Modeling Randomness

Quiz Study Guide — Chapter 2

Manuel D. Rossetti, PhD P.E.

How to Use This Deck

This deck is a **checklist**, not a summary. Every slide is phrased as something you should be able to *state*, *derive*, or *compute* without looking it up.

- **Question formats:** multiple choice, true/false, matching, fill-in-the-blank, multi-slot fill-in-the-blank, and numeric answers.
- Work the **Drills** slides by hand first; the answers are in the appendix at the end.
- The **Common Traps** slides are built from the actual wrong answers — read them last.

Where this material lives

Chapter 2 of the book. Chapter 2 is the *KSL* view; Appendix A is the theory behind it and Appendix B is the input-modeling side. The three overlap on purpose.

What This Chapter Covers

Random numbers in the KSL

- The L'Ecuyer generator
- Seeds, streams, sub-streams
- Stream control methods
- Common random numbers, antithetic variates

Random variates in the KSL

- The random variable classes and their parameters
- Empirical, mixture, truncated, shifted
- Acceptance-rejection
- Distribution fitting with `PDFModeler`

Objectives — Random Numbers

You should be able to:

- Describe the KSL's generator and quote its period, stream count, and sub-stream length
- Define **seed**, **stream**, and **sub-stream**, and explain what determines each
- Explain why a re-run of a simulation reproduces its results exactly
- State what each stream-control method does
- Define **common random numbers** and the **antithetic** variate of U

Objectives — Random Variates

You should be able to:

- Give the parameters of the common KSL random variable classes
- Explain how `DEmpiricalRV` and `MixtureRV` are constructed
- Name the classes that implement truncation, shifting, and acceptance-rejection
- State which generation method the KSL uses for almost all distributions
- Distinguish the immutable `rvariable` classes from the mutable `Distribution` classes
- Name the four default `PDFModeler` scoring models and describe each

The KSL Random Number Generator

Based on L'Ecuyer et al. (2002) — the combination of **two multiple recursive generators (MRGs)**.

Table 1: Numbers worth memorizing

Property	Value
Period	$\approx 3.1 \times 10^{57}$
Number of streams	$\approx 1.8 \times 10^{19}$
Sub-stream length	$\approx 7.6 \times 10^{22}$

Know these as **powers of ten**: the quiz asks for the exponent alone (57, 19, 22).

Seeds, Streams, and Reproducibility

- A **seed** is the starting value that determines a stream.
- If a simulation always produces the same results when re-run, the most likely cause is that the generator **starts with the same seed each time**. This is a feature: it is what makes debugging and controlled comparison possible.
- Two separately constructed `RNStreamProvider` instances use the **same default underlying seeds**, so they produce **identical** sequences of streams.
- A stream produced by `crnInstance()` is a **clone**; it is *not* managed by the provider and does *not* count as one of the provider's streams.
- A random variable constructed **without** a stream number is automatically given a fresh, unique stream from the default provider.

Stream Control Methods

Table 2: Know these cold — they are a guaranteed matching question

Method	Effect
`resetStartStream()`	Position at the beginning of the sequence assigned when the stream was created
`resetStartSubstream()`	Position at the start of the **current** substream
`advanceToNextSubStream()`	Move to the beginning of the **next** substream
`crnInstance()`	Clone the stream with the same underlying state, for CRN
`antitheticInstance()`	New stream instance configured to produce antithetic variates

`KSLRandom` and the Provider

- `KSLRandom` is the convenient **single entry point**. It wraps a default `RNStreamProvider` and offers a wide range of random-variate-generation methods.
- `KSLRandom.nextRNStream()` calls the underlying default provider to **create a new random number stream**.

> ⚠ Trap
>
> `nextRNStream()` does **not** return the next pseudo-random number U, and it does not advance the default stream to its next substream. It makes a *new stream*.

Common Random Numbers and Antithetics

- **Common Random Numbers (CRN)** is a **variance reduction technique** in which different experiments use the **same** random numbers, so as to *block out the effect of randomness* when comparing them.
- It is **not** a way to ensure independence, and it is **not** a goodness-of-fit test.
- The **antithetic** variate of a pseudo-random number U is $1 - U$.

Random Variable Classes — Parameters

Table 3: The variance-not-standard-deviation point is asked directly

Class	Parameters
`NormalRV`, `LognormalRV`	mean, **variance**
`TriangularRV`	min, mode, max
`UniformRV`	min, max
`GammaRV`	shape, scale
`BinomialRV`	probability of success, number of trials

Discrete Random Variable Classes

Class	Description
`BernoulliRV`	0/1 outcomes from a probability of success
`PoissonRV`	Counts of events in an interval given a mean rate
`DUniformRV`	Discrete uniform over an integer range, min to max
`ShiftedGeometricRV`	Trials until the first success — range starts at **1**
`GeometricRV`	Range starts at **0**
`NegativeBinomialRV`	Failures before the rth success — range starts at 0

Degenerate and Empirical Random Variables

- `ConstantRV` — a degenerate mass on a single value that **cannot** be changed after construction.
- `VConstantRV` — the same, but the value **can** be changed after construction.
- `DEmpiricalRV` is constructed from an array of values **plus an array of cumulative distribution probabilities**, with the last element equal to 1.0.
- `MixtureRV`'s second argument is likewise an array representing the **CDF** over the component random variables, last element 1.0.

> ⚠ Trap
>
> Both take **CDF** arrays, not probability-mass arrays.

Shaping an Existing Distribution

Class	What it does
`TruncatedRV`	Restricts a distribution to a sub-interval of its support
`ShiftedRV`	Wraps an `RVariable` and adds a constant δ to each observation
`AcceptanceRejectionRV`	Generates from a target PDF given a proposal distribution and a majorizing constant

For almost all distributions the KSL generates variates by the **inverse transform method**, using exact inverses where available and numerical approximations otherwise.

Truncated and Shifted — The Algorithms

Truncation to $[a, b]$. Generate $U \sim U(0, 1)$, then

$$W = F(a) + \big(F(b) - F(a)\big)\, U, \qquad X = F^{-1}(W)$$

Shift by δ. If X has CDF F and density f, then $X + \delta$ has density

$$g(x) = f(x - \delta)$$

Generate X by any preferred method and add δ.

Acceptance-Rejection

The algorithm requires **three inputs**: the target PDF $f(x)$, the proposal distribution $w(x)$, and the majorizing constant c.

Chapter example. $f(x) = \frac{3}{4}(1 - x^2)$ on $[-1, 1]$ with majorizing function $g(x) = 3/4$:

$$c = \int_{-1}^{1} \frac{3}{4}\, dx$$

Be able to evaluate that integral and state the resulting acceptance probability.

Mutability and Best Practice

- The classes in `ksl.utilities.random.rvariable` create **immutable** random variables — their parameters **cannot** be changed after construction.
- The `Distribution` classes (e.g. `Binomial`, `Normal`) **do** permit parameter changes, and can create random variables from their current parameter values.
- The `value` property returns a **newly generated** random value each time it is accessed.
- `sample()` on `SampleIfc` returns an **array** of a specified number of generated values.
- **Best practice:** create the random variable object **outside** the generation loop.

Distribution Fitting — `PDFModeler`

The four **default scoring models** for continuous distribution recommendation are **BIC, AD, CVM, and QQC**.

Two things get asked about them:

- **Which four are the defaults.** KS is a KSL goodness-of-fit statistic, but it is *not* one of the four default scoring models.
- **Which direction is better.** Lower AIC and lower BIC are preferred.

The Scoring Metrics

Metric	Description
BIC	Log-likelihood with a penalty for the number of parameters; **lower is better**
AD	Anderson-Darling — sensitive to discrepancies in the **tails**
CVM	Cramer-von Mises — distance between theoretical and empirical CDFs
KS	Kolmogorov-Smirnov — largest vertical distance between the two CDFs
QQC	Pearson **correlation** of empirical and theoretical quantiles in a Q-Q plot

The Fitting Framework Classes

Class or interface	Role
`PDFModeler`	Encapsulates continuous distribution estimation and scoring
`PMFModeler`	Estimates parameters for a set of discrete distributions
`ParameterEstimatorIfc`	Implemented by classes that estimate distribution parameters
`ContinuousCDFGoodnessOfFit`	Computes chi-squared, K-S, AD, and CVM statistics for a continuous CDF
`PDFScoringModel`	Abstract base class for a metric that scores how well a distribution fits

Why Equal-Probability Break Points?

For chi-squared goodness-of-fit testing the KSL uses **equal-probability break points** rather than arbitrary histogram bins.

Reason: each interval then has approximately the **same expected number of observations**, which reduces the test's sensitivity to the choice of intervals.

Drills

Work these by hand. Answers follow at the end of the deck.

1. The L'Ecuyer generator's period is approximately 3.1×10^{57}. Expressed as a power of ten, what is the exponent? What is the exponent for the number of streams?
2. Using the inverse transform for an exponential distribution with mean $\mu = 1.333333$ and $U = 0.7$, compute $X = -\mu \ln(1 - U)$ to four decimal places.
3. For $f(x) = \frac{3}{4}(1 - x^2)$ on $[-1, 1]$ with $g(x) = 3/4$, compute the majorizing constant $c = \int_{-1}^{1} g(x)\, dx$.
4. For the shifted negative binomial with $r = 3$, $p = 0.3$, generated by convolution from $u = 0.35,\ 0.64,\ 0.14$ using $X_i = 1 + \lfloor \ln(1 - u) / \ln(1 - p) \rfloor$, what is the resulting integer sum?

Common Traps — Streams

- `KSLRandom.nextRNStream()` **creates a new stream**; it does not return the next U.
- A `crnInstance()` clone is **not** one of the provider's managed streams.
- Two separately constructed providers **do** give identical stream sequences — they share the same default seeds.
- `resetStartSubstream()` returns to the start of the **current** substream; `advanceToNextSubStream()` moves to the **next** one. Do not swap them.

Common Traps — Random Variables

- `NormalRV` and `LognormalRV` take mean and **variance**, not mean and standard deviation.
- `DEmpiricalRV` and `MixtureRV` take **CDF** arrays ending at 1.0, not PMF arrays.
- `ConstantRV` is the **immutable** one; `VConstantRV` is the mutable one.
- The `rvariable` classes are **immutable**; the `Distribution` classes are **mutable**.
- Create the random variable **outside** the generation loop, not inside it.
- Lower AIC and BIC are **better**.

Appendix — Drill Answers

1. **57** and **19**. (Sub-stream length is 7.6×10^{22}, exponent 22.)
2. $X = -1.333333 \ln(0.3) = -1.333333(-1.203973) = \mathbf{1.6053}$
3. $c = \frac{3}{4} \cdot 2 = \mathbf{1.5}$, so $P_a = 1/c = 2/3$.
4. With $\ln(0.7) = -0.356675$: $u = 0.35 \to X_1 = 2$; $u = 0.64 \to X_2 = 3$; $u = 0.14 \to X_3 = 1$. Sum $= \mathbf{6}$.