival and service times). The long-run server utilization at each node can be calculated based on the arrival and service rates .

Would you like further details on any specific part, or should I generate a diagram of this network?

Random Sampling and Shuffling

Random Sampling and Shuffling

Random Sampling and **Shuffling** are techniques used in simulations, statistics, and data analysis to generate random selections or reorder data. Here's an overview of both concepts:

---

1. Random Sampling

Random sampling refers to the process of selecting a subset of individuals, data points, or elements from a larger population in such a way that each element has an equal chance of being selected. There are different types of random sampling methods:

Simple Random Sampling

Each individual or data point is selected randomly and has an equal probability of being chosen. This is often used when there is no prior grouping or stratification required.

Stratified Random Sampling

The population is divided into distinct subgroups, or strata, and a random sample is taken from each subgroup. This method is useful when specific characteristics of different subgroups need to be represented in the sample.

Systematic Random Sampling

Elements are selected at regular intervals from the population. This method is efficient but requires careful planning to ensure it is random and not biased by the ordering of data.

Example:

In a simulation where you need to generate random arrival times for customers, you can use random sampling to select times from a predefined distribution (e.g., uniform or exponential).

2. Shuffling

Shuffling refers to the process of randomly rearranging the order of elements in a dataset or sequence. It is commonly used when you want to randomize data or ensure that the order of observations doesn't influence results.

Methods of Shuffling:

- **Fisher-Yates Shuffle** (also known as the **Knuth Shuffle**) is the most efficient algorithm for shuffling. It works by iterating over the elements of the list and swapping each element with another randomly chosen one.

Example:

If you have a deck of cards, shuffling ensures that the order of the cards is random, which is crucial for games that rely on randomness, such as poker.

Applications in Simulation

In discrete-event simulations or randomized algorithms, both random sampling and shuffling are important:

- **Random Sampling** is used to simulate different scenarios by drawing samples from random distributions, such as customer arrival times or service durations.
- **Shuffling** ensures that the data or events are processed in a non-biased way, preventing any unintended order effects in the simulation.

Key Considerations

- **Uniform Distribution**: The most basic form of random sampling often assumes a uniform distribution, ensuring that each item has an equal probability of selection.
- **Efficiency**: Random sampling and shuffling can be computationally intensive, especially when dealing with large datasets or many simulations.
- **Reproducibility**: For debugging and result verification, it is important to ensure that random processes can be replicated, often achieved by setting random seeds.

Would you like to see a code example of random sampling or shuffling implemented in Python or another language? Let me know!

Overview on Birth Death Processes, Finite-State Markov Chains

Overview of Birth-Death Processes and Finite-State Markov Chains

1. Birth-Death Processes

A **birth-death process** is a type of **continuous-time Markov process** where the system's state evolves through two types of events: births and deaths. These events can represent various phenomena, such as the arrival of customers or the completion of services.

- **Births** represent the arrival or addition of entities (e.g., customers, jobs).
- **Deaths** represent the departure or completion of entities (e.g., service completions, job finishes).

The birth-death process is primarily used in **queueing theory**, and it models systems like **M/M/1 queues**, where:

- **M** represents a **Markovian (memoryless)** arrival process (Poisson process).
- **M** represents a **Markovian service process** (exponentially distributed service times).
- **1** represents a single server.

In such processes:

- **Transition rates** for births and deaths are denoted by λ (birth rate) and μ (death rate), respectively.
- The system is **continuous-time**, meaning the state of the system can change at any moment in time.

Key Properties:

- The process is **Markovian**: The future state depends only on the current state, not on the path taken to reach it.
- **Stationary Distribution**: Over time, the system reaches a **steady-state** distribution where the probabilities of being in various states stabilize.
- **Ergodicity**: In many birth-death processes, especially **infinite-population models**, the process is **ergodic**, meaning that, over the long term, every state will be visited infinitely often .

2. Finite-State Markov Chains (FSMC)

A **finite-state Markov chain (FSMC)** is a discrete-time process where the system evolves in a series of states, and the probability of moving from one state to another depends solely on the current state.

- **States**: The system can be in one of a finite number of states at any time.
- **Transition Probabilities**: The system transitions between states based on predefined probabilities, which are constant over time.

A typical FSMC can be described by a **transition matrix** P, where each entry $P(i, j)$ represents the probability of transitioning from state i to state j.

Key Properties:

- **Memoryless Property**: Like birth-death processes, FSMCs are **Markovian**, meaning that the future state depends only on the current state.
- **Stationary Distribution**: The **stationary distribution** is the long-run proportion of time the system spends in each state. It is derived from the **balance equations** and can be found using the transition matrix.
- **Recurrent and Transient States**: FSMCs can have **recurrent** states, which the system will return to infinitely often, and **transient** states, which the system may eventually leave permanently .

Applications:

- **Queueing Systems**: FSMCs are used to model customer flow in systems with a finite number of servers or customers.
- **Reliability Modeling**: FSMCs can represent systems with multiple states of health (e.g., operational, degraded, failed) in maintenance or repair systems.

Example:

Consider a **3-state Markov chain** representing a system with three stages: **OFF**, **ON**, and **BURSTY**. The system can transition between these states with fixed probabilities, and each state might have its own arrival rate for customer arrivals (births) and service rate for departures (deaths).

Applications in Simulation:

- **Stochastic Input Models**: In simulations, FSMCs and birth-death processes help model the input processes like arrivals, departures, and state transitions (e.g., customer arrival rates changing with system states).
- **Queueing Models**: Both birth-death processes and FSMCs are integral to **queueing models** like **M/M/c/K** and **M/M/1**, where the states represent customer counts and server statuses, and transitions reflect customer arrivals and service completions .

Would you like a more detailed mathematical formulation or an example code based on these processes?

Monte Carlo Simulation

Monte Carlo Simulation Overview

Monte Carlo simulation is a method for solving problems using random sampling techniques. It is particularly useful when the problem involves uncertainty or complex systems that are difficult to model analytically. This technique is widely used in various fields, including finance, engineering, physics, and operations research.

Basic Concept

Monte Carlo simulation involves running simulations many times (often thousands or millions of iterations) to obtain a distribution of possible outcomes based on random sampling. The idea is to approximate the solution of a problem by simulating random variables that represent the uncertainties in the system and then aggregating the results to draw conclusions.

Steps in Monte Carlo Simulation:

1. **Define a problem**: Identify the parameters and variables that need to be simulated.
2. **Generate random variables**: Use a random number generator to simulate the uncertain parameters of the system. These variables follow a known probability distribution (e.g., uniform, normal).
3. **Run simulations**: Execute the system model using these random variables.
4. **Analyze results**: After running the simulation many times, aggregate the results to estimate the expected outcome, variability, and other statistical measures.

Applications:

- **Risk Analysis**: Monte Carlo simulations are used to model the uncertainty of outcomes, such as estimating the risk in financial investments or supply chain management.
- **Optimization**: It is used for solving complex optimization problems where traditional methods might not work, for example in finding the best inventory levels in an uncertain environment.
- **Engineering and Physics**: In simulations for material science or particle physics, where deterministic solutions are complex or unavailable.

Example in Simulation:

Consider an **artillery simulation** where the goal is to estimate the probability of hitting a target. A Monte Carlo simulation might involve randomly generating coordinates for the artillery's shot and the target, then determining how often the shot hits the target over many iterations.

Random Variables:

- In Monte Carlo simulations, random variables are typically generated using random number generators that follow specific probability distributions. These variables represent real-world uncertainties in the model.
- For example, using a **normal distribution** to model uncertainties in service times or demand rates in a queueing system.

Advantages:

- Can model complex systems with multiple variables and interdependencies.
- Flexibility to handle any form of probability distribution.
- Provides a statistical understanding of possible outcomes, offering insights into risks and uncertainties.

Monte Carlo Example (from the text):

In the context of **random normal number generation**, a Monte Carlo simulation might be used to simulate bombing runs, as discussed in the provided examples, where random normal variables are used to determine the coordinates of each shot, followed by calculating the number of hits or misses .

Would you like further examples or a more detailed explanation on a specific aspect of Monte Carlo simulations?

Follow up