

### 1 回归示例动机 (Motivating Example)

考虑一个线性回归模型：

$$Y = X\beta + \varepsilon$$

其中：

- $Y = (y_1, y_2, \dots, y_n)$  为响应变量；
- $X$  为  $n \times p$  的设计矩阵；
- $\varepsilon \sim N(0, \sigma^2 I)$ 。

标准最小二乘估计为：

$$\hat{\beta} = (X^\top X)^{-1} X^\top Y$$

根据经典假设（线性期望与独立同方差误差），可得：

$$\begin{aligned} E\hat{\beta} &= \beta \\ \text{Var}(\hat{\beta}) &= E((\hat{\beta} - E\hat{\beta})^2) = \sigma^2(X^\top X)^{-1} \\ \text{MSPE}(Y^*) &= E(Y^* - \hat{Y})^2 = \sigma^2(1 + X^{*\top}(X^\top X)^{-1} X^*). \end{aligned}$$

其中  $Y^*$  为新样本的预测值。

---

### 1.1 当模型假设不成立时

如果：

- 均值与  $X$  的关系非线性；
- 误差项非独立或异方差；
- 或者我们使用了稳健估计方法（例如抗离群点估计）——

则上述解析结果不再适用。

此时我们可使用 Monte Carlo 模拟 来研究估计量的性质。

---

### 1.2 Monte Carlo 回归模拟的基本思路

若生成  $m$  个独立样本集  $Y_1, \dots, Y_m$  来估计  $\beta$ ，并在第  $i$  个样本集上得到  $\hat{\beta}_i$ ，则：

- 估计期望：

$$\widehat{E}[\hat{\beta}] = \bar{\beta} = \frac{1}{m} \sum_{i=1}^m \hat{\beta}_i$$

- 估计方差：

$$\widehat{\text{Var}}(\hat{\beta}) = \frac{1}{m} \sum_{i=1}^m (\hat{\beta}_i - \bar{\beta})^2$$

Monte Carlo 方法能帮助我们评估在“非标准假设”下的估计稳定性与偏差。

## 2 蒙特卡洛方法基础 (Monte Carlo Basics)

### 2.1 核心思想 (Overview)

我们通常希望估计某个期望：

$$\phi = E_f[h(Y)]$$

其中  $Y \sim f$ ,  $h(Y)$  为感兴趣的函数。

#### 2.1.1 典型示例

若  $h(Y) = I(Y \leq y)$ , 则：

$$\phi = P(Y \leq y) = E_f[I(Y \leq y)]$$

表示累积分布函数  $F(y)$ 。

## 2.2 蒙特卡洛估计量 (Monte Carlo Estimator)

若从分布  $f$  独立采样  $Y_1, \dots, Y_m$ , 则：

$$\hat{\phi} = \frac{1}{m} \sum_{i=1}^m h(Y_i)$$

根据**大数定律**：

$$\hat{\phi} \xrightarrow[m \rightarrow \infty]{} E_f[h(Y)] = \phi$$

在实际模拟中,  $Y_i$  通常是一个完整的数据集, 而非单个观测值。

若每个样本有  $n$  个观测, 则  $Y_i = (Y_{i1}, \dots, Y_{in})$ 。

## 2.3 回归问题中的对应关系 (Back to Regression Example)

若我们希望检验回归估计量是否有偏:

$$\phi = E[\hat{\beta}], \quad h(Y) = \hat{\beta}(Y)$$

则 Monte Carlo 估计为：

$$\hat{\phi} = \frac{1}{m} \sum_{i=1}^m \hat{\beta}_i$$

若要估计方差：

$$\phi = \text{Var}(\hat{\beta}) = E[(\hat{\beta} - E\hat{\beta})^2]$$

则：

$$\hat{\phi} = \frac{1}{m} \sum_{i=1}^m (\hat{\beta}_i - \bar{\hat{\beta}})^2$$

## 2.4 置信区间覆盖率 (Confidence Interval Coverage)

我们可进一步评估置信区间的覆盖概率：

$$h(Y) = I[\beta \in CI(Y)]$$

Monte Carlo 估计：

$$\hat{\phi} = \frac{1}{m} \sum_{i=1}^m I[\beta \in CI(Y_i)]$$

理想情况下， $\hat{\phi} \approx 1 - \alpha$  (例如 0.95)。

覆盖率偏低的原因：

1. 估计的方差过小；
2. 估计量存在偏差。

## 3 模拟误差 (Simulation Uncertainty)

### 3.1 Simulation uncertainty

因为  $\hat{\phi}$  是  $h(Y_i)$  的样本均值，

其方差为：

$$\text{Var}(\hat{\phi}) = \frac{\sigma^2}{m}, \quad \sigma^2 = \text{Var}(h(Y))$$

样本估计为：

$$\hat{\sigma}^2 = \frac{1}{m-1} \sum_{i=1}^m (h(Y_i) - \hat{\phi})^2$$

从而：

$$\widehat{\text{Var}}(\hat{\phi}) = \frac{\hat{\sigma}^2}{m} = \frac{1}{m(m-1)} \sum_{i=1}^m (h(Y_i) - \hat{\phi})^2$$

其阶为  $O(1/m)$ ，标准误差随  $1/\sqrt{m}$  减小。

### 3.2 区分模拟误差与统计误差

#### ⚠ Remark

Monte Carlo 不确定性 ≠ 统计不确定性

- Monte Carlo 不确定性：来自模拟随机数的误差
- 统计不确定性：来自数据生成过程的不确定性

我们可以通过增大  $m$  来减少 Monte Carlo 方差，但其收敛速度为  $\propto 1/\sqrt{m}$ 。

### 3.3 回归示例中的模拟方差 (Regression Example)

可关心的 Monte Carlo 方差包括：

- $\widehat{\text{Var}}(\hat{\beta}) - \beta$ ：偏差估计的不确定性；
- $\widehat{\text{Var}}(\widehat{\text{Var}}(\hat{\beta}))$ ：方差估计的不确定性；
- $\widehat{\text{Var}}(\widehat{\text{MSPE}}(Y^*))$ ：预测误差估计的不确定性。

在每种情况下， $\widehat{\text{Var}}()$  表示估计模拟方差。

---

## | 4 模拟研究的基本步骤 (Basic Steps of a Simulation Study)

### 1. 定义单次实验 (Specify an Experiment)

明确：

- 样本量  $n$
  - 数据分布 (e.g., Normal, t, Poisson)
  - 参数值 (e.g., 均值, 方差, 回归系数)
  - 感兴趣的统计量 (估计量、检验统计量等)
  - 数据生成机制 (DGP: Data Generating Process)
- 

### 2. 确定要变化的输入因子 (Identify Factors to Vary)

- 样本量 ( $n$ )
  - 参数 ( $\beta_1, \sigma^2$ )
  - 分布类型 (Normal / t / skewed)
  - 相关结构 (independent vs correlated)
- 每种组合形成一个 scenario (情景)。
- 

### 3. 编写实验函数 (Write Simulation Code)

- 该函数接收上述输入参数；
- 返回感兴趣的统计量 (如估计偏差、方差等)。



Python

```
1 def simulate_once(n, beta, dist='normal'):  
2     X = np.random.normal(size=n)  
3     if dist == 't':  
4         eps = np.random.standard_t(df=3, size=n)  
5     else:  
6         eps = np.random.normal(size=n)  
7     Y = beta * X + eps  
8     est = np.sum(X * Y) / np.sum(X**2)  
9     return est
```

---

### 4. 重复实验 (Repeat the Experiment m Times)

- 对每个 scenario 重复模拟  $m$  次。
- 这可高度并行化 (并行于 scenario 与 replicate 两个维度)。



Python

```
1 results = [simulate_once(100, 1.0) for _ in range(1000)]
```

---

### 5. 总结与不确定性量化 (Summarize Results & Quantify Uncertainty)

- 计算均值、标准差、偏差、MSE 等；
- 模拟标准误差：

$$SE_{sim} = \frac{s}{\sqrt{m}}$$

- 若  $SE_{sim}$  远小于效应量，可忽略模拟误差。

## 6. 可视化与报告 (Visualization and Reporting)

- 表格呈现 bias、RMSE、coverage；
- 图形化呈现 (箱线图、误差条、曲线对比)。

# 5 各种重要考量 (Design Considerations)

## 1. 效率与可重现性

- 代码应结构化、模块化；
- 记录随机种子以确保可复现。

## 2. 与真实数据相似的结构

- 模拟分布、参数、依赖性应与目标问题相符；
- 可引入异常值、随机效应等以测试鲁棒性。

## 3. 控制随机变异 (Paired Designs)

- 在比较两种方法时，**使用相同的模拟数据集**，即固定随机数种子，使两方法共享相同噪声。
- 分析上可使用配对差值以降低方差。

## 4. 控制随机数的一致性

- 例如，在比较 t 分布与 Normal 分布数据时，可以通过相同的累积分布概率匹配：



Python

```
1 from scipy.stats import t, norm
2 import numpy as np, matplotlib.pyplot as plt
3
4 devs = np.random.normal(size=100)
5 tdevs = t.ppf(norm.cdf(devs), df=1)
6
7 plt.scatter(devs, tdevs)
8 plt.xlabel('Normal Deviates')
9 plt.ylabel('t Deviates (df=1)')
10 plt.plot([min(devs), max(devs)], [min(devs), max(devs)], color='red')
11 plt.show()
```

# 5.1 重复次数 $m$ 的选择 (Choosing $m$ )

- 理论上可用基本功效 (power) 计算决定  $m$ ；
- 实践中常采用**顺序方式 (sequential)**：
  - 逐步增加  $m$ ，直到模拟结果的精度达到可接受水平；
  - 若  $s/\sqrt{m}$  已小于差异效应量，可停止。

# 6 Computational Efficiency

## 6.1 并行计算与仿真研究 (Parallel Processing for Simulations)

- **仿真研究的特点**

仿真 (simulation) 通常是高度可并行的任务 (embarrassingly parallel) :

- 每个重复试验 (replicate) 可以独立运行在不同的 CPU 核上
- 运行结束后再将结果汇总
- 理论上，运行速度应与处理器数量近似线性增长

- **C/C++ 编译优化**

若仿真中包含大量循环或计算密集型部分，可考虑：

- 使用 **C/C++** 编写核心函数
- 在 Python 中通过绑定 (linking) 调用已编译代码
- 这种方式能显著提升性能
- 虽然本课程不涉及，但这是常见的高效策略

---

## 6.2 Python 中的笛卡尔积工具 (Using `itertools.product`)

在仿真设计中，我们常需要遍历多个参数的所有组合。

Python 提供 `itertools.product` 生成参数笛卡尔积：



Python

```
1 import itertools
2
3 thetaLevels = ["low", "med", "hi"]
4 n = [10, 100, 1000]
5 tVsNorm = ["t", "norm"]
6 levels = list(itertools.product(thetaLevels, tVsNorm, n))
```

结果：



Python

```
1 [("low", "t", 10), ("low", "t", 100), ("low", "t", 1000),
2  ("low", "norm", 10), ..., ("hi", "norm", 1000)]
```

**应用场景：**

用于生成实验条件、超参数组合或多场景仿真设定。

---

## 7 Generating Random Uniforms on a Computer

### 7.1 基本思想 (Fundamental Idea)

所有随机变量的生成都可以追溯到标准均匀分布  $U(0, 1)$ 。

我们只需能生成独立均匀随机数，就能通过变换生成各种分布的随机变量。

在计算机上：

- 随机数由 **伪随机数生成器 (RNG)** 产生；
- 其输出为有限的、离散的数值序列；
- 每个 RNG 由**状态 (state)** 与**输出 (output)** 组成；
- **种子 (seed)** 决定 RNG 序列的起始位置。

## 7.2 Sequential Congruential Generators (序列同余生成器)

一种常见的 RNG 类型为**序列同余法**:

$$x_k = f(x_{k-1}, \dots, x_{k-j}) \mod m$$

通常  $j = 1$ , 其中:

- $m$ : 模数 (modulus)
- $f$ : 递推函数
- 输出随机数  $u_k = x_k/m \in [0, 1)$

### 7.2.1 一个例子: 线性同余法 (Linear Congruential Generator, LCG)

最常见形式:

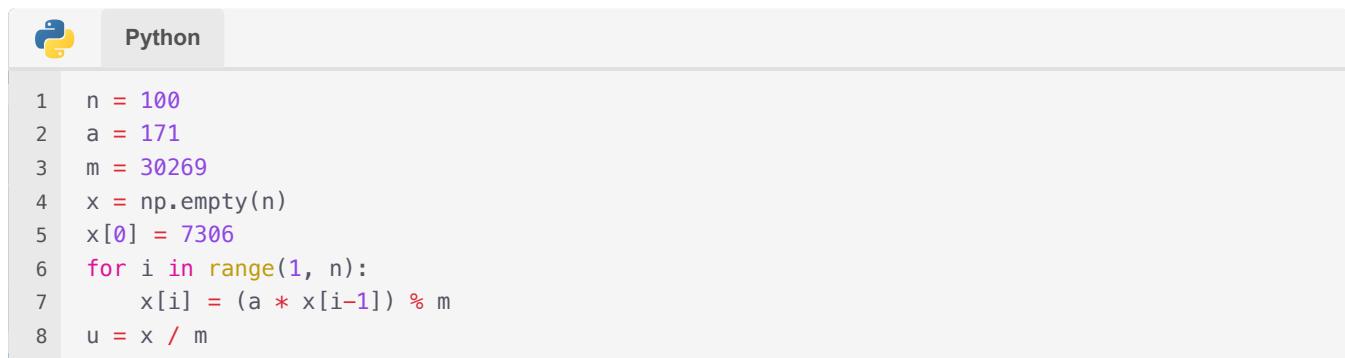
$$x_k = (ax_{k-1} + c) \mod m$$

其中  $a, c, m$  为整数, 且种子为  $x_0$ 。

性质:

- 序列是有限循环的 (周期性);
- 若  $c = 0$ , 最大周期为  $m - 1$ ;
- 典型模数选取为 **Mersenne 素数**  $2^p - 1$ 。

Python 示例:



```
1 n = 100
2 a = 171
3 m = 30269
4 x = np.empty(n)
5 x[0] = 7306
6 for i in range(1, n):
7     x[i] = (a * x[i-1]) % m
8 u = x / m
```

### 7.2.2 RNG 性能评估

一个“好”的 RNG 应满足:

1. 单个样本均匀分布;
2. 各样本独立;
3. 多维情况下呈均匀性 (在高维超立方体中)。

然而, LCG 常在高维空间中呈**格状分布 (lattice structure)**,  
即点落在若干平行超平面上, 而非均匀散布在整个空间中。

## 7.3 Wichmann–Hill 组合生成器

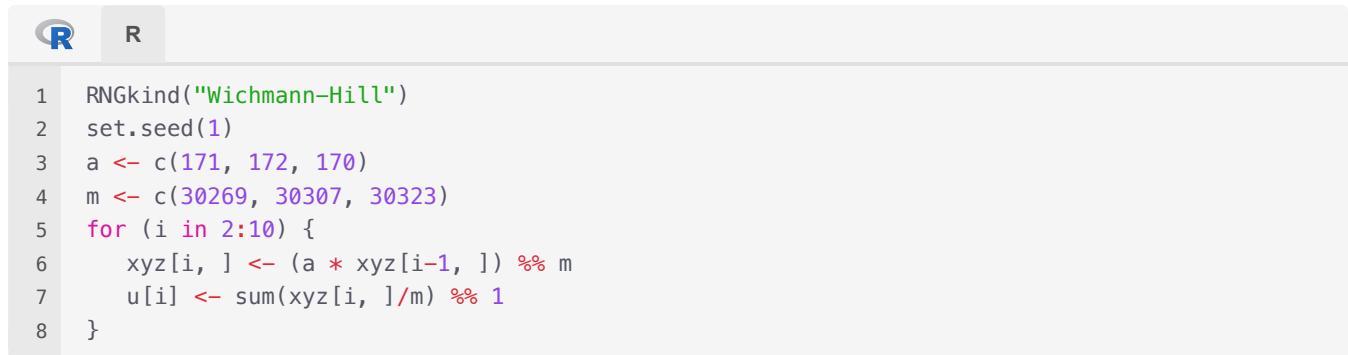
Wichmann–Hill 方法结合了三组参数:

$$a = 171, 172, 170, \quad m = 30269, 30307, 30323$$

生成:

$$u_i = (x_i/30269 + y_i/30307 + z_i/30323) \mod 1$$

R 示例 (手动实现):



```
RNGkind("Wichmann-Hill")
set.seed(1)
a <- c(171, 172, 170)
m <- c(30269, 30307, 30323)
for (i in 2:10) {
  xyz[i, ] <- (a * xyz[i-1, ]) %% m
  u[i] <- sum(xyz[i, ]/m) %% 1
}
```

该算法提升了多维独立性，是早期 R 默认的 RNG。

## 7.4 PCG Generators (Permutation Congruential Generators)

### 7.4.1 核心思想

1. 使用 极大模数  $m = 2^k$  (如 64 或 128 位);
2. 对 LCG 的 states 进行位操作置换 (permutation);
3. 输出经过旋转或位移的结果，提升统计性能。

### 7.4.2 特性

- 状态长度: 64 或 128 bit;
- 周期长度:  $2^{128}$ ;
- 多流机制: 支持  $2^{127}$  个独立随机流;
- 当前 NumPy 默认实现: `numpy.random.default_rng()`。

## 7.5 Mersenne Twister Generator

### 7.5.1 特点

- 周期:  $2^{19937} - 1 \approx 10^{6000}$ ;
- 状态由 624 个 32-bit 整数组成;
- 属于 广义反馈移位寄存器 (GFSR) 类;
- 通过 异或操作 (exclusive-or) 生成伪随机位序列。

## 7.6 Period vs. Unique Values

虽然 RNG 周期极长，但输出值仍有限：

- PCG-64 输出 64 位 → 最多  $2^{64}$  种唯一值;

- MT19937 输出 32 位 → 最多  $2^{32}$  种唯一值。

这意味着可能出现重复值，但不会出现重复状态序列，因此不影响周期特性。

## 7.7 The Seed and the State

- **Seed (种子)**: RNG 序列的起始位置；
- **State (状态)**: RNG 的内部存储，包括寄存器、计数器等；
- **重现性**: 通过相同种子保证相同序列；
- **NumPy**: PCG-64 的状态包含两个 128-bit 整数（状态与增量  $c$ ）。



Python

```
1 rng = np.random.default_rng(seed=1)
2 saved_state = rng.bit_generator.state
3 saved_state['state']['state']    # 状态
4 saved_state['state']['inc']     # 增量 c
```

## 8 RNG in Python

### 8.1 Choosing a Generator



Python

```
1 rng = np.random.default_rng()          # 默认使用 PCG-64
2 rng = np.random.Generator(np.random.MT19937(seed=1)) # 指定 Mersenne Twister
3
4 rng = np.random.Generator(np.random.PCG64(seed = 1)) # PCG-64
```

- `np.random.default_rng()` → 新 API (推荐)
- `np.random` → 旧 API，仍使用 Mersenne Twister

使用时必须通过 `Generator` 对象调用：



Python

```
1 rng.normal(size=3)      # 使用选定生成器
2 np.random.normal(size=3) # 使用旧接口，不受影响
```

### 8.2 Using PCG-64



Python

```
1 rng = np.random.default_rng(seed=1)
2 rng.normal(size=5)
3 # array([ 0.34558419,  0.82161814,  0.33043708, -1.30315723,  0.90535587])
4
5 saved_state = rng.bit_generator.state
```

```

6   rng.normal(size=5)
7   # array([ 0.44637457, -0.53695324,  0.5811181 ,  0.3645724 ,  0.2941325 ])
8
9   rng.bit_generator.state = saved_state # 恢复状态
10  rng.normal(size=5)
11  # array([ 0.44637457, -0.53695324,  0.5811181 ,  0.3645724 ,  0.2941325 ])

```

PCG-64 的状态包含：

- `'state'`：主状态；
- `'inc'`：增量参数。


Python

```

1  saved_state
2  # {'bit_generator': 'PCG64', 'state': {'state': 216676376075457487203159048251690499413,
3    'inc': 194290289479364712180083596243593368443}, 'has_uint32': 0, 'uinteger': 0}
4
5  saved_state['state']['state'] # actual state
6  # 216676376075457487203159048251690499413
7
8  saved_state['state']['inc'] # increment ('c')
9  # 194290289479364712180083596243593368443

```

## 9 RNG in Parallel

### 9.1 并行随机数的潜在问题

并行环境中使用 RNG 时，应避免每个进程使用相同种子，否则所有进程生成相同序列。

### 9.2 正确做法

- 为每个进程指定不同且独立的随机流；
- NumPy 提供多流机制支持（PCG64 的  $2^{127}$  独立流）；

## 10 Summary

生成器	类型	周期长度	状态大小	NumPy 支持	特点
LCG	线性同余	中等	单一整数	手动实现	简单但高维性差
Wichmann–Hill	组合型	长	三组整数	R 可用	改善独立性
PCG-64	置换同余	$2^{128}$	两个 128-bit 整数	NumPy 默认	高性能、支持并行
Mersenne Twister	反馈移位寄存器	$2^{19937} - 1$	$624 \times 32$ -bit	NumPy 旧版、R 默认	稳定且快速

## 11 Multivariate Normal Distributions

若协方差矩阵为  $\Sigma$ ，则可通过 Cholesky 分解 生成：



Python

```
1 L = np.linalg.cholesky(covMat) # L 为下三角矩阵
2 x = L @ np.random.normal(size = covMat.shape[0])
```

此时生成的  $x$  满足  $x \sim \mathcal{N}(0, \Sigma)$ 。

### ⚠ Remark ↴

若协方差矩阵奇异，可使用带 pivoting 的 Cholesky 分解，将秩亏部分对应行置零，从而生成满足约束的随机向量。但需注意输出向量的重新排序。

## 12 Inverse CDF Method (反CDF法)

要生成  $X \sim F$ ，其中  $F$  是累积分布函数 (CDF)，若  $F$  可逆，则可按以下步骤：

1. 生成  $Z \sim \mathcal{U}(0, 1)$ ；
2. 设  $X = F^{-1}(Z)$ 。

即：

$$X = F^{-1}(Z), \quad Z \sim \mathcal{U}(0, 1)$$

对于离散分布，可使用离散化的 CDF。

对于多元分布，可按条件分布顺序采样：

$$f(x_1)f(x_2|x_1)f(x_3|x_1, x_2)\cdots f(x_k|x_1, \dots, x_{k-1})$$

该方法在存在解析条件分布时尤为有效。

## 13 Rejection Sampling (拒绝采样)

### 13.1 基本思想

若目标密度为  $f(x)$ ，我们希望从中采样。

引入辅助变量  $u$ ，则可写作：

$$f(x) = \int_0^{f(x)} du$$

即  $(X, U)$  的联合分布为：

$$(X, U) \sim \mathcal{U}(x, u) : 0 < u < f(x)$$

但我们无法直接从  $f(x)$  采样，因此使用“包络函数”  $cg(x)$  来近似  $f(x)$ ，

其中  $g(x)$  易于采样，且存在常数  $c > 0$  满足：

$$cg(x) \geq f(x), \quad \forall x$$

### 13.2 算法步骤

1. 生成  $x \sim g(x)$ ；
2. 生成  $u \sim \mathcal{U}(0, 1)$ ；
3. 若  $u \leq \frac{f(x)}{cg(x)}$ ，则接受  $x$ ；否则拒绝并重采样。

### 13.3 接受概率

接受率为：

$$P(\text{accept}) = \frac{1}{c} = \frac{\int f(x)dx}{\int cg(x)dx}$$

因此，理想的  $g(x)$  应：

- 具有比  $f(x)$  更“胖的尾部”；
- 使  $c$  尽量小，以减少拒绝次数。

## 13.4 拒绝采样中的 Squeezing 技巧

若  $f(x)$  计算代价高，可使用其下界  $f_{\text{low}}(x)$ ：

$$u \leq \frac{f_{\text{low}}(x)}{cg(x)}$$

满足条件时可直接接受，避免计算完整  $f(x)$ ，从而减少计算负担。

## 13.5 例：截断正态分布

若标准正态分布截断点在  $a > 0$ ，直接采样并拒绝效率极低。

Robert (1995) 提出用平移指数分布作为包络函数：

$$g(x) = \lambda e^{-\lambda(x-a)}, \quad x > a$$

其中  $\lambda$  根据截断点选择，使  $cg(x)$  能包络正态尾部。

当  $a < 0$  时，可通过取负对称实现。

## 14 Importance Sampling

### 14.1 目标

估计期望：

$$\phi = E_f[h(Y)] = \int h(y)f(y)dy$$

若直接从  $f(y)$  采样困难，可引入辅助分布  $g(y)$ ，则：

$$\phi = \int h(y) \frac{f(y)}{g(y)} g(y)dy$$

Monte Carlo 估计量为：

$$\hat{\phi} = \frac{1}{m} \sum_{i=1}^m h(y_i) \frac{f(y_i)}{g(y_i)}, \quad y_i \sim g(y)$$

定义权重：

$$w_i = \frac{f(y_i)}{g(y_i)}$$

若  $f$  仅已知至比例常数（如贝叶斯情境），使用归一化权重：

$$w_i^* = \frac{w_i}{\sum_j w_j}$$

## 14.2 性质与建议

- 只需  $f, g$  有相同 support, 不必要求  $cg(x) \geq f(x)$ ;
- 若  $g$  尾部比  $f$  轻, 方差可能爆炸;
- 为降低方差, 应确保  $w_i$  大时  $h(y_i)$  小, 即避免过度权重。

估计量方差为:

$$\text{Var}(\hat{\phi}) = \frac{1}{m} \text{Var} \left( h(Y) \frac{f(Y)}{g(Y)} \right)$$

启发:

若  $h(y)$  大的区域贡献主要,  $g(y)$  应更多地采样这些区域。

在稀有事件估计中, 这意味着过采样稀有事件, 再用权重修正。

---

## 14.3 Sampling Importance Resampling (SIR)

若目标是从  $f$  中生成样本而非仅估计期望:

1. 从  $g(y)$  生成样本  $y_i$ ;
2. 根据权重  $w_i$  进行有放回重采样。

此过程称为 SIR (Sampling Importance Resampling)。