计算机模拟讲义

王何宇 *
数学科学学院
浙江大学

2018 年 11 月 23 日

*email:wangheyu@zju.edu.cn

目录

第一章 概述 **1**
- 1.1 例子：羊还是车——对概率论结果的数值验证 1
- 1.2 随机优化算法简介 3
- 1.3 本课程内容 6

第二章 随机数的产生 **9**
- 2.1 真随机数产生器 9
- 2.2 早期伪随机数算法 10
  - 2.2.1 经典 Fibonacci 产生器 10
  - 2.2.2 线性同余产生器 12
- 2.3 伪随机数产生器的进展 14
  - 2.3.1 非线性同余产生器 14
  - 2.3.2 多步线性递推产生器 14
- 2.4 梅森旋转 (Mersenne Twister) 产生器 15
  - 2.4.1 初始化 15
  - 2.4.2 梅森旋转 16
  - 2.4.3 递推 17
- 2.5 多维随机数产生方法 18
- 2.6 随机数的检测 18

第三章 随机抽样 **21**
- 3.1 基础知识回顾 21
- 3.2 直接抽样方法 23
  - 3.2.1 逆变换算法 23
  - 3.2.2 列表查找法 23
  - 3.2.3 连续分布的情形 24
- 3.3 取舍算法 27
- 3.4 挤压算法 31
- 3.5 正态分布抽取 33

第四章 Monte Carlo 法：估算体积和计数 **35**
- 4.1 体积估计 35
- 4.2 误差和样本大小 38
- 4.3 置信区间 41
- 4.4 界的估计 45
- 4.5 网络可靠性 48

第一章　概述

本课程主要讨论用如何计算机模拟随机事件，以及以此为基础的优化算法、数值方法和数学模型。随机，或者说不确定性，是我们所处世界的一个基本属性。而我们人类，特别是受到过严格科学训练的人，其思维结构往往是偏向确定性的。这就导致一部分人，甚至是受过高等数学教育的人，在对不确定性事物的直观理解上，会有偏差。最常见的，比如赌徒心理：一个骰子如果已经连开十四次大，那么它下一次开小的概率会不会上升？这个问题相对简单一点，我们来看一个更复杂的游戏。

1.1　例子：羊还是车——对概率论结果的数值验证

这个游戏最早来源于一个北美的电视节目，有点像综艺节目 [33]。最终环节主持人要来宾挑选礼物。礼物藏在三扇门后面，两个门后面是一只山羊，而一扇门后面是一辆汽车。来宾知道礼物是什么，但不清楚哪扇门后面是汽车。他只能随机打开一扇门，得到他的奖品。显然，汽车价值要远高于山羊，所以来宾都更想得到汽车。主持人是知道哪扇门后是汽车的。在来宾挑选了一扇门，尚未打开的时候，主持人会打开一扇藏了山羊的门（注意他总能做到这一点）。此时，来宾有一个选择，是坚持自己的门，还是换一扇？

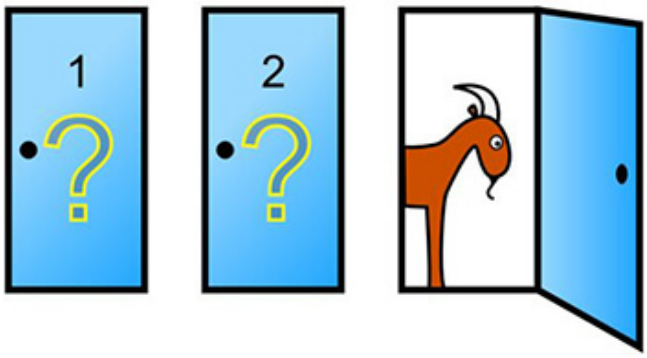

图 1.1: 车还是羊？

这个场景在电影《决胜 21 点》中也有出现。剧中男主角用概率论分析了这个问题：“如果不换，那么选中车的概率就一直保持最初的 1/3；而在主持人打开门以后换一扇门，同时又能选中车的概率则是 2/3。”我们知道好莱坞电影里的数学理论大多数不那么严肃。那么这个主角也是在胡扯么？

选修了这门课的同学都不难自己用概率论验证，主角是正确的。但即便

是一个严格的概率论推导过程，真的就能真正说服我们么？是不是有一部分同学（甚至是大部分同学），始终觉得这就是在玩弄概率论。直觉上似乎始终有一个声音告诉我们，一个确定的选择导致的已经发生的概率，不会因后续事件而改变。

对这件事情最好的回答，无过于我们真的去玩这个节目足够多的次数，比如 10000 次，然后统计不同策略的得失。（注意是统计，这一点电影还是错了，它始终说是上面的分析是一个统计学过程，而事实上应该是概率论。真实的去数充分多的游戏中不同策略的结果，才是统计。统计就是数数字，而概率则是测度论。）于是我们首先需要 10000 辆车和 20000 只羊？如果谁愿意赞助我一定不会反对。但更实际的，我们考虑一下能否在计算机中模拟这 10000 次游戏来达到同样的统计结论。

为方便起见，在这门课中我选择 Python 3.0 来描述算法和实现程序。理由是这是一门最接近自然语言和伪代码风格的语言。同时它又是大数据领域被广泛接受的语言。我们不会专门讲述如何使用 Python 语言编程。而是在使用的过程中不断熟悉这门语言。对于从未接触过 Python 的同学，我们会有一个简单的入门环节。可能在上机课程或者以讲义的形式提供。

我们直接来读代码，Python 的代码非常容易阅读。算法1.1给出了用计算机模拟整个游戏的过程。并提供了 simulation 函数用于测试不同的策略，并分别对两种策略进行了 10000 次模拟，统计了最终结果。一次典型的结果如下（注意每次模拟的结果都会不同，如果是真随机的话）：

```
win rate of never exchange: 0.3357
win rate of exchange: 0.671
```

这个结果清晰地表明在“实际”游戏过程中，换与不换确实会有实质的不同。概率论的结果并不是一个观点，而是客观事实。相反我们的直观并不一定总是正确的。特别对于随机事件，我们要更加小心对待。最后让我们回到对概率论结果的直观理解中。只要考虑一下，如果第一次选错的嘉宾，他在主持人开门之后换一扇门一定可以得到车。而第一次选错的概率就是 2/3。通过这个例子，我们明白计算机模拟确实可以在我们发生怀疑的时候，帮助我们理解一个随机事件的本质，即便已经有了概率估计，它也可以帮助我们建立正确的直观去理解概率结果。

在这一过程中，真正起作用的是两件事情，第一是随机数的产生；第二是计算机能够高效率地产生大量重复试验的结果，并可以对结果进行有效的统计。概率论在这里确实是一个指导性思想。但我们的具体实现始终都是统计过程。这二者能够达成一致的基本原理就是大数定律和中心极限定律。概率论是数学，它的结论是客观事实。但如何理解是一个严重的问题。而统计实现过程也必须足够小心。比如这个算法1.1大家觉得有没有严重的缺陷使得它至多只是一个模拟验证，而不能称为是一个数学结论？

问题就在于我们用计算机产生的“随机数”只能是伪随机数。它具备像一个真正随机序列那样的分布特征，比如这里的随机数是均匀分布的，那么它在均匀性上会表现的和真正的随机序列“几乎”一样。然而，它事实上是有规律的。这使得我们一方面需要认真研究如何才能更高质量和高效率地产生伪随机数，另一方面需要对这件事情有充分的认识，并在设计算法的时候小心谨慎。科研历史上确实发生过由于伪随机序列的隐含规律导致模拟结果错误的事情。

算法 1.1 游戏：羊还是车

输入 total 模拟的次数
输出 显示两种策略的胜率

```
def setup_game():
    # 三扇门初始化成都藏了羊。
    doors = [Prize.goat, Prize.goat, Prize.goat]
    # 先随机挑选一扇门，将羊换成车。
    car = np.random.randint(0, 3)
    doors[car] = Prize.car
    # 嘉宾挑选一扇门，然后主持人打开一扇藏有羊的门。
    guest = np.random.randint(0, 3)
    for host in range(0, 3):
        # 既不是藏有车的门，也不是嘉宾挑选的
        if host is not car and host is not guest:
            break
    return doors, guest, host

# 策略A，打死也不换。这也是个函数，没毛病。
def strategyA():
    # DO NOTHING.
    return

# 策略B，换换更健康。
def strategyB(doors, guest, host):
    for new in range(0, 3):
        # 既不是之前挑选的门，也不是主持人打开的
        if new is not guest and new is not host:
            break
    return new

def simulation(total):
    # 嘉宾和主持人的选择，初始为-1。
    win = 0
    for i in range(total):
        doors, guest, host = setup_game()
        strategyA()
        if doors[guest] is Prize.car:
            win = win + 1
    print("win rate of never exchange:", win/total)
    win = 0
    for i in range(total):
        doors, guest,host = setup_game()
        guest = strategyB(doors, guest, host)
        if doors[guest] is Prize.car:
            win = win + 1
    print("win rate of exchange:", win/total)
    return
```

1.2 随机优化算法简介

除了对一个随机事件进行直接模拟。随机算法还可以用在优化求解上。这一类算法经常被归入 Monte Carlo 方法，也有人将其进一步分化成 Monte Carlo 方法和 Las Vegas 方法。它们分别代表了两种侧重略有区别的策略。不太严格地说，对一个优化问题

$$\min f(x), x \in \mathcal{D}. \tag{1.1}$$

这里 $\mathcal{D} \in \mathbb{R}^m$ 是问题的定义域, m 是维数. Monte Carlo 法的思路是在 $\mathcal{D}$ 中以某种分布进行随机采样 $\xi \in \mathcal{D}$, 并根据 $f(\xi)$ 的结果不断调整分布. 然后随着采样次数 n 的增多，有

$$\lim_{n\to\infty} E(\xi) \overset{1}{\hookrightarrow} x^*.$$

其中 x^* 是问题 (1.1) 的真解。而 Las Vegas 方法则不停地随机投点并根据反馈调整分布，直到在某一次测试中，发现 ξ 严格地等于 x^* 才能结束。显然，Las Vegas 方法可以看作是 Monte Carlo 方法的一个特例。而且它更加适用于离散优化。我们在之后的讨论中不再区分二者。

Monte Carlo 法的实际应用也远比优化要宽广。理论上它几乎可以解决任何问题（在不计效率的前提下）。下面给一个几乎所有 Monte Carlo 讲义都会提到的例子，计算圆面积.

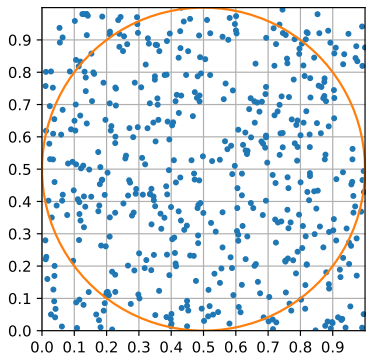

图 1.2: 求圆面积

如图1.2，我们通过随机投点来估计这个内接在单位正方形内部的单位圆的面积。我们要确保每一次投点都是满足在单位正方形内均匀分布的，这样点落在圆内的概率 p 就是圆面积和正方形面积之比。因此我们只要统计落在圆内和所有的点的数目，那么它们二者之比就会在大数定律的控制下依概率收敛到 p。这样就可以计算出圆面积。我们在今后的学习中会用严格的概率理论重建这个过程，而现在先用朴素的语言描述以便于直观理解，好在具体的算法和程序过程仍然是严格的。

算法 1.2 Monte Carlo 法求圆面积

输入 times 抽样次数
输出 圆面积的计算值

```
def area_circle(times):
    inside = 0
    dots = np.random.rand(2, times)
    for i in range(times):
        x = dots[0, i] - 0.5
        y = dots[1, i] - 0.5
        if x * x + y * y < 0.25:
            inside += 1
    return inside / times
```

由于我们在计算圆面积的时候没有使用圆面积公式。因此实际上我们可以通过这个程序来估计 π 值。这个其实不是我们接触到的第一个求圆周率的随机算法。大家还记得上概率论时介绍过的浦丰投针实验么？那就是一个相当硬核的物理随机模拟算法。当然它也能计算机模拟，但是要小心处理边界的情况（针掉到平行线组之外去了）。

我们现在来观察我们求得的 π 值。和所有随机算法一样，它的结果是不稳定的。比如图 1.3中给出了 100 组独立随机模拟的结果，每次模拟都是 10000 个随机投点。可以看到它们在精确的 π 值附近波动。

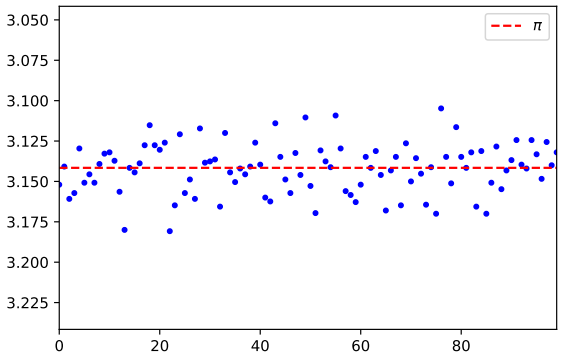

图 1.3: Monte Carlo 法求圆周率，每次 10000 次随机投点，重复了 100 组。

增加投点数量显然可以减少模拟结果的波动，或者说，从随机算法的角度，提高结果的精度。从图 1.4可以看到，随着投点的增加，模拟结果越来越聚集到精确 π 值的附近。这张图也很形象地表现了什么叫依概率收敛。

Monte Carlo 方法的优点和缺点都一样明显。好处是它只基于随机投点，整个过程非常简单机械，非常适合计算机完成。在很多数学上根本无法建模的问题中，它也能很好地实现。它确实是一种“万能”的方法。坏处就是，我们稍微思考一下就能明白，它有效完全依赖于大数定律和中心极限定理。即随着独立试验的增加，均值总是会依概率收敛到（设计好的）期望。那么误差在这里就会收敛到方差！显然，误差是 $O(1/\sqrt{n})$ 的（后续课程我们会精确分析这一过程）。这意味这如果我们在十进制下要增加一位有效数字，那么工作量会增加 100 倍。而一般的数值积分方法，误差一般都

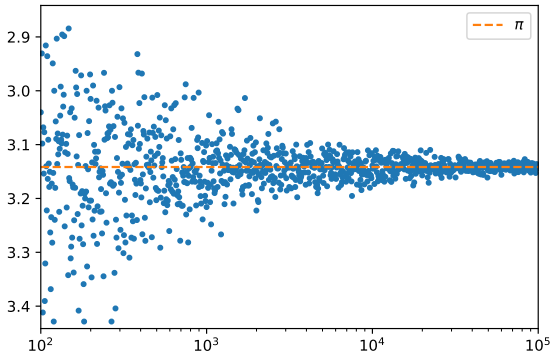

图 1.4: Monte Carlo 法求圆周率，投点数从 10^2 到 10^5，按 log 分布做了 1000 组。

是 $O(1/n^p), p > 1$ 阶的。所以只有在相对复杂，经典数值方法无法给出高效算法的情况下，我们才会考虑 Monte Carlo 方法。比如，极高维度的数值积分。我们注意到，经典数值积分方法是维数相关的，随着空间维数的增加，为了得到同样阶精度，积分点的数量也要以空间指数增加。比如在 3 维要达到 $O(1/n^p), p > 1$ 的精度，相对 1 维来说，积分点也要从 n 个增加到 $O(n^3)$ 个。于是对于一些超高维数问题，经典数值方法变得非常昂贵。然而，大家也许已经注意到，Monte Carlo 方法是维数无关的，不管这个区域是几维，我们要做的就是随机投点，数数。当然，实际上产生一个 m 维空间的随机点，相比 1 维空间，工作量是其 m 倍。但至少只是一个线性增长。因此维数越高，Monte Carlo 方法的优势越明显。

1.3 本课程内容

在这个学期，我们的重点既不在概率论，也不在统计，甚至也不会专门讲太多的计算机编程。（当然概率论是一切的理论基础，所以如果你没有学过概率论，那么赶紧去自学一下也许还能活。）我们将注意力更多地集中在随机算法和相应的技术实现上。这其实是一门优化或者数学规划 (Operations Research) 的分支课程。它变得如此重要的原因相信大家都是理解的。在目前大数据和 AI 领域的最前沿，随机优化算法正在发挥巨大的作用。然而在传统的授课体系中，不论在计算机学院还是数学院，和随机优化有关的内容之前都不是很受重视，至少缺乏系统性的讲述。本课程的设计初衷就是试图弥补这个空缺。在接下去的一个学期中，我们将从随机数的产生和随机抽样开始，逐步讨论 Monte Carlo 方法、模拟退火算法、遗传算法等经典的随机优化方法，介绍它们的原理和实现。同时我们也会重点分析一些比较重要的具体算法，比如神经网络中经常采用的随机梯度下降算法。此外，我们也会解析几个重要案例，来体会这些理论和技术是如何在具体问题中得到应用的。

由于这是一门面向全新领域的全新课程，所以以上的一切都还不是定案。在学习过程中，我们可能随时根据需要增删。同时可能也会有大量的参

考材料和技术资料需要课外阅读。希望通过我们这一学期的努力，能够初步建立起这样一门关键课程的框架和雏形。在这个过程中，需要我们每一位参与者的共同努力。

第二章　随机数的产生

随机模拟的关键是高质量的随机数。但计算机是一个确定性主导的系统。它无法产生真正的随机数。因此，我们要么专门构建特殊的装置来产生真随机数，要么通过一些数学公式来产生性质接近真随机数的伪随机数[34]。

2.1　真随机数产生器

真随机数产生器是一种物理装置，它依赖客观世界中的真实随机现象如基本粒子的随机热运动，量子效应等来产生随机数。这种装置能产生几乎完美的随机数，但往往产生效率不高，且装置本身较为昂贵。一种可以考虑的方法是使用真随机数产生器生成的随机数表。比如网站：
https://www.random.org
就提供这样的服务。表2.1是它的一张价格表。

随着伪随机数理论和实际质量的提高，以及真随机数在巨型计算系统上的应用限制，如 Monte Carlo 法这样随机算法已经没有必要再使用真随机数，高质量的伪随机数生成算法完全可以胜任。在继续讨论之前，我们先给出随机数的定义。

定义 2.1. 随机数 从均匀分布 $U(0,1)$ 抽样得到的简单子样称为随机数，其概率密度函数为

$$f(x) = 1, 0 \leq x \leq 1.$$

随机数用专门符号 U 表示，序列 $\{U_1, U_2, \cdots\}$ 代表独立同分布的随机序列。

Integers
Prices of Example Files

Integers	Size of Range	Price
50,000	10	$5.00
250,000	1,000	$5.00
1,000,000	100	$6.66
5,000,000	1,000	$36.17
10,000,000	1,000	$61.09

表 2.1: 网站 https://www.random.org 提供的随机整数表的价格。

随机数的一个非常重要的性质是高维分布均匀性。由 s 个随机数所组成的 s 维空间上的点

$$(U_{n+1}, U_{n+2}, \cdots, U_{n+s})$$

在 s 维空间的单位正方体 C_s 上是均匀分布的，即 $\forall 0 \leq a_i \leq 1$，$i = 1, 2, \cdots, s$，$U_{n+i} \leq a_i$ 的概率为：

$$P(U_{n+i} \leq a_i, i = 1, 2, \cdots, s) = \prod_{i=1}^{s} a_i. \tag{2.1}$$

用数学方法产生的随机数称为伪随机数。Tezuka 1995 [29] 提出伪随机数应该具有如下特点：

1. 能通过统计检验；
2. 有数学理论支撑；
3. 可以递推产生，不用占用大量内存；
4. 产生速度快，效率高；
5. 重复周期长，至少有 10^{50}；如果问题需要 N 个随机数，则周期需要 $2N^2$；
6. 并行性好，可在计算集群上快速产生；

2.2 早期伪随机数算法

2.2.1 经典 Fibonacci 产生器

Taussky 和 Todd 1956 年 [31] 提出可以用加法同余产生在 $[0, M-1]$ 均匀分布的伪随机整数，其递推公式为

$$X_i = (X_{i-2} + X_{i-1}) \mod M, i > 2. \tag{2.2}$$

当 $X_0 = X_1 = 1$ 时，公式 (2.2) 产生的序列在小于 M 部分就是 Fibonacci 序列，因此又称为经典 Fibonacci 产生器。该算法过程实现很简单。算法2.1给出了它的程序。显然，$U_i = X_i/M$ 给出了相应的 $U(0,1)$ 分布随机数。

算法 2.1 加法同余产生器

输入	X0，X1	初值
	M	最大整数值
	N	随机数个数
输出		N 个服从 $U(0,1)$ 分布的随机浮点数序列

```
1 def rand_add_mod(X0, X1, M, N):
2     X = np.zeros(N)
3     X[0] = X0
4     X[1] = X1
5     for i in range(2, N):
6         X[i] = (X[i - 2] + X[i - 1]) % M
7     return X / (M - 1)
```

尽管在简单的测试中，这个方案给出的伪随机数似乎还不错，比如图2.1中，我们看到在该序列的投点似乎非常均匀。然而，在特殊的抽样公式

$$X_i = \sqrt{U_i}\cos(2\pi U_{i+1})\sin(\pi U_{i+2}) \tag{2.3}$$
$$Y_i = \sqrt{U_i}\sin(2\pi U_{i+1})\sin(\pi U_{i+2}) \tag{2.4}$$

下，我们看到 X_i 和 Y_i 与应有的统计预测不符。见图 2.2。这说明，抽样公式 (2.3) 和 (2.4) 实际上回归出了加法同余产生器的规律。这在实际模拟中是非常危险的。除此以外，文献指出，该算法还有其他较为严重的问题，比如随机序列不能居中 (U_i, U_{i+1}, U_{i+2}) 在 3 维空间分布相关等等。但同余运算在伪随机数生成算法中，仍具有一定的意义。不少现在仍在使用的方法，是经典 Fabonacci 方法的一个改进。

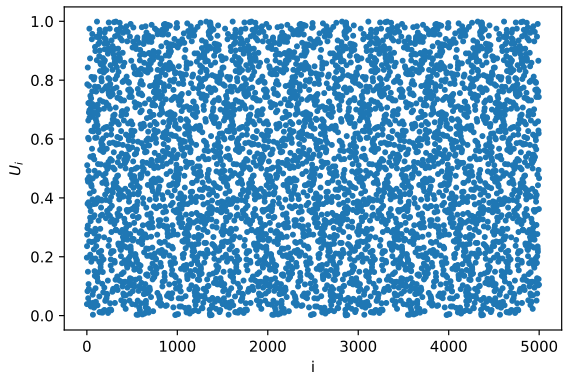

图 2.1: 加法同余产生器的均匀性测试，$X_0 = 197$，$X_1 = 39$, $M = 1000$，一共产生了 5000 个随机数。

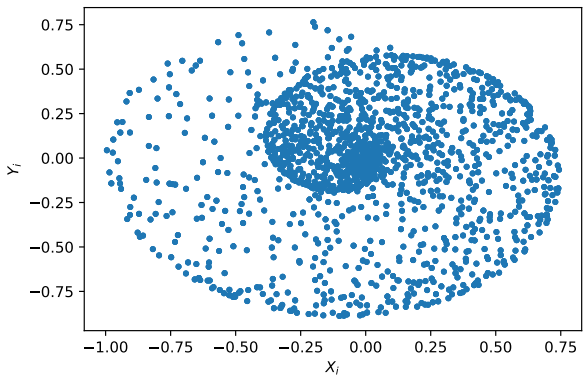

图 2.2: 加法同余产生器的缺陷。在特殊抽样下与统计预测不符。X_i 和 Y_i 分别用公式 (2.3) 和 (2.4) 产生。

2.2.2 线性同余产生器

Lehmer 1951[13] 和 Rotenberg 1960[27] 分别提出了乘同余和混合同余产生器。递推公式为：

$$X_i = (A * X_{i-1} + C) \mod M, \tag{2.5}$$

其中参数 M，A 和 C 分别称为模，乘子和增量。当 $C = 0$ 的特例即是乘同余产生器，而 $C > 0$ 则称为混合同余产生器。同样可以用 $U_i = X_i/M$ 将其调整为 $U(0,1)$ 分布。算法2.2给出了具体的产生程序。

算法 2.2 混合同余产生器

输入	X0	初值
	M	最大整数值
	A，C	人工参数
	N	随机数个数
输出		N 个服从 $U(0,1)$ 分布的随机浮点数序列

```
def rand_mul_mod(X0, M, A, C, N):
    X = np.zeros(N)
    X[0] = X0
    for i in range(1, N):
        X[i] = (A * X[i - 1] + C) % M
    return X / (M - 1)
```

不恰当地选择参数也会有有类似加同余的异常相关现象，图2.3 对参数 $A = 7$，$M = 2^{31} - 1$ 给出在特殊抽样

$$X_i = \sqrt{-2 \ln U_i} \cos(2\pi U_{i+1}) \tag{2.6}$$

$$Y_i = \sqrt{-2 \ln U_i} \sin(2\pi U_{i+1}) \tag{2.7}$$

下的二维分布图。可以看到样本点全部落在一条螺旋线上。

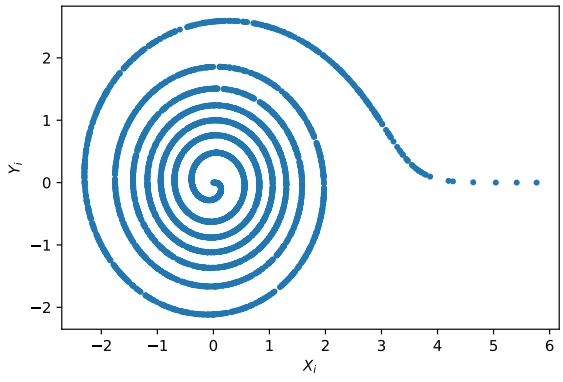

图 2.3: 线性同余产生器的缺陷。X_i 和 Y_i 分别用公式 (2.6) 和 (2.7) 产生。所有的点都落在一条螺旋线上。

线性同余产生器的参数选择需要一些数论的概念。首先引入定义：

定义 2.2. 原根（*Primitive Root*）称线性同余产生器参数乘子 A 是模 M 的原根，当且仅当

1.
$$(A^{M-1}-1) \mod M = 0; \tag{2.8}$$

2. 任取 $I < M-1$, $(A^I-1)/M$ 不是整数.

我们总是希望伪随机数的周期尽可能长（伪随机数周期总是会有的），在 [9] 中，作者指出当模 M 是素数时，当且仅当乘子 A 是其原根时，乘同余产生器的周期是 $M-1$。因此我们一般将 M 取成 $2^\beta-1$（梅森数），这里 β 是机器字长。而对于混合同余产生器，模 M 一般取 2^β，其周期为 M 当且进当 $A=4k+1$，$k\in\mathbb{N}$，且 M 和 C 互质。

线性同余方法曾经广泛使用于各个计算机平台做为伪随机数发生器。比如 IBM 360 系统下曾有一个著名的随机数产生器 RANDU 被使用多年。它就是 $A=65539$，$M=2^{31}$ 的线性乘同余产生器。然而，可以验证，它的三重分布，即 (U_i,U_{i+1},U_{i+2}) 只分布在 15 个空间平面上见图 2.4，它们是：

$$9x-6y+z=k, k=-5,-4,\cdots,8,9. \tag{2.9}$$

这一现象被称为降维，它导致随机数的高维分布均匀性消失。具体推导可参见：
https://en.wikipedia.org/wiki/RANDU

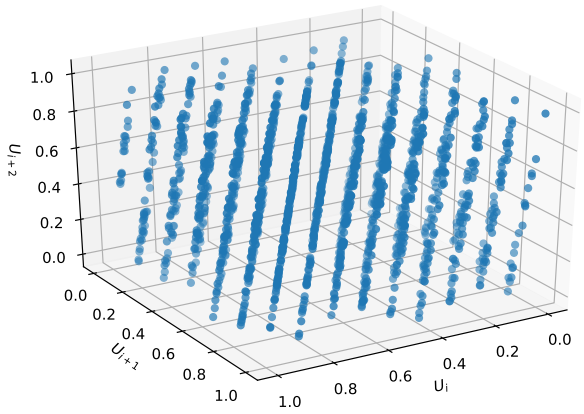

图 2.4: IBM360/370 RANDU 的三维分布稀疏栅格现象。这里 $A=65539$, $M=2^{31}$, $C=0$。

可以证明，降维现象是线性同余产生器的一个必然结果。一般地，对与线性同余产生器产生的伪随机点列构成的 s 维空间点列

$$P=(U_i,U_{i+1},\cdots,U_{i+s-1}), i=1,2,\cdots \tag{2.10}$$

只落在个数不超过 $(s!M)^{1/s}$ 的彼此平行的超平面上（参见：[16]）。

由于以上这些缺陷，目前的主流意见是不要单独使用线性同余产生器。特别是在进行类似 Monte Carlo 模拟时，要格外警惕伪随机数的降维和异

常相关现象。当我们在具体的系统上运算时，也应该先阅读文档，查找具体的随机数产生方式。比如，某些低版本的 GCC 编译器提供的伪随机数就有可能是 $M = 2^{32}$，$A = 1103515245$，$C = 12345$ 的一个混合同余产生器。

2.3 伪随机数产生器的进展

线性同余算法尽管快速简单，但由于它的缺陷，人们陆续提出了一些新的算法以产生更高质量的随机数。主要的改进思路来自以下几个方向：

2.3.1 非线性同余产生器

这种生成器的思想是当 M 是素数时，通过构建某种排列多项式，将 $\{0, 1, \cdots, M-1\}$ 作为一个 Galois 域重排成 $\{g(0), g(1), \cdots, g(M-1)\}$，然后在此基础上构建递推生成器。以继承这种基于重排的均匀性。

比如逆同余产生器：

$$X_i = (AX_{i-1}^{-1} + C) \mod M, i \geq 1. \tag{2.11}$$

这里的 $g(x) = x^{-1}$ 是一种非线性整函数，定义为：

定义 2.3. 乘法逆（*multiplicative inverse*）称 z^{-1} 是 z 关于 M 的乘法逆，如果

$$z \cdot z^{-1} \mod M = 1, z^{-1} \in \mathbb{N}, z^{-1} < M. \tag{2.12}$$

并且规定 0 的乘法逆为 0。

可以验证，对 $M = 11$，$A = 8$，$C = 1$ 和 $X_0 = 0$ 产生的前 9 个数是

$$0, 1, 9, 8, 2, 5, 7, 10, 4, 3,$$

基本上就是 $0 \sim 10$ 的重排（漏掉了 6 以外）。这种产生器可以通过很高维的栅格检验，但是它的生成效率远低于线性同余。关于如何高效求乘法逆涉及数论、群论和计算机算法的高深内容，有兴趣的同学可参见 [14].

此外，常用的非线性同余产生器还有二次或高次同余，二次同余递推公式为：

$$X_i = (AX_{i-1}^2 + BX_{i-1} + C) \mod M. \tag{2.13}$$

更多的内容可自行参考 [9, 34]。

2.3.2 多步线性递推产生器

即增加线性递推的递推层数，一般性的多步线性递推公式为：

$$X_i = (A_1X_{i-1} + A_2X_{i-2} + \cdots + A_jX_{i-j}) \mod M, i \geq j. \tag{2.14}$$

为提高效率，一般我们只保留其中两个系数 A_j 和 A_l 不为零，由此产生了一系列具体的方案：

1. FMRG（$A_1 = 1$，$A_l = B$），

$$X_i = (X_{i-1} + BX_{i-l}) \mod M; \tag{2.15}$$

2. DX-k-2 $(A_1 = A_l = B)$,

$$X_i = B(X_{i-1} + X_{i-l}) \mod M; \tag{2.16}$$

3. DX-k-3 $(A_1 = A_{[l/2]} = A_l = B)$,

$$X_i = B(X_{i-1} + X_{i-[1/2]} + X_{i-l}) \mod M; \tag{2.17}$$

4. DX-k-4 $(A_1 = A_{l/3} = A_{[2l/3]} = A_l = B)$,

$$X_i = B(X_{i-[l/3]} + X_{i-[2l/3]} + X_{i-l}) \mod M. \tag{2.18}$$

以上, $i \geq l$。这一系列公式的极致是 L. Y. Deng 等人 2003 年设计的 DX3803 生成器，周期长达 2^{117923}，而且在 3803 维空间内分布均匀。参见：[6]。

除了上面的方案，常用的改良手段还有进位借位运算产生器 [30]，延迟 Fibonacci 生成器 [5]，线性同余组合产生器 [12, 26]，混合组合产生器 [8, 18, 19] 等等，限于篇幅不一一叙述，有兴趣的同学请自行查阅文献。

2.4 梅森旋转 (Mersenne Twister) 产生器

最后我们介绍一种目前最广泛使用的随机数产生器——梅森旋转产生器。它基于四位日本数学家所发表的一系列论文，Matsumoto（松本）& Kurita（栗田）1992, 1994[21, 22], Matsumoto & Nishimura（西村）1998, 2000 [20, 23], Matsumoto & Saito（斋藤）2008[28]。它的一个重要实例 MT19937 被几乎所有重要编程平台包括并不限于：C，C++，Python，Matlab 等所采纳。它周期长达 $2^{19937} - 1$，能通过目前几乎所有的随机数检验程序，且在至多 623 维空间分布均匀。MT19937 是基于 32 位字长机器运算。下面我们来介绍 MT19937 的具体实现过程。本章代码在 cnblog 上网友 xlxw 的 blog《Python 下探究随机数的产生原理和算法》，网址：
https://www.cnblogs.com/lzxwalex/p/6880748.html
提供的代码基础上修改而成，并参考了 CSDN 网友 tick_tokc97 的 blog《伪随机数生成——梅森旋转（Mersenne Twister/MT）算法笔记》
https://blog.csdn.net/tick_tock97/article/details/78657851。

2.4.1 初始化

梅森旋转需要有一个工作区。MT19937 的工作区是一个长度为 624 的数组，每个元素都是 32 位字长的整数。生成工作区的过程称为初始化。初始化本身也是一个伪随机数产生过程，这个过程采用了类似线性产生器，并混合了位运算以保证产生数字在二进制位的形式上分布均匀。具体公式为：

$$M_i = f * (M_{i-1} \oplus (M_{i-1} >> (w-2))) + i, i = 1, 2, \cdots, 624. \tag{2.19}$$

这里 32 位整数向量 M 存放的就是工作区，M_0 由用户作为种子给出，f 是参数，在 MT19937 中取 1812433253（十六进制：6C078965）。运算符 $\oplus$ 表示异或运算 XOR，w 表示机器字长，在 MT19937 中取 32。具体的算法代码见算法 2.3，

算法 2.3 MT19937 工作区初始化

输入 seed 随机数种子
M 存储梅森工作区的空间
输出 生成的梅森工作区

```
# 转换成32位整数。
def inter(t):
    return(0xFFFFFFFF & t) #截取最后32位

def mainset(seed, M):
    M[0] = seed
    for i in range(1,624):
        M[i] = inter(1812433253 * (M[i - 1] ^ M[i - 1] >> 30) + i
            )
    return M
```

初始化本身就是一个还算不错的伪随机数产生器。

2.4.2 梅森旋转

接下去通过梅森旋转，直接从二进制层面在整个工作区重新分布 0 和 1，并且用旋转和异或操作使二进制数分布均匀。由于工作区的长度是 624，故每产生 624 个数，都要重新执行一次梅森旋转。

$$M_i = M_{i+m} \oplus ((\text{upper\_mask}(M_i)||\text{lower\_mask}(M_{i+1}))A) \tag{2.20}$$

此处，upper_mask(M_i) 和 lower_mask(M_{i+1}) 分别表示取 M_i 和 M_{i+1} 的高 $w-r$ 位和低 r（在 MT19937 中 $w=32$，$r=31$），符号 $||$ 表示将两边的高位和低位连接成一个 32 位的数。下一步 xA 运算的定义为：

$$xA = \begin{cases} x >> 1, & x_0 = 0, \\ (x >> 1) \oplus a, & x_0 = 1. \end{cases} \tag{2.21}$$

这里 x_0 表示 x 的最低位，a 是给定参数，在 MT19937 中设置为 0x9908B0DF（十六进制）。最后将 i 后第 m 个工作区数（在 MT19937 中 $m=397$），即 M_{i+m} 拿来和连接成的数做异或，注意上述下标 $i+1$ 和 $i+m$ 在实际实现中都要和 624 取模，因此实际上整个工作区是循环边界的。这也是梅森旋转的名字中旋转的来历。完整的算法过程见算法 2.4。

算法 2.4 MT19937 梅森旋转

输入 梅森工作区的空间
输出 旋转后的梅森工作区

```
def twister(M):
    for i in range(624):
        # 截取M[i]高位和M[i+1](越界就返回M[0])低位，用普通加法合
            并，对齐32位
        # 这里高位取了1位，低位取了31位。
        y = inter((M[i] & 0x80000000) +(M[(i + 1) % 624] & 0
            x7fffffff))
        yA = y >> 1
        if y & 1 == 1: #取最低位
            yA = yA ^ 0x9908b0df
        M[i] = M[(i + 397) % 624] ^ yA
    return M
```

2.4.3 递推

最后我们可以利用工作区，产生梅森旋转的递推公式：

$$
\begin{aligned}
y &= x \oplus ((x >> u)\&d), && (2.22)\\
y &= y \oplus ((y << s)\&b), && (2.23)\\
y &= y \oplus ((y << t)\&c), && (2.24)\\
z &= y \oplus (y >> l). && (2.25)
\end{aligned}
$$

这里 x 依次从 M 中取，取完 624 个则重新执行梅森旋转。y 是中间变量，z 是返回的下一个随机数。其他 u，d，s，b，t，c 和 l 均为人工参数。在 MT19937 中，$l = 18$，其余参数分别取作

$$
\begin{aligned}
(u, d) &= (11, 0\text{xFFFFFFFF}_{16}),\\
(s, b) &= (7, 0\text{x9D2C5680}_{16}),\\
(t, c) &= (15, 0\text{xEFC60000}_{16}),
\end{aligned}
$$

这里第二列都是十六进制。

最后，完整的随机数生成算法见2.5。

算法 2.5 MT19937 主流程

输入	seed	随机数种子
	num	随机数个数
输出		num 个服从 $U(0,1)$ 分布的随机浮点数序列

```
def exnum(M, index):
    y = M[index]
    y = y ^ y >> 11
    y = y ^ y << 7 & 2636928640
    y = y ^ y << 15 & 4022730752
    y = y ^ y >> 18
    index = index + 1
    return inter(y)

def MT19937(seed, num):
    U = [0]*num
    M = [0]*624
    M = mainset(seed, M)
    twister(M)
    for i in range(num):
        index = i % 624
        U[i] = exnum(M, index) / (2**32 - 1)
        if (index == 623):
            twister(M)
    return U
```

由于梅森旋转当中大量核心算法是位运算，因此运算效率极高，且便于机器底层实现。这也是它成为目前最通用和最受欢迎的伪随机数生成算法。它同样也有 64 和 128 位版本。其作者还在不断提高其计算效率和随机数质量，最新的进展可参见广岛大学提供的网站：
http://www.math.sci.hiroshima-u.ac.jp/ m-mat/MT/emt.html。

2.5 多维随机数产生方法

多维随机数有直接产生的方法，具体参见 Niederreiter 1992[24]，这种方法产生的多维随机数性质较好，但更为简便的方法是基于下述概率论定理用一维随机数间接产生：

定理 2.1. *若 $\{A_i, i = 1, 2, \cdots, ns\}$ 是独立同分布的随机变量序列，则*

$$\{(A_{(i-1)s+1}, A_{(i-1)s+2}, \cdots, A_{is}), i = 1, 2, \cdots\} \tag{2.26}$$

必是 s 维独立同分布的随机向量序列。而且每个分量序列

$$\begin{array}{rcl} \mathrm{U}_1 & = & \{U_1, U_{s+1}, \cdots, U_{(n-1)s+1}\}, \\ \mathrm{U}_2 & = & \{U_2, U_{s+2}, \cdots, U_{(n-1)s+2}\}, \\ & \cdots & \\ \mathrm{U}_s & = & \{U_s, U_{2s}, \cdots, U_{ns}\}, \end{array} \tag{2.27}$$

在各自的维度中也是独立同分布的。

我们用上面的 MT19937 产生一维 50 万个随机数，并按定理次序重新组织成 500 个 1000 维的随机向量。观察一下 (U_1, U_2) 和 (U_{999}, U_{1000}) 平面的随机数分布，都很均匀（见图2.5）。

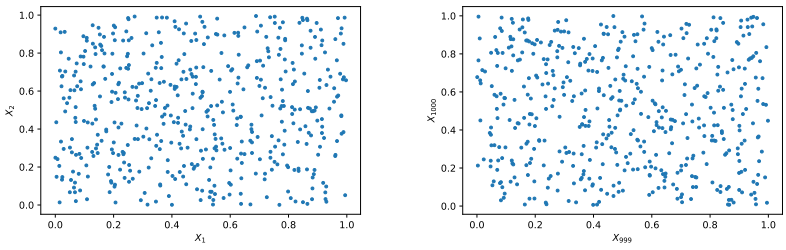

图 2.5: 用 MT19937 生成 50 万个随机数，重新组织成 500 个 1000 维随机向量。左图和右图分别是第 $(1, 2)$ 维和第 $(999, 1000)$ 维平面的随机数分布情况。

2.6 随机数的检测

随机数的理论检测即从随机数的产生原理出发，寻找它的缺陷。比如之前的栅格现象（图2.4）等。实际上这种缺陷也是在应用过程中发现，才被人们所重视，并且在设计随机数产生器时格外注意。理论检验都只针对具体的产生原理，并没有一般性。更多的，我们需要用现代统计学手段对随机数的质量进行统计分析。较为一般的检验，可以是检查随机数的频率，频次，相关性，各阶矩等等。更复杂的检验，则要精确设计统计实验，并涉及到较为复杂和专业的统计学知识。目前被大家所接受的统计学检验程序是 Marsaglia 给出的 DieHard 程序。它由一系列精心设计的复杂统计实验组成，几乎是目前检测随机数质量的标准过程。George Marsaglia 教授已于 2011 年去世，但 DieHard 程序仍然有人不断更新和维护，并引入了 GPL

协议使得可以公开下载和免费使用。具体可参见网站：
http://webhome.phy.duke.edu/ rgb/General/dieharder.php

第三章　随机抽样

上一章我们讨论了最简单的随机分布，均匀分布随机数的产生方法。有了这个均匀分布随机数的基础，这一章我们将进一步讨论如何产生各种所需的随机变量和随机过程。首先我们回顾一下概率论的一些基本定义。

3.1　基础知识回顾

我们用概率分布来描述一个随机变量服从的分布规律，具体来说，离散型随机变量的概率分布称为

定义 3.1. 概率质量函数（*Probability Mass Function, PMF*）若离散型随机变量 ξ 取值为 $x_1, x_2, \cdots, x_n$ 的概率分别为 $p_1, p_2, \cdots, p_n$，即

$$P(\xi = x_i) = p_i, i = 1, 2, \cdots, n, \tag{3.1}$$

则其概率分布函数

$$f(x) = \begin{cases} p_i, & x \in S, \\ 0, & x \in \mathbb{R} \setminus S \end{cases} \tag{3.2}$$

又称为概率质量函数，简称为 *PMF*。其中 $S = \{x_1, x_2, \cdots, x_n\}$ 称为样本集。显然，在离散的情形下，有

$$\sum_{x_i \in S} p_i = \sum_{i=1}^{n} p_i = 1.$$

定义 3.2. 累积分布函数（*Cumulative Distribution Function, CDF*）称

$$F(x) = P(\xi \leq x) = \sum_{x_i \leq x} p_i, x \in \mathbb{R} \tag{3.3}$$

为累积分布函数，简称为 *CDF*。

例 3.1. 两点分布 最简单的两点分布的 *PMF* 为

$$f(x) = \begin{cases} 0.5, & x = 0; \\ 0.5, & x = 1; \\ 0, & x \notin \{0, 1\}. \end{cases} \tag{3.4}$$

对应的 *CDF* 为

$$F(x) = \begin{cases} 0, & x < 0; \\ 0.5, & 0 \leq x < 1; \\ 1, & x = 1. \end{cases} \tag{3.5}$$

更常见的，离散型随机变量的特征会像下面那样表示。

例 3.2. 二项分布 的 *PMF* 一般写做

$$b(k;n,p)=\binom{n}{k}p^k(1-p)^{n-k},0<p<1,k=0,1,\cdots,n. \tag{3.6}$$

我们考虑 $n=10$，$p=0.3$ 的特例，我们可以根据其 PMF 给出对应的二项分布表3.1，以及对应的图3.1。

表 3.1: 二项分布表，$n=10$，$p=0.3$

k	0	1
$P(\xi=k)$	0.028247524900000005	0.12106082100000018
k	2	3
$P(\xi=k)$	0.2334744405	0.26682793200000016
k	4	5
$P(\xi=k)$	0.20012094900000013	0.10291934520000007
k	6	7
$P(\xi=k)$	0.03675690899999999	0.009001692000000002
k	8	9
$P(\xi=k)$	0.0014467004999999982	0.00013778100000000015
k	10	
$P(\xi=k)$	5.904899999999995e-06	

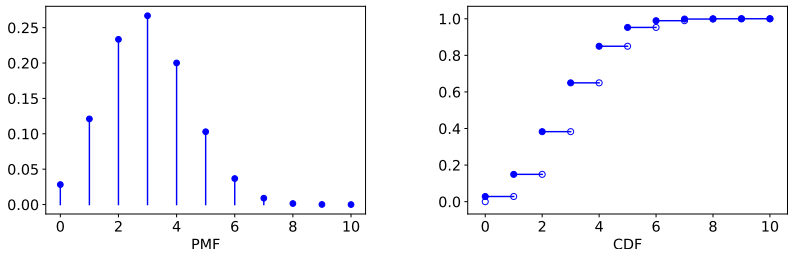

图 3.1: 二项分布，$n=10$，$p=0.3$，左：PMF，右：CDF。

随机模拟的一个重要步骤是根据需要产生服从各种分布的样本集。之前我们已经讨论过如何产生服从 $U(0,1)$ 的均匀分布的随机数，那么现在，我们就要从服从 $U(0,1)$ 的均匀分布的随机序列出发，产生独立同分布的目标随机数序列。这里两个问题的解决是各自独立的。首先随机序列的独立性完全由均匀分布的随机序列的独立性决定，这在上一章已经讨论。我们接下去主要讨论如何确保随机序列同分布，也即和要求的目标分布一致。这里还有一个重要问题是产生效率，因为我们必须在计算机上算法实现。

3.2 直接抽样方法

如果随机变量的 PMF 是已知的，那么我们可以从均匀分布的随机数出发，直接根据概率论定义构建抽样算法。这种方法称为直接抽样方法。即产生均匀分布的随机序列，然后通过某种变换或抽取，使得从中变换抽取后的随机数样本集 X 服从 $F(x)$，这里 $F(x)$ 是已知积累分布函数。也即要求 $\forall \xi \sim F$，

$$P(\xi \leq x) = F(x), x \in \mathbb{R}. \tag{3.7}$$

这里，一方面，我们要从算法保证（证明）X 确实服从指定分布 $F(x)$；另一方面，对于一个实际的产生的抽样结果，也要能通过指定的统计检验。

3.2.1 逆变换算法

1947 年，曼哈顿计划的参与者 Stanislaw Marein Ulam 提出了逆变换算法，注意到随机变量 ξ 的累积分布函数 $F(x) : \mathbb{R} \mapsto [0, 1]$ 是非降的，故定义其逆函数为

$$F^{-1} : [0, 1] \mapsto \mathbb{R}, F^{-1}(y) = \inf\{x \,|F(x) \geq y\} \tag{3.8}$$

定理 3.1. *若随机变量 η 服从 $[0, 1]$ 上的均匀分布，$F(x)$ 是指定分布的 CDF，令 $\xi = F^{-1}(\eta)$，则 ξ 服从 $F(x)$.*

证明： $\forall x \in \mathbb{R}$,

$$P(\xi \leq x) = P(F^{-1}(\eta) \leq x) = P(\eta \leq F(x)) = F(x). \tag{3.9}$$

证毕.

所以我们算法的设计思路就是对一个均匀分布的随机数序列 U，求其逆变换 $X = F^{-1}(U)$，则 X 的分布服从 $F(x)$。

3.2.2 列表查找法

这种做法一般针对离散分布，特别是能给出分布表的离散分布。若离散分布的 PMF 表为

$$\begin{matrix} x_0 & x_1 & \cdots & x_n \\ p_0 & p_1 & \cdots & p_n, \end{matrix}$$

且 $x_0 < x_1 < \cdots < x_n$, 则我们不难构建 CDF 表

$$\begin{matrix} x_0 & x_1 & \cdots & x_k & \cdots & x_n \\ F_0 = p_0 & F_1 = p_0 + p_1 & \cdots & F_k = \sum_{i=0}^{k} p_i & \cdots & F_n \equiv 1. \end{matrix}$$

根据定义3.1, $\forall x \notin \{x_0, x_1, \cdots, x_n\}$, 有 $f(x) = 0$, 故我们可以认为 $F_{-1} = 0$。现在对 $\eta \sim U(0, 1)$, 必有 $0 \leq k \leq n$（为什么?）, 满足

$$F_{k-1} < \eta \leq F_k, \tag{3.10}$$

则可取 $\xi = x_k$。从 PMF 角度，就是

$$\xi = \min \left\{ x_k \middle| \eta \leq \sum_{i=0}^{k} f(x_i) \right\}, \eta \sim U(0, 1). \tag{3.11}$$

在具体的抽取算法中，这一过程就是查表。对于较长的表，应该注意采用二分查找（不知为何经典教材都是顺序遍历，大家仔细想想我有没有 bug?）。具体算法见算法3.1。

算法 3.1 离散随机分布的查表抽样

输入	F	CDF
	eta	$U(0\ 1)$ 分布的随机变量
	start	起始抽样点
	end	终止抽样点
输出		服从 F 分布的随机变量

```
def bisection_search(F, eta, start, end):
    if (eta <= F[start]):
        return start
    n = end - start
    if (n <= 0):
        sys.exit()
    k = (start + end) // 2
    if (eta > F[k]):
        if (eta <= F[k + 1]):
            return k + 1
        else:
            return bisection_search(F, eta, k + 1, end)
    else:
        return bisection_search(F, eta, start, k)
```

图3.2给出了对 $n = 10, p = 0.3$ 的二项分布的 100 万次抽样的统计检验结果。可以看到统计结果和理论 PMF 符合的很好。

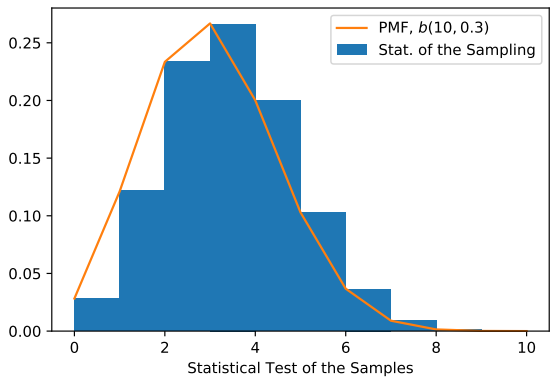

图 3.2: 对算法3.1抽样生成的 $n = 10$，$p = 0.3$ 二项分布的统计检验。柱状图是对 100 万次抽样生成样本集分布的统计，折线是 $b(10, 0.3)$ 的 PMF。二者吻合一致。

3.2.3 连续分布的情形

而对于连续型随机变量，可以看作是离散型随机变量的一种极限情形。此时概率分布无法逐点定义，而是定义在样本集内的正测度集上，即考察随机变量落在一个正测度集内的概率是多少。而从这个角度我们发现，之前在

定义 3.2所引入的累积分布函数

$$F(x) = P(\xi \leq x), x \in \mathbb{R}$$

对连续型随机变量仍然适用，而在此基础上，可进一步引入直观上“概率如何在一点上的定义”。由于一点是零测度集，所以这种定义无法脱离正测度集的概念独立存在，而被看作“微元正测度集”的概率。

定义 3.3. 概率密度函数（*Probability Density Function, PDF*）对连续型随机变量，$\forall x \in \mathbb{R}$, 我们仍称

$$F(x) = P(\xi \leq x), x \in \mathbb{R} \tag{3.12}$$

为累计分布函数，若存在某个非负的可积函数 $f(x)$，满足

$$F(x) = \int_{-\infty}^{x} f(y)dy, \forall x \in \mathbb{R}, \tag{3.13}$$

则称 $f(x)$ 为对应随机分布的概率密度函数，简称为 *PDF*。此时，

$$f(x) = F'(x). \tag{3.14}$$

注意到我们之前定义的针对离散型随机变量的 PMF，事实上和这里定义的 PDF 并无冲突，甚至可以看作是 PDF 的一个特例（这里要看你如何定义积分了）。关于逆变换的定义也可以完全沿用。

表 3.2: 常用连续概率分布的 PDF 和逆变换

概率分布	PDF	逆变换
均匀分布	$\frac{1}{b-a}, x \in [a,b]$	$a + (b-a)y$
Rayleigh 分布	$\frac{x}{\sigma^2} e^{-\frac{x^2}{2\sigma^2}}, x > 0$	$\sqrt{-2\sigma^2 \ln y}$
指数分布	$\frac{1}{\lambda} e^{-\frac{x}{\lambda}}, \lambda > 0, x \geq 0$	$-\lambda \ln y$
Weibull 分布	$\frac{a}{b^a} x^{a-1} e^{-\left(\frac{x}{b}\right)^a}, a, b > 0, x \geq 0$	$b\,(-\ln y)^{\frac{1}{a}}$
Beta 分布	$ax^{a-1}, a > 0, x \in [0,1]$	$y^{\frac{1}{a}}$
Beta 分布	$b(1-x)^{b-1}, b > 0, x \in [0,1]$	$1 - y^{\frac{1}{b}}$
逻辑分布	$\frac{e^{-\frac{x-a}{b}}}{b\left[1+e^{-\frac{x-a}{b}}\right]^2}, b > 0$	$a + b \ln\left[\frac{y}{1-y}\right]$
Cauchy 分布	$\frac{b}{\pi[b^2+(x-a)^2]}$	$a + \frac{b}{\tan(\pi y)}$
正态分布	$\frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$	$*$
Pareto 分布	$\frac{ab^a}{x^{a+1}}, x \geq b > 0, a > 0$	$\frac{b}{(1-y)^{\frac{1}{a}}}$

对于连续分布的采样抽取，如果存在容易计算的逆变换解析表达式 $F^{-1}(y)$，那么对于 $U(0,1)$ 的均匀分布的随机变量 η，直接计算 $F^{-1}(\eta)$ 即

可。也就是说，

$$\xi = \min\left\{x \middle| \eta \le \int_{-\infty}^{x} f(t)dt\right\} = F^{-1}(\eta), \eta \sim U(0,1), \tag{3.15}$$

则 $\xi \sim f(x)$。算法3.2给出了 Rayleigh 分布逆变换抽样的实现。

算法 3.2 Rayleigh 分布的逆变换抽样

输入	U	$U(0,1)$ 分布的随机序列
	sigma	Rayleigh 分布参数
输出		服从 Rayleigh 分布的浮点随机数组

```
1 def sample_Rayleigh(U, sigma):
2     return [np.sqrt(-2 * sigma**2 * np.log(u)) for u in U]
```

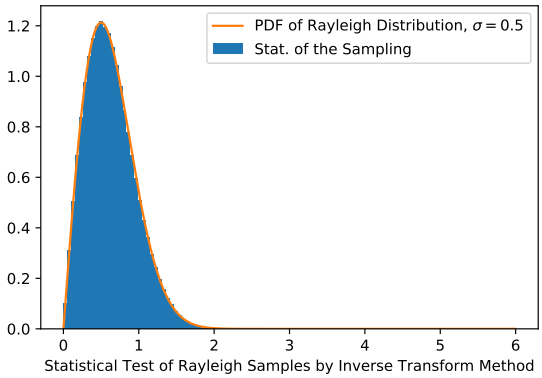

图 3.3: 对算法3.2抽样生成的 Rayleigh 分布（$\sigma = 0.5$）的统计检验。柱状图是对 100 万次抽样生成样本集分布的统计，折线是 Rayleigh 的 PDF。二者吻合一致。

图3.3给出了 Rayleigh 分布 ($\sigma = 0.5$) 的采样结果和 PDF 的对比。可以看到对于能快速计算逆变换的分布，这种方法是最直接的采样手段。表3.2给出了一些常见分布的 PDF 以及相应的逆变换表达式。从表上我们就发现并不是每一个能写出表达式的逆变换在计算上都是高效率的，而有的甚至根本没有解析形式。比如最重要的正态分布我们就没有解析的逆变换形式。不过 Hastings 1955[10]（这是一本神奇的书，给出了各种难以计算公式的有理逼近估计）给出了一个有理逼近形式：

$$F^{-1}(y) = \mu + \text{sign}(y - \frac{1}{2})\sigma\left(t - \frac{c_0 + c_1 t + c_2 t^2}{1 + d_1 t + d_2 t^2 + d_3 t^3}\right), \tag{3.16}$$

其中

$$t = \sqrt{-\ln\left[\min(y, 1-y)\right]^2},$$

各参数值为

$$c_0 = 2.515517, c_1 = 0.802853, c_2 = 0.010328,$$
$$d_1 = 1.432788, d_2 = 0.189269, d_3 = 0.001308.$$

此公式绝对误差小于 $0.45 \times 10^{-}3$。

如果逆变换本身没有解析形式或者难以计算，那么另一个方法是直接从 CDF 入手计算（注意不是 PDF），当然前提是 PDF 和 CDF 可以快速计算。这本质上就是之前离散的情况的查表的极限沿拓，到了连续情形就变成对一个 $\forall y \in [0,1]$，寻找 $x \in \mathbb{R}$，满足

$$F(x) = y \Rightarrow F(x) - y = 0. \tag{3.17}$$

而上述方程，我们可以用数值方法，比如二分法，或 Newton 法求解。

3.3 取舍算法

总是有一些分布，或者写不出反函数的解析表达，或者该解析表达难以计算（或计算量很大）。比如最重要的正态分布就没有显式的积累分布函数的解析表达。而一些初等函数，如指数，对数的计算，事实上计算量是极大的。1947 年，冯 · 诺伊曼 [32] 从条件概率入手，提出了取舍算法 (Accept-Rejection Method，简称 AR 法)。

定理 3.2. *(von Neumann 1951)* 令 $f(z)$，$a \le z \le b$ 是某分布的概率密度函数且具有分解形式

$$f(z) = cg(z)h(z), \tag{3.18}$$

其中

$$h(z) \ge 0, \int_a^b h(z)dz = 1, c = \sup_z \left[\frac{f(z)}{h(z)}\right], \tag{3.19}$$

并且

$$0 \le g(z) \le 1. \tag{3.20}$$

令 Z 是 *PDF* 为 $h(z)$ 的随机变量，U 为服从 $U(0,1)$ 的随机变量。则满足 $U \le g(Z)$ 的随机变量 Z 服从 *PDF* 为 $f(z)$ 的概率分布。

证明. 随机变量 U 的 PDF 为 1，$u \in [0,1]$；而 Z 的 PDF 为 $h(z)$，$z \in [a,b]$，故二者联合分布，即随机向量 (U,Z) 的 PDF 为

$$f_{U,Z}(u,z) = h(z). \tag{3.21}$$

（不严格地，可看作任取 $[0,1] \times [a,b]$ 中的一点，事件恰好落在这一点的概率。把点想像成一个点微元。）现在考虑 Z 的条件概率分布

$$h_Z(z\,|U \le g(Z)\,) = \frac{\int_0^{g(z)} f_{U,Z}(u,z)du}{P(U \le g(Z))}. \tag{3.22}$$

（这个事件说的是，首先 Z 是一个随机变量，它按 $h(Z)$ 的概率分布发生。然后 U 是一个 $U(0,1)$ 分布的随机变量。两件事情同时发生的概率，可以认为是先有 $h(Z)$ 的概率 Z 落在了 z 点，我们还是把它想像成一个微元事件，同时 U 落在 $[0,g(z)]$ 区间上，可以是其中任何一点，所以要积分。这两件事情的全概率是上式的分子。现在我们求的条件概率是，如果 U 和 Z 各自都是独立的，U 在投点的时候并不知道 Z 在哪里，以及 $g(Z)$ 是多少，但我们还是想问这种情况下，$U \le g(Z)$ 的这个事件，关于 Z 的概率密度分布是多少？那么这个就是条件概率分布，它要除以 $U \le g(Z)$ 的概率。）

再由（因为 $f(z)$ 是 PDF）

$$c = \frac{1}{\int_a^b g(z)h(z)dz}, \tag{3.23}$$

以及

$$\int_0^{g(z)} f_{U,Z}(u,z)dx = \int_0^{g(z)} h(z)du = g(z)h(z), \tag{3.24}$$

和

$$P(U \le g(Z)) = \int_a^b h(z)g(z)dz = \frac{1}{c}, \tag{3.25}$$

（Z 落在 $z \in [a,b]$ 点的概率是 $h(z)$，然后此时 $U < g(Z)$ 的概率就是 $g(z)$，故 Z 落在 z 点，同时 $U < g(Z)$ 的单点微元概率是 $h(z)g(z)$，对 Z 取遍全部可能性积分，就得到全部可能的概率。注意和上面分子的不同。分子求的是一个边际分布，是一个关于 Z 的概率密度分布，而分母是一个概率，是一个数。在条件概率公式下，构成了所求的条件概率密度分布函数。）最终有

$$h_Z(z\,|U \le g(Z)\,) = cg(z)h(z) = f(z). \tag{3.26}$$

□

根据定理结果，如果我们已知 $g(z)$ 和 $h(z)$，则可以构建算法产生服从分布 $f(z)$ 的随机变量。基本步骤为：

1. 产生服从分布 $h(z)$ 的随机变量 Z；
2. 产生服从 $U(0,1)$ 的随机变量 U；
3. 计算 $g(Z)$；
4. 若 $U \le g(Z)$，返回 Z，否则放弃，返回第 1 步.

例 3.3. 标准半正态分布（*half-normal distribution*）的 *PDF* 为：

$$f(z) = \sqrt{\frac{2}{\pi}} e^{\frac{-z^2}{2}}, 0 \le z < \infty. \tag{3.27}$$

我们可以将其写作

$$f(z) = \sqrt{\frac{2e}{\pi}} e^{\frac{-(z-1)^2}{2}} e^{-z}, \tag{3.28}$$

并令

$$h(z) = e^{-z}, \tag{3.29}$$

$$g(x) = e^{\frac{-(z-1)^2}{2}}, \tag{3.30}$$

以及

$$c = \sqrt{\frac{2e}{\pi}} \approx 1.3155. \tag{3.31}$$

相应的，我们可以构建算法3.3, 产生标准半正态分布。

算法 3.3 AR 法抽取标准半正态分布

输入	N	实际抽样总数
输出	k	实际接受样本总数
	X	输出服从标准半正态分布的样本（前 k 个有效）

```
def sample_half_normal(N):
    k = 0 # 实际接受总数
    U = np.random.rand(N) # 产生均匀分布
    X = [-np.log(u) for u in U] # 产生服从H的随机变量X
    G = [np.exp(-(x-1)**2/2) for x in X] # 计算G(X)
    U = np.random.rand(N) # 再次产生均匀分布
    for i in range(N):
        if U[i] <= G[i]: # 在G发生的条件下接受
            X[k] = X[i]
            k = k + 1
    return k, X
```

有了标准半正态分布样本集 X，不难通过多一层均匀分布随机抽取将其变换为标准正态分布。也即如果随机变量 $\omega \sim U(0,1)$，　遍历 $x \in X$，若 $\omega \leq 0.5$，则 $x = -x$，否则 $x = x$. 则改造后的 X 服从标准正态分布。这一过程应该结合在 AR 过程之中，完整的标准正态分布算法见算法 3.4。

算法 3.4 AR 法抽取标准正态分布

输入	N	实际抽样总数
输出	k	实际接受样本总数
	X	输出服从标准正态分布的样本（前 k 个有效）

```
def sample_normal(N):
    k = 0 # 实际接受总数
    U = np.random.rand(N) # 产生均匀分布
    X = [-np.log(u) for u in U] # 产生服从H的随机变量X
    G = [np.exp(-(x-1)**2/2) for x in X] # 计算G(X)
    U = np.random.rand(N) # 再次产生均匀分布
    for i in range(N):
        if U[i] <= G[i]: # 在G发生的条件下接受
            w = np.random.rand()
            if w <= 0.5:
                X[k] = X[i]
            else:
                X[k] = -X[i]
            k = k + 1
    return k, X
```

图3.4给出了标准半正态分布和正态分布的抽样统计结果和对应的概率密度函数对比的结果，可见效果不错。

我们可以如图3.5那样绘制出用 AR 法抽取半正态分布的接受和拒绝区域，它们分别是 $u \leq g(-\ln x)$ 和 $u > g(-\ln x)$，也即曲线 $g(-\ln x)$ 的上下部分。而常数 c 就是接受区域的面积。

注意产生目标抽样的办法肯定不止一个。我们上面严格按照 AR 的算法流程，先用逆变换产生了 h 的指数分布，然后再计算 g 并通过独立的均匀分布来判定接受或拒绝。这里产生 h 的时候用了一个 log 运算，产生 g 的时候做了一次指数运算。这两个运算都是代价较高的。特别是 log 运算。注意到 g 本质上也是一个变换后的指数分布，所以如果我们有一个很好的

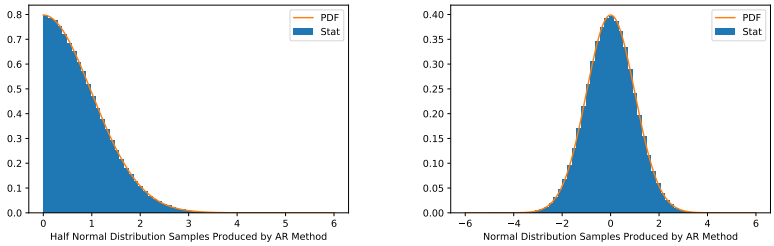

图 3.4: AR 法抽样生成的标准半正态分布（左，算法 3.3）和标准正态分布（右，算法 3.4）的抽样统计结果和各自的 PDF 对比。可见抽样质量不错。

指数分布生成程序，那么可以将具体的算法调整为各自独立地产生两个指数分布，然后对 g 产生的分布做一个变换，再执行接受拒绝判断。具体算法见3.5。

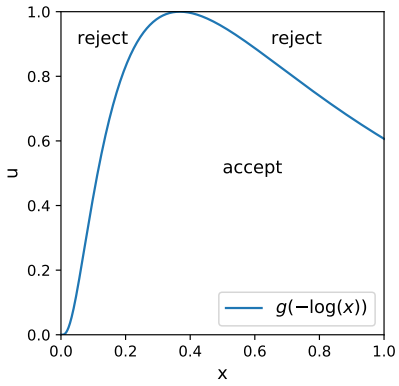

图 3.5: AR 法抽样生成的标准半正态分布的接受 (accept) 和拒绝 (reject) 区域

算法 3.5 AR 法抽取标准半正态分布（基于快速指数分布生成器）

输入 N 实际抽样总数
输出 k 实际接受样本总数
X 输出服从标准半正态分布的样本（前 k 个有效）

```
def sample_half_normal_v2(N):
# N, 实际采样数
    k = 0 # 实际接受总数
    X = stats.expon.rvs(size=N)
    G = stats.expon.rvs(size=N) # 产生两个独立的BETA(1)
    for i in range(N):
        if (X[i] - 1)**2/2 <= G[i]: # 等价于在G的条件下接受
            X[k] = X[i] # 将采样记下来
            k = k + 1
    return k, X
```

3.4 挤压算法

我们已经注意到在抽取半正态分布时，每个样本平均需要执行 $2\sqrt{\frac{2e}{\pi}} \approx 2.631$ 次函数求值，一次指数运算和一次对数运算，还得丢弃一部分。如果我们能稍微减少一些这些基本函数运算，也可以显著降低计算代价。Marsaglia 于 1977 年 [17] 提出一个非常简单但却是有效的思想，挤压算法（squeeze）。在 AR 法中，我们主要的计算量来自要和 $g(z)$ 比大小，但 $g(z)$ 本身的计算是高代价的。如果我们能够构建关系

$$g_L(z) \le g(z) \le g_U(z), z \in [a,b], \tag{3.32}$$

这里区间 $[a,b]$ 是我们的抽样范围。而且 $g_L(z)$ 和 $g_U(z)$ 都是便于计算的函数，那么对是否小于 $g(z)$ 的判定就可以先用这两个界函数过滤一下。如果这两个界充分接近 $g(z)$，那么有很大几率，我们根本不需要再计算 $g(z)$ 了。也即可以将 AR 法的基本步骤改为：

1. 产生服从分布 $h(z)$ 的随机变量 Z；
2. 产生服从 $U(0,1)$ 的随机变量 U；
3. 若 $U \le g_L(Z)$，返回 Z；
4. 否则，若 $U \le g_U(Z)$ 并且 $U \le g(Z)$，返回 Z，否则放弃，返回第 1 步。

事实上，

$$P(\text{计算}g(Z)) = P(g_L(Z) < U < g_U(Z)). \tag{3.33}$$

我们用 S 表示 $U \le g(Z)$ 的事件，分别用 S_L 和 S_U 表示 $U \le g_L(Z)$ 和 $U \le g_U(Z)$ 的事件，则

$$P(S_L) = \int_a^b \max\{0, g_L(z)\}h(z)dz, \tag{3.34}$$

$$P(S_U) = \int_a^b \min\{1, g_U(z)\}h(z)dz, \tag{3.35}$$

再由3.33，有

$$P(\text{计算}g(Z)) = P(S_U) - P(S_L). \tag{3.36}$$

即在一次抽样中，要计算 g 的概率就是这么多。我们还可以令 N 为重复一次成功抽取中计算 $g(Z)$ 的次数，则

$$E[N] = P(\bar{S}_L S_U | \bar{S}) \left(\frac{1}{P(S)} - 1 \right) + P(\bar{S}_L S_U | S). \tag{3.37}$$

这里 $\bar{S}_L$ 和 $\bar{S}$ 表示 $U > g_L(Z)$ 和 $U > g(Z)$，即补事件。由条件概率公式，

$$P(\bar{S}_L S_U | \bar{S}) = \frac{P(S_U) - P(S)}{1 - P(S)}, \tag{3.38}$$

和

$$P(\bar{S}_L S_U | S) = \frac{P(S) - P(S_L)}{P(S)}, \tag{3.39}$$

有

$$E[N] = \frac{P(S_U) - P(S_L)}{P(S)}. \tag{3.40}$$

其中，由定义，

$$P(S) = \int_a^b g(z)h(z)dz = \frac{1}{c}. \tag{3.41}$$

用这个公式可以估计一下大概要实际抽样多少次。

经常遇到的指数和对数函数估计式如下，对指数函数，

$$1 - z \le e^{-z} \le 1 - z + \frac{z^2}{2}, z \ge 0; \tag{3.42}$$

对数函数

$$\frac{z-1}{z} \le \ln z \le -1 + z, z > 0. \tag{3.43}$$

例 3.4. 对半正态抽样采用挤压法 在算法3.3中，我们需要计算

$$g(z) = e^{-\frac{(z-1)^2}{2}},$$

现令

$$g_L(z) = 1 - \frac{(z-1)^2}{2}, \tag{3.44}$$

以及

$$g_U(z) = 1 - \frac{(z-1)^2}{2} + \frac{(z-1)^4}{8} = g_L(z) + \frac{(z-1)^4}{8}. \tag{3.45}$$

则

$$P(S_L) = \frac{1}{2} + (1+\sqrt{2})e^{-(1+\sqrt{2})} \approx 0.7159, \tag{3.46}$$

$$P(S_U) = \frac{13}{8} - 16e^{-3} \approx 0.8284, \tag{3.47}$$

以及

$$P(\text{计算}g(Z) = P(S_U) - P(S_L) \approx 0.1125, \tag{3.48}$$

同时平均每生成一个成功采样需要计算 $g(z)$ 的次数为

$$E[N] = \frac{P(S_U) - P(S_L)}{P(S)} = (P(S_U) - P(S_L)) \times c \approx 0.1480. \tag{3.49}$$

于是每成功生成一个采样需要计算 $g(Z)$ 的次数从 $c \approx 1.3155$ 下降到了 0.1480，效果还是很显著的。从图3.6也可以看到需要计算 $g(Z)$ 的范围大大减少。

可以想象，高效率随机抽样这个问题既是计算机模拟的核心底层操作，又是一个没有止境的操作。同时，和随机数生成一样，它还涉及到很多具体的理论和技术细节。可以预见，随着计算机模拟的重要性不断提升，这些基础核心问题也会日益受到重视。我们到这里只介绍了最基本的算法思想，更多的内容，请参阅 [9] 的第 2 4 章。同时要注意这方面不断有新文献产生。

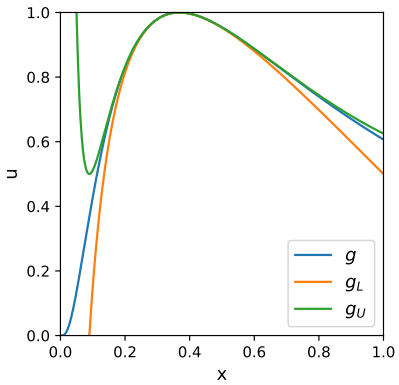

图 3.6: 用 AR 法抽取半正态分布，挤压后在 g_L 和 g_U 之间的区域才实际需要计算 $g(Z)$。

3.5 正态分布抽取

在本章最后，我们单独介绍一个特别重要的分布的抽样方法——正态分布。服从 $N(\mu, \sigma^2)$ 的正态分布相应的 PDF 为

$$f(z) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{\frac{-(z-\mu)^2}{2\sigma^2}}, \sigma^2 > 0, z \in \mathbb{R}. \tag{3.50}$$

关于正态分布的一些性质，大家应该是熟知的：

1. 若 U 服从正态分布 $N(0,1)$，则 $Z = \mu + \sigma U$ 服从 $N(\mu, \sigma^2)$ 分布；
2. 若 $Z_1, Z_2, \cdots, Z_n$ 是分别服从 $N(\mu_1, \sigma_1^2), N(\mu_2, \sigma_2^2), \cdots, N(\mu_n, \sigma_n^2)$ 的独立随机变量，则 $Z = Z_1 + Z_2 + \cdots + Z_n$ 服从分布 $N(\mu_1 + \mu_2 + \cdots + \mu_n, \sigma_1^2 + \sigma_2^2 + \cdots + \sigma_n^2)$.

根据以上的性质，我们知道我们只需要抽样生成 $N(0,1)$ 就足够了。我们之前已经介绍了如何由 AR 法产生半正态分布，进而抽样改造成标准正态分布。下面我们再介绍两种简单的抽取方法。假设

$$Z_n = \sqrt{12/n}(X_1 + X_2 + \cdots + X_n), \tag{3.51}$$

其中 X_i 是独立的服从 $U(-\frac{1}{2}, \frac{1}{2})$ 的随机变量，那么显然有对 $i = 1, 2, \cdots, n$, $E[X_i] = 0$, 且

$$D[X_i] = \int_{-\frac{1}{2}}^{\frac{1}{2}} x^2 dx = \frac{1}{12}.$$

也即 Z_n 的头两阶矩和 $N(0,1)$ 一致。当 $n \to \infty$ 时，$Z_n \sim N(0,1)$. 如果我们取 $n = 12$，那么甚至连根号都不用计算了，但是这时我们发现 Z_{12} 实际上分布在 $(-6, 6)$ 区间而不是 $\mathbb{R}$. 这个方法只有在条件简陋的情况下可以用一下。

第二个方法可以总结成一个定理，由 Box and Muller 在 1958 年 [4] 提出：

定理 3.3. 令随机变量 U 和 V 分别服从 $U(0,1)$ 和 $E(1)$（表示参数为 *1* 的指数分布），那么

$$X = \sqrt{2V}\cos 2\pi U \tag{3.52}$$

和

$$Y = \sqrt{2V}\sin 2\pi U \tag{3.53}$$

是服从 $N(0,1)$ 的独立随机变量。

证明： 注意到

$$f_U = 1, f_V = e^{-v}.$$

再令

$$x = \sqrt{2v}\cos 2\pi u, y = \sqrt{2v}\sin 2\pi u,$$

则有

$$2v = x^2 + y^2$$

和

$$\tan 2\pi u = \frac{y}{x}.$$

因此 X 和 Y 的联合分布函数为

$$f_{X,Y}(x,y) = f_{U,V}(u(x,y), v(x,y))\left|\frac{\partial u}{\partial x}\frac{\partial v}{\partial y} - \frac{\partial u}{\partial y}\frac{\partial v}{\partial x}\right| = \frac{1}{2\pi}e^{-\frac{x^2+y^2}{2}}. \tag{3.54}$$

则 f_X 和 f_Y 各自的边界分为标准正态分布。证毕。

而当前已知生成正态分布最快的方法是由 Marsaglia 在 1964 年 [15] 提出的 rectangle-wedge-tail 方法，并在 1972 年由 Ahrens 和 Dieter 修改成 RT 算法 [7]。但是他们的方法都需要准备大量的预制表格，这一系方法实现比较繁琐。在不用查表的情况下，生成正态分布最快的方法，它是由 Ahrens 和 Dieter 在 1988 年提出的 NA 算法 [2]。有兴趣的同学可自行参阅相应文献。

第四章　Monte Carlo 法：估算体积和计数

这一章我们开始介绍 Monte Carlo 法的基础理论。从如何估算一个多维欧氏空间中的有界区域的体积 (Volumn) 开始。更一般的问题则是在相同区域中估算积分。当区域或目标函数复杂时，Monte Carlo 法具有经典数值方法所无法取代的优势。而计数估计 (Evaluating count) 则是另一个非常重要的问题。它的一个来源是当我们有时需要估计一个元素数量为 m 的集合中具有指定性质的子集的个数。但这种估算的经典算法往往是关于 m 指数增长的。也即我们无法精确地列举全部子集，然后进行计数。Monte Carlo 法在这种情形下，也具有优势。

尽管体积估算和计数估计一个是连续，一个是离散。但实际上这是一类基础问题的不同方面。它们实际上的理论和技巧是相通的。甚至我们有时直接将计数包含进体积的概念之中以简化叙述。

4.1　体积估计

令 $\mathscr{R}$ 是一个包含在 m 维超立方体

$$\mathscr{J}^m = [0,1]^m = [0,1] \times \cdots \times [0,1] \tag{4.1}$$

中的 m 维有界区域，且其体积 $\lambda(\mathscr{R})$ 未知。而对于一般的有界区域，我们假设可以通过适当的变换将其映射进 $\mathscr{J}^m$。如果我们能够产生 m 维点列

$$\mathscr{K}_{m,n} = \left\{ \vec{x}^{(j)} = (x_1^{(j)}, x_2^{(j)}, \cdots, x_m^{(j)}) \in \mathscr{J}^m, j = 1, \cdots, n \right\}, \tag{4.2}$$

则 $\lambda(\mathscr{R})$ 可以用 $\bar{\lambda}_n(\mathscr{R})$ 来逼近。伪代码如算法4.1。

算法 4.1 估算体积

目标 估算 $\lambda(\mathscr{R})$.
输入 包含在 $\mathscr{J}^m$ 中的区域 $\mathscr{R}$, 以及 n 个计值点.
输出 $\bar{\lambda}_n(\mathscr{R})$.

1. $j=1$, $S=0$.
2. While $j \le n$:
   (a) 从 $\mathscr{K}_{m,n}$ 中产生 $\vec{x}^{(j)}$.
   (b) $\phi(\vec{x}^{(j)})=0$.
   (c) If $\vec{x}^{(j)} \in \mathscr{R}, \phi(\vec{x}^{(j)})=1$.
   (d) $S=S+\phi(\vec{x}^{(j)})$.
   (e) j = j + 1.
3. 计算 $\bar{\lambda}_n(\mathscr{R})=S/n$.

这里具体的 $\bar{\lambda}(\mathscr{R})$ 的数值精度依赖 $\mathscr{R}$ 和 $\mathscr{K}_{m,n}$ 的性质。比如如果取均匀划分的网格点，则有

$$\mathscr{K}_{m,n}=\left\{\vec{x}=(x_1,\cdots,x_m): x_i=\frac{z_i+\frac{1}{2}}{k}, z_i=0,\cdots,k-1, i=1,\cdots,m, n=k^m\right\}. \tag{4.3}$$

这个公式的含义是格点在第 i 维的坐标 x_i 可以取 $z_i=0,\cdots,k-1$ 中的任何一个，可以重复取，这样的组合一共有 $n=k^m$ 个，每一组都代表了一个 m 维超单位正方体 $\mathscr{J}^m$ 中的均匀格点。比如我们不难画出 2 维的特例（见图4.1）。可以看到 k 代表了在一个维度上的均匀格点个数。每一个格点，均是一个体积为 $1/k^m=1/n$ 的小超正方体的中心。因此落在 $\mathscr{R}$ 内的格点个数 S 可以估算出 $\mathscr{R}$ 的体积

$$\lambda(\mathscr{R}) \approx \bar{\lambda}_n(\mathscr{R})=\frac{S}{n}.$$

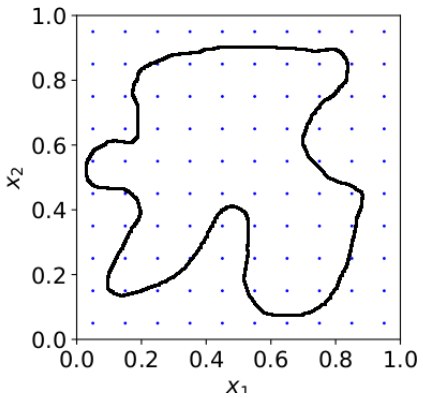

图 4.1: 二维单位正方体内的均匀网格，$k=10$。中间不规则区域表示 $\mathscr{R}$。

如果我们用 $s(\mathscr{R})$ 表示二维区域 $\mathscr{R}$ 的边长，那么通过观察不难看出绝

对误差有上界估计

$$\left|\bar{\lambda}_n(\mathscr{R}) - \lambda(\mathscr{R})\right| \leq \frac{s(\mathscr{R})}{k}. \tag{4.4}$$

事实上，包含了不确定的绝对误差部分最大不会超过长度为 $s(\mathscr{R})$，宽度为 $1/k$ 的条状地带面积。而更一般的，如果用 $s(\mathscr{R})$ 表示一个 m 维区域 $\mathscr{R}$ 的表面积，则对应的绝对误差界也是 (4.4)。由 $n = k^m$，有

$$\left|\bar{\lambda}_n(\mathscr{R}) - \lambda(\mathscr{R})\right| \leq \frac{s(\mathscr{R})}{n^{\frac{1}{m}}}, \tag{4.5}$$

也即误差收敛率随 m 上升而下降。或者说，如果要保证误差小于 $\varepsilon \in (0,1)$，则需要

$$n(\varepsilon) = \left\lceil \left(\frac{s(\mathscr{R})}{\varepsilon} \right)^m \right\rceil$$

个点才能保证。

而 Monte Carlo 方法只改变其中一步，即将序列 $\{\vec{x}^{(1)}, \cdots, \vec{x}^{(n)}\}$ 从一个确定的序列，改成一系列在 $\mathscr{J}^m$ 中均匀分布的独立随机变量

$$\vec{X}^{(j)} = \{X_1^{(j)}, \cdots, X_m^{(j)}\}, j = 1, \cdots, n.$$

也即，X_i 的 PDF 均为 $U(0,1)$. 做相应修改后的算法4.1就是 Monte Carlo 算法，简称 MC，或者，也称作标准 Monte Carlo 采样 (standard Monte Carlo sampling)。我们用算法4.2表示。

算法 4.2 MC

目标 估算 $\lambda(\mathscr{R})$.
输入 包含在 $\mathscr{J}^m$ 中的区域 $\mathscr{R}$, 样本尺寸 n, 置信水平 $1-\delta$.
输出 估计值 $\bar{\lambda}_n(\mathscr{R})$, 方差 $\mathrm{var}\bar{\lambda}_n(\mathscr{R})$ 以及置信区间.

1. $j = 1$, $S = 0$.
2. While $j \leq n$:
   (a) 生成 $\vec{X}^{(j)}$.
   (b) $\phi(\vec{X}^{(j)}) = 0$.
   (c) If $\vec{X}^{(j)} \in \mathscr{R}, \phi(\vec{X}^{(j)}) = 1$.
   (d) $S = S + \phi(\vec{X}^{(j)})$.
   (e) j = j + 1.
3. 做统计分析:
   (a) 计算 $\bar{\lambda}_n(\mathscr{R}) = S/n$ 作为 $\lambda(\mathscr{R})$ 的点估计.
   (b) 计算 $V[\bar{\lambda}_n(\mathscr{R})] = (S/n)(1-S/n)/(n-1)$ 作为 var $\bar{\lambda}_n(\mathscr{R})$ 的点估计.
   (c) 根据置信水平计算百分之 $100 \times (1-\delta)$ 的置信区间.

注意到 $\vec{X}^{(1)}, \cdots \vec{X}^{(n)}$ 均为独立随机变量且 PDF 为

$$f(\vec{x}) = \begin{cases} 1, & 0 \leq x_i \leq 1, i = 1, \cdots, m, \\ 0, & \text{其他}. \end{cases} \tag{4.6}$$

因此 $\phi(\vec{X}^{(1)}),\cdots,\phi(\vec{X}^{(n)})$ 为独立的 Bernoulli 分布随机变量，满足

$$P\{\phi(\vec{X}^{(j)})=1\}=\int_{\mathscr{R}} f(\vec{x})dx=\lambda(\mathscr{R}) \tag{4.7}$$

且

$$P\{\phi(\vec{X}^{(j)})=0\}=\int_{\mathscr{I}^m\setminus\mathscr{R}} f(\vec{x})dx=1-\lambda(\mathscr{R}),j=1,\cdots,n. \tag{4.8}$$

所以，$S=\phi(\vec{X}^{(1)})+\cdots+\phi(\vec{X}^{(n)})$ 服从二项分布

$$P\{S=i\}=f_i(n,\lambda)=\binom{n}{i}\lambda^i(1-\lambda)^{n-i},\lambda=\lambda(\mathscr{R}),i=0,\cdots,n. \tag{4.9}$$

二项分布的期望和方差分别是 $E[S]=n\lambda$ 和 $\text{var } S=n\lambda(1-\lambda)$. 于是 $\bar{\lambda}_n=\bar{\lambda}_n(\mathscr{R})$ 可以看作是 λ 的一个无偏估计，且有

$$\text{var }\bar{\lambda}_n=\frac{1}{n^2}\text{var } S=\lambda(1-\lambda)/n. \tag{4.10}$$

注意其实 λ 是未知量，因此我们并不能用公式（4.10）来估算误差。

4.2 误差和样本大小

首先我们要明确，对 Monte Carlo 法而言，收敛指的是依概率 1 收敛，也即

$$P\left\{\lim_{n\to\infty}\bar{\lambda}_n=\lambda\right\}=1. \tag{4.11}$$

因此，所谓误差也是点估计或区间估计意义上的。比如算法4.2中的 $V[\bar{\lambda}_n(\mathscr{R})]$ 就是 $\text{var }\bar{\lambda}_n$ 的一个无偏估计，因为

$$\begin{aligned} E[(S/n)(1-S/n)] &= \frac{1}{n}E[S]-\frac{1}{n^2}(\text{var } S+E^2[S]) \\ &= \frac{\lambda(1-\lambda)(n-1)}{n}. \end{aligned} \tag{4.12}$$

而 $\sqrt{V[\bar{\lambda}_n]}$ 被称为 $\bar{\lambda}_n$ 的标准误差（standard error）。有时被用作 $\bar{\lambda}_n$ 的统计误差的一个粗略估计，不过由于 $V[\bar{\lambda}_n]$ 本身就是一个估计量，因此要慎重使用。要系统考虑误差的分布，则需要统计学理论，我们陆续会选择一些重要的（或者对我们而言有用的）介绍。

定理 4.1. Chebshev 不等式 令 Z 是一个累积分布为 $F(z)$ 的随机变量，$z\in(-\infty,\infty)$。且 $E[Z]=0$，$\sigma^2=\text{var } Z=E[Z^2]<\infty$。则对 $\beta>0$，有

$$P\left\{\frac{|Z|}{\sigma}\geq\beta\right\}\leq\frac{1}{\beta^2}. \tag{4.13}$$

证明. 对 $\varepsilon > 0$，有

$$\begin{aligned} P\{|Z| \geq \varepsilon\} &= \int_{-\infty}^{-\varepsilon} dF(z) + \int_{\varepsilon}^{\infty} dF(z) \\ &\leq \int_{-\infty}^{-\varepsilon} \frac{z^2}{\varepsilon^2} dF(z) + \int_{\varepsilon}^{\infty} \frac{z^2}{\varepsilon^2} dF(z) \qquad (4.14) \\ &\leq \frac{1}{\varepsilon^2} \int_{-\infty}^{\infty} z^2 dF(z) = \frac{\sigma^2}{\varepsilon^2}. \end{aligned}$$

取 $\beta = \varepsilon/\sigma$ 即得（4.13）。 □

现取 $Z = S/n - \lambda$ 和 $\sigma^2 = \lambda(1-\lambda)/n$，得

$$P\{|\bar{\lambda}_n - \lambda| < \varepsilon\} \geq 1 - \lambda(1-\lambda)/n\varepsilon^2. \qquad (4.15)$$

即

$$\lim_{n\to\infty} P\{|\bar{\lambda}_n - \lambda| \geq \varepsilon\} = 0. \qquad (4.16)$$

上式称为概率收敛（convergence in probability），由依概率 1 收敛可以导出概率收敛，但反过来不行。它比依概率 1 收敛（convergernce with probability 1，缩写 w.p.1）要弱。但这里实际给出了一个收敛性和样本大小 n 之间的关系。不过这里再大的 n 都不能确保误差小于 ε。为了正确的体现和评估随机性，我们需要引入置信水平（confidence level）$1-\delta$，$0 < \delta < 1$，由 Chebyshev 不等式，当

$$n \geq n_C(\varepsilon, \delta, \lambda) = \lceil \lambda(1-\lambda)/\delta\varepsilon^2 \rceil \qquad (4.17)$$

时，误差满足

$$P\{|\bar{\lambda}_n - \lambda| < \varepsilon\} \geq 1 - \delta. \qquad (4.18)$$

我们称其为绝对误差准则（absolute error criterion）。注意到 $\lambda(1-\lambda) \leq 1/4$，故有最坏情形样本数（the worst-case sample size）

$$n_C(\varepsilon, \delta) = \lceil \frac{1}{4\delta\varepsilon^2} \rceil, \qquad (4.19)$$

对所有 $\lambda \in [0,1]$。

公式（4.19）表达了 Monte Carlo 法最重要的性质，$n_C(\varepsilon, \delta, \lambda)$ 和维数 m 无关。而在实际计算中，由于随机投点本身是一个 $O(m)$ 的操作，且在每个维度上我们要通过计算来判断 $\vec{X}$ 是否属于 $\mathscr{R}$，这些操作加起来不超过 $O(m^\beta)$，$\beta \geq 1$。故实际在程序中估算绝对误差界的代价是

$$O([m^\beta \frac{\lambda(1-\lambda)}{\delta\varepsilon^2}]) = O(\frac{m^\beta}{4\delta\varepsilon^2}).$$

仍然是一个关于 m 的多项式时间算法，而相应精确逼近算法关于 m 是指数时间的。

尽管公式 (4.19) 给出了一个样本数的估计，但这个估计明显是过宽的。实际上的样本数需求要远小于这个估计。为此我们考虑根据中心极限定理，当 $n \to \infty$ 时，特征量

$$\frac{S - n\lambda}{[n\lambda(1-\lambda)]^{\frac{1}{2}}}$$

是服从标准正态分布

$$\Phi(z) = (2\pi)^{-\frac{1}{2}} \int_{-\infty}^{z} e^{-\frac{y^2}{2}} dy, -\infty < z < \infty, \tag{4.20}$$

也即 $N(0,1)$ 分布。令

$$n_N(\varepsilon, \delta, \lambda) = \left\lceil \lambda(1-\lambda) \left[\frac{\Phi^{-1}(1-\frac{\delta}{2})}{\varepsilon} \right]^2 \right\rceil, \tag{4.21}$$

其中

$$\Phi^{-1}(\theta) = \inf \left[z : (2\pi)^{-\frac{1}{2}} \int_{-\infty}^{z} e^{-\frac{y^2}{2}} dy = \theta, 0 < \theta < 1 \right]. \tag{4.22}$$

于是当 $\varepsilon \to 0$, 公式 (4.21) 给出了在参数 (ε, δ) 下，绝对误差满足

$$\lim_{\varepsilon \to 0} P \left\{ \frac{\left| \frac{S}{n_N(\varepsilon,\delta,\lambda)} - \lambda \right|}{\varepsilon} \le 1 \right\} = 1 - \delta \tag{4.23}$$

的样本数。而对应的最坏情形样本数是

$$n_N(\varepsilon, \delta) = \left\lceil \left[\frac{\Phi^{-1}(1-\frac{\delta}{2})}{2\varepsilon} \right]^2 \right\rceil. \tag{4.24}$$

我们可以比较在 $\delta = 0.001, 0.01$ 和 0.05 下，

$$\frac{n_C}{n_N} = \frac{1}{\delta \left[\Phi^{-1}(1-\frac{\delta}{2})\right]^2}$$

的值分别是 92.36, 15.07 和 5.21。可见这个比例还是很大的。不过这么估计也有比较严重的问题。因为我们实际上就把 S/n 当作正态分布计算了，而这个只是一个概率极限情形。因此，当 n 不够大（注意这里逻辑上有矛盾），或者 ε 太小时，这个结果都有很大的误差。一个弥补的办法是故意高估 n，比如有人建议总是取两倍的 n_N。而更加靠谱的估计，来自 Hoeffding 于 1963 年的工作 [11]。（以下部分只要了解结论，不作要求。）

定理 4.2. Hoeffding 1963 令 $Z_1, \cdots, Z_n$ 是独立随机变量且

$$\mu_i = E[Z_i] \in (0,1), P\{0 \le Z_i \le 1\} = 1, i = 1, \cdots, n.$$

令 $\mu = (\mu_1 + \cdots + \mu_n)/n$。则对 $0 < \varepsilon < 1 - \mu$，和 $\bar{Z}_n = (Z_1 + \cdots + Z_n)/n$，

$$P\{\bar{Z}_n - \mu \ge \varepsilon\} \le e^{nw(\varepsilon,\mu)} \tag{4.25}$$

$$\le e^{-\mu g(\mu)\varepsilon^2} \tag{4.26}$$

$$\le e^{-2n\varepsilon^2} \tag{4.27}$$

其中

$$w(\varepsilon,\mu)=(\mu+\varepsilon)\log\left(\frac{\mu}{\mu+\varepsilon}\right)+(1-\mu-\varepsilon)\log\left(\frac{1-\mu}{1-\mu-\varepsilon}\right), \tag{4.28}$$

以及

$$g(\mu)=\begin{cases}\dfrac{1}{1-2\mu}\log\left(\dfrac{1-\mu}{\mu}\right), & 若0<\mu<\frac{1}{2}\\[2ex] \dfrac{1}{2\mu(1-\mu)}, & 若\frac{1}{2}\le\mu<1.\end{cases} \tag{4.29}$$

推论 4.1. 对公式（4.28）中定义的 $w(\varepsilon,\mu)$，

$$\lim_{\varepsilon\to0}\varepsilon^{-2}w(\varepsilon,\mu)=-\frac{1}{2\mu(1-\mu)} \tag{4.30}$$

推论 4.2. 函数 $w(\varepsilon,y)$，$0<\varepsilon<1-y$，$0<y<1$ 关于 y 是凹的。故有最大值 $y^*\in[\max(0,\frac{1}{2}-\varepsilon),\frac{1}{2}]$。

推论 4.3. 函数 $w((y+v)\varepsilon,y)$，$0<\varepsilon\le\frac{1-y}{y+v}$，$0<y<1$，$v\ge0$；和函数 $w((y+v)\varepsilon,1-y)$，$0<\varepsilon\le\frac{y}{y+v}$，$0<y<1$，$v>0$，关于 y 都是凹的。

推论 4.4. 对公式（4.28）中定义的 $w(\varepsilon,y)$，有 $w(\varepsilon,y)\ge w(\varepsilon,1-y)$，$\varepsilon\le y\le\frac{1}{2}$。

以上证明略。自行参考 [9] 第 24 页、附录及 Hoeffding 原文。该定理可导出一系列有用的结论。比如样本量的估计：

$$n_H(\varepsilon,\delta)=\left\lceil\frac{2\log(\frac{2}{\delta})}{4\varepsilon^2}\right\rceil. \tag{4.31}$$

以及对 $\varepsilon<\min\{\mu,1-\mu\}$（这里 $\mu=\lambda$），有

$$n_H(\varepsilon,\delta)=\left\lceil\frac{\log(\frac{\delta}{2})}{\sup_{\varepsilon\le v\le1-\varepsilon}w(\varepsilon,v)}\right\rceil. \tag{4.32}$$

这里公式（4.32）分母中的最大值可以用数值方法求解。这两个估计一般认为是比 n_N 更精确的估计。

4.3 置信区间

之前我们给出了面积的一个点估计 $\bar{\lambda}_n$，以及讨论在置信度 δ 下的一个关于偏差 ε 的样本数量估计。似乎所有关于计算模拟的信息已经具备了，为何还要继续讨论区间估计呢？或者说区间估计难道不就是

$$P\{|\bar{\lambda}_n-\lambda|\le\varepsilon\}=1-\delta$$

吗？这个结果有两个比较可疑的地方，一个是这个区间居然是对称的；另一个是这个估计居然和 λ 无关。但是我们考虑一下如果一个实际的 λ 比较接近 0，那么在随机投点时，落入 $\mathscr{R}$ 内的点数本身就相对有限，因此其频率 $\bar{\lambda}_n$ 往负方向可能的偏差就很有限。比如 $\lambda = 0.005$，那么负方向的偏差不可能大于 0.005。但正方向的偏差却仍然服从方差分布的规律，可以大于 0.005。于是我们应该能观察到频率 $\bar{\lambda}_n$ 的分布往正方向偏的现象。图 (4.2) 给出了 $n = 1000$ 时，对一个面积为 0.005 的圆进行 Monte Carlo 求面积时的频率分布的统计，总共做了 50000 组实验。我们可以看到，实际的频率分布确实是往正方向偏移的。因为负方向根本没有偏移的空间了。

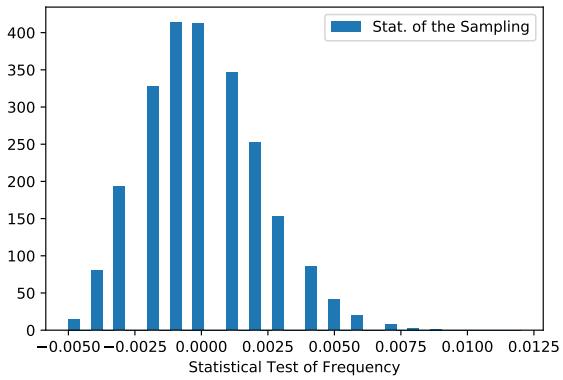

图 4.2: 对一个面积为 0.005 的圆进行 Monte Carlo 求面积时频率分布的统计，$n = 1000$，总共做了 50000 组实验。

以上这些观察提示我们，实际频率的分布和频率本身有关，同时也应该有一定的偏移。这就是区间估计的出发点。我们希望将频数 S 也纳入统计估计，给出一个置信区间的上下界范围。也即寻找 $I_1 = I_1(S, n, \delta)$ 和 $I_2 = I_2(S, n, \delta)$，使得

$$P\{I_1, I_2, \delta\} = P\{I_1 < \lambda < I_2\} \geq 1 - \delta.$$

这里 $P\{I_1, I_2, \delta\}$ 称为范围概率(coverage probability)或者可靠系数(confidence coefficent)。而区间 $[I_1, I_2]$ 称为置信区间 (confidence interval)。置信区间是比给出一个单点估计 $\bar{\lambda}_n$ 更加可靠的统计估计。关于二项分布的区间估计的理论和方法很多。我们这里给出三种主要方法。

记二项分布的累计分布函数为

$$F_i(n, \mu) = \sum_{j=0}^{i} \binom{n}{j} \mu^j (1-\mu)^{n-j}, 0 < \mu < 1, i = 0, 1, \cdots, n. \tag{4.33}$$

因此对 $0 < \alpha_1 < \alpha_2 < 1$，存在唯一的 $\theta_1(s, n, \alpha_1)$ 和 $\theta_2(s, n, \alpha_2)$ 分别满足

$$1 - F_{s-1}(n, \theta) = \alpha_1 \tag{4.34}$$

和

$$F_s(n, \theta) = 1 - \alpha_2. \tag{4.35}$$

特别地，对 $s=1$，我们分别令 $\theta_1=0$ 和 $\theta_2=1$。注意到 $F_s(n,\theta)$ 关于 θ 递减和关于 s 递增，所以有

$$0\le\theta_1(s,n,\alpha_1)<\theta_2(s,n,\alpha_2)\le 1.$$

因此，若对 n 次独立的 Bernolli 试验，恰好成功 S 次的概率是 μ，则对满足

$$1+\alpha_1-\alpha_2=\delta$$

的 α_1 和 α_2，区间 $[\theta_1(S,n,\alpha_1),\theta_2(S,n,\alpha_2)]$ 就是 μ 关于置信水平 δ 的范围区间，也即

$$P\{\theta_1(S,n,\alpha_1)<\mu<\theta_2(S,n,\alpha_2)\}\ge 1-\delta.$$

满足上面条件的 α_1 和 α_2 很多，我们这里采用的是 $\alpha_1=\frac{\delta}{2}$ 和 $\alpha_2=1-\frac{\delta}{2}$. 也即我们最终取

$$I_1(S,n,\delta)=\theta_1(S,n,\frac{\delta}{2}),I_2(S,n,\delta)=\theta_2(S,n,1-\frac{\delta}{2}).$$

为了计算 θ_1 和 θ_2，我们引入不完全 Beta 函数 [1]：

$$1-F_{j-1}(n,\theta)=H_\theta(j,n-j+1)=\frac{1}{B(j,n-j+1)}\int_0^\theta z^{j-1}(1-z)^{n-j}dz,0\le\theta\le 1. \tag{4.36}$$

其中 $B(\cdot,\cdot)$ 表示 Beta 函数。而 $H_\theta(j,n-j+1)$ 实际上就是参数为 j 和 $n-j+1$ 的 Beta 分布的累积分布函数。而所求

$$\theta_1(S,n,\frac{\delta}{2}),\theta_2(S,n,1-\frac{\delta}{2})$$

就分别是方程

$$H_\theta(S,n-S+1)=\frac{\delta}{2},H_\theta(S+1,n-S)=1-\frac{\delta}{2}$$

的根。

在具体求根的时候，注意到 H_θ 关于 θ 的单调性，我们可以用二分法来求解。初始含根区间可以选为 $\theta_a=0$，$\theta_b=1$。如果要加速收敛，可以在误差达到一定精度后采用 Newton 迭代。这些方法是经典数值方法，这里不再详述，具体例子可参见讲义第四章附带代码。表 4.1给了一些实际计算的结果做参考。

表 4.1: 置信区间举例，$n=1000,1-\delta=0.99$。其中 $H_1=H_\theta(S,n-S+1,\theta_1)$，$H_2=H_\theta(S+1,n-S,\theta_2)$。

S	λ_n	θ_1	θ_2	H_1	H_2	$P\{\theta_1<\lambda<\theta_2\}$
11	0.0011	0.004334	0.02265	0.005003	0.99501	0.990007

第二种置信区间的求法可以总结成一个定理：

定理 4.3. *令 S 有期望 $n\lambda$ 和方差 $n\lambda(1-\lambda)$，定义*

$$\omega_i(S,n,\beta)=\frac{S+\frac{\beta^2}{2}+\beta(-1)^i\sqrt{\frac{\beta^2}{4}+\frac{S(n-S)}{n}}}{n+\beta^2} \tag{4.37}$$

其中 $\beta>0$，$n\in\mathbb{N}^+$，$0\le S\le n$ 并且 $i=1,2$。则存在 $\beta=\beta(\delta)$ 使得开区间 $(\omega_1(S,n,\beta),\omega_2(S,n,\beta))$ 以大于等于 $1-\delta$ 的概率覆盖了 λ。

证明. 由 Chebshev 不等式，

$$P\left\{\frac{\left|\frac{S}{n}-\lambda\right|}{\sqrt{\lambda(1-\lambda)/n}}<\beta\right\}=P\left\{(\frac{S}{n}-\lambda)^2<\frac{\beta^2\lambda(1-\lambda)}{n}\right\}\ge 1-\frac{1}{\beta^2},\quad(4.38)$$

所以只要取 $\beta=\frac{1}{\sqrt{\delta}}$ 即得：

$$P\left\{\left[\omega_1(S,n\frac{1}{\sqrt{\delta}})-\lambda\right]\left[\omega_2(S,n,\frac{1}{\sqrt{\delta}})-\lambda\right]<0\right\}\ge 1-\delta.\quad(4.39)$$

□

我们知道，Chebyshev 不等式是过宽估计的，所以我们一般实际取

$$\beta=\Phi^{-1}(1-\frac{\delta}{2}),0<\lambda<1,$$

然后令 $I_1(S,n,\delta)=\omega_1(S,n,\beta)$ 和 $I_2(S,n,\delta)=\omega_2(S,n,\beta)$。

不过基于正态分布的 β 估值，同样会导致在 λ 接近 0 或 1 时，置信区间覆盖不住 λ。通过修正

$$\bar{\omega}_i(z,n,\beta)=\omega_i(z+0.5\times(-1)^i,n,\beta),i=1,2.\quad(4.40)$$

一般在 $\lambda<0.3$ 或 $\lambda>0.7$，实际操作中，发现 S 小于 $0.3n$ 或大于 $0.7n$ 时，可以考虑做此修正。具体理论分析参见 [3]。

第三种置信区间估计不再依赖具体的分布，它也可以总结成一个定理。首次发表人就是我们参考书的作者。

定理 4.4. *Fishman 1991* 令 $Z_1,\cdots,Z_n$ 为独立随机变量，且 $\mu=E[Z_i]\in(0,1)$，$P\{0\le Z_i\le 1\}=1$，令

$$\bar{Z}=\frac{Z_1+\cdots+Z_n}{n},$$

对 $0\le z\le 1$ 和 $0<\delta<1$，定义

$$\rho_1(z,n,\delta)=\begin{cases}\{t:0<t\le z\le 1,e^{nw(z-t,t)}=\frac{\delta}{2}\} & z>0\\ 0 & z=0.\end{cases}\quad(4.41)$$

和

$$\rho_2(z,n,\delta)=\begin{cases}\{t:0<t\le z\le 1,e^{nw(t-z,1-t)}=\frac{\delta}{2}\} & z<1\\ 0 & z=1.\end{cases}\quad(4.42)$$

其中 w 的定义如 *(4.28)*。则

$$P\{\rho_1(\bar{Z}_n,n,\delta)<\mu<\rho_2(\bar{Z}_n,n,\delta)\}\ge 1-\delta.$$

具体应用时，取 $I_1(S,n,\delta)=\rho_1(\frac{S}{n},n,\delta)$ 和 $I_2(S,n,\delta)=\rho_2(\frac{S}{n},n,\delta)$。在求解过程中，需要关于 w 的方程，可以用和之前一样的求根技巧。以上三个方法，请大家自行编写代码验证，这里给出一些结果供对比。

我们这里介绍的只是误差估计的非常基础的部分，更深入的理论和分析需要更专业的统计学知识，大家可参阅相关资料。参考书 2.4-2.5 章也提供了更多的讨论。

表 4.2: 置信区间举例，$n = 1000$，$1 - \delta = 0.99$。

S	$\bar{\lambda}_n$	ω_1	ω_2	$\bar{\omega}_1$	$\bar{\omega}_2$	ρ_1	ρ_2
11	0.0011	0.005163	0.02328	0.004844	0.02396	0.003421	0.02540
323	0.2862	0.3622	0.2857	0.3627	0.2762	0.3723	0.02540

4.4 界的估计

我们在这一节集中讨论一下一个在计算机模拟中常见的策略。如果存在 $\mathscr{R}$ 的超矩形下界 $\mathscr{R}_L$ 和上界 $\mathscr{R}_U$（在二维可以想像成一个内接矩形和一个外接矩形），那么我们的随机投点范围可以缩小，从而提高算法效率。这里可令

$$\mathscr{R}_L = \{\vec{x} \in \mathscr{J}^m : 0 \le \alpha_{Li} \le x_i \le \beta_{Li} \le 1, 1 \le i \le m\} \tag{4.43}$$

和

$$\mathscr{R}_U = \{\vec{x} \in \mathscr{J}^m : 0 \le \alpha_{Ui} \le x_i \le \beta_{Ui} \le 1, 1 \le i \le m\} \tag{4.44}$$

并且，$\forall \vec{x} \in \mathscr{J}^m$，如果 $\vec{x} \in \mathscr{R}_L$，则必有 $\vec{x} \in \mathscr{R}$；如果 $\vec{x} \in \mathscr{J}^m \backslash \mathscr{R}_U$，则必有 $\vec{x} \notin \mathscr{R}$. 图4.3给出了一个二维的例子（俺先盗个图）。注意到 $\mathscr{R}_L$ 和 $\mathscr{R}_U$ 的体积分别为

$$\lambda_L = \prod_{i=1}^{m} (\beta_{Li} - \alpha_{Li}) \tag{4.45}$$

和

$$\lambda_U = \prod_{i=1}^{m} (\beta_{Ui} - \alpha_{Ui}) \tag{4.46}$$

这里如果可能，我们会尽可能通过调整 $\mathscr{R}_L$ 和 $\mathscr{R}_U$ 使得 λ_L 越大越好同时 λ_U 越小越好，然后我们可以通过缩小投点的分布范围获得更多的好处。

定理 4.5. 令

$$\phi_L(\vec{x}) = \begin{cases} 1, & \vec{x} \in \mathscr{R}_L \\ 0, & \text{其他.} \end{cases} \tag{4.47}$$

和

$$\phi_U(\vec{x}) = \begin{cases} 1, & \vec{x} \in \mathscr{R}_U \\ 0, & \text{其他.} \end{cases} \tag{4.48}$$

若 $f(\vec{x})$ 表示在 $\mathscr{J}^m$ 均匀分布的概率密度函数，令 $\vec{X}$ 服从的概率密度函数为：

$$f(\vec{x}, \lambda_L, \lambda_U) = \frac{\phi_U(\vec{x}) - \phi_L(\vec{x})}{\lambda_U - \lambda_L} f(\vec{x}), \vec{x} \in \mathscr{J}^m. \tag{4.49}$$

则

$$Z = \lambda_L + (\lambda_U - \lambda_L)\phi(\vec{X}), \tag{4.50}$$

满足

1.

$$E[Z] = \lambda. \tag{4.51}$$

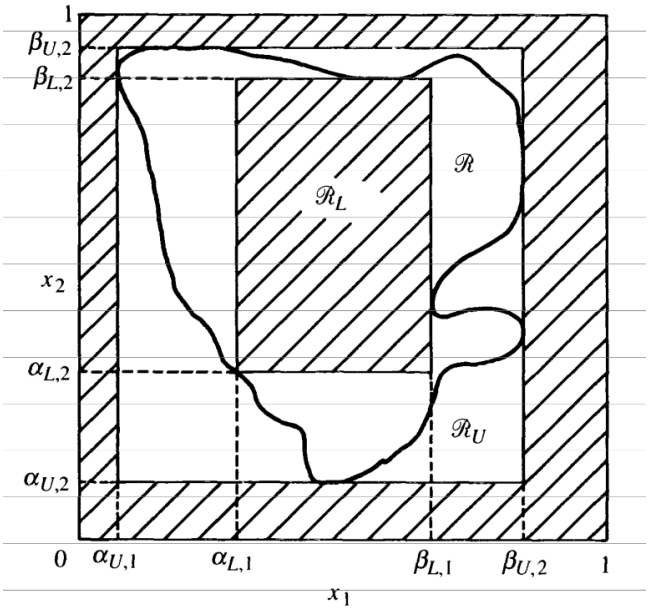

图 4.3: 二维的矩形界：$\mathscr{R}_L \subset \mathscr{R} \subset \mathscr{R}_U \subset \mathscr{J}^2$。

2.

$$\begin{aligned} \mathrm{var}Z &= (\lambda - \lambda_L)(\lambda_U - \lambda) \\ &\leq \frac{(\lambda_U - \lambda_L)^2}{4}. \end{aligned} \tag{4.52}$$

这个定理的证明是简单的。定理中的 $\phi(\vec{x})$ 的取值为

$$\phi(\vec{x}) = \begin{cases} 1, & \vec{x} \in \mathscr{R} \\ 0, & \text{其他}. \end{cases} \tag{4.53}$$

接下去的问题如何产生服从 $f(\vec{x}, \lambda_L, \lambda_U)$ 的随机数？办法基本上有两种，一种是仍然产生在整个 $\mathscr{J}^m$ 均匀分布的点，但是用拒绝接受（AR）策略来过滤出落在 $\mathscr{R}_L$ 和 $\mathscr{R}_U$ 之间的点。第二种是考虑整个 $\mathscr{R}_U \backslash \mathscr{R}_L$ 区域被 α_{Li}，α_{Ui}，β_{Li} 和 β_{Ui} 分成了 m 个区域，每个区域由两个小超矩形块组成。我们可以先从这些块中，按体积为概率选择一个，再在上面均匀投点，这样投点的计算量仍然是 $\theta(m)$ 的。这两种策略的具体实现都参见参考书 2.5 节。其中第 50 页的 Version C 是上面的第二种思路的 m 维一般化，见算法 (4.3)。

算法 4.3 m 维超矩形上下界约束下的直接抽取

目标 对 $\mathscr{R}_L \subset \mathscr{R}_U \subset \mathscr{J}^m$, 直接抽取在 $\mathscr{R}_U \backslash \mathscr{R}_L$ 中均匀分布的随机向量 $\vec{X}$.
输入 包含在 $\mathscr{J}^m$ 中的超矩形区域 $\mathscr{R}_L$ 和 $\mathscr{R}_U$ 的顶点坐标数据 $\alpha_{Lj}, \beta_{Lj}, \alpha_{Uj}, \beta_{Uj}$, 且 $\mathscr{R}_L \subset \mathscr{R}_U$, $j = 1, 2, \cdots, m$.

1. 计算分块的体积（测度），并由此产生落入各分块的累计分布函数 s_j, $j = 1, 2, \cdots, m$.
2. 产生一个服从 $U(0,1)$ 的随机数 Z, 根据 Z 具体位置产生实际抽样分块 $J \in \{1, 2, \cdots, m\}$, $s_J - 1 \leq Z < s_J$, $s_0 = 0$.
3. 计算 $\Delta_1 = \beta_{Uj} - \beta_{Lj}$, $\Delta_2 = \alpha_{Lj} - \alpha_{Uj}$, $j = 1, 2, \cdots, m$.
4. While $j \leq J$:
   (a) 产生服从 $U(\alpha_{Lj}, \beta_{Lj})$ 的随机数 X_j;
   (b) j = j + 1.
5. 产生服从 $U(0,1)$ 的随机数 Y,
   $$X_J = \beta_{LJ} + \lfloor (\Delta_1 + \Delta_2) Y / \Delta_1 \rfloor (\alpha_{UJ} - \beta_{UJ}) + Y(\Delta_1 + \Delta_2).$$
6. j = j + 1.
7. While $j \leq m$:
   (a) 产生服从 $U(\alpha_{Lj}, \beta_{Uj})$ 的随机数 X_j.
   (b) j = j + 1.

在算法（4.3）中，s_j 的生成如下：令 $a_i = \beta_{Li} - \alpha_{Li}$，$b_i = \beta_{Ui} - \alpha_{Ui}$，$1 \leq i \leq m$,

$$s_j = \begin{cases} (b_1 - a_1) \left(\prod_{i=2}^{m} b_i \right) \cdot \dfrac{1}{\lambda_U - \lambda_L}, & j = 1 \\ \left(\prod_{i=1}^{j-1} a_i \right) (b_j - a_j) \left(\prod_{i=j+1}^{m} b_i \right) \cdot \dfrac{1}{\lambda_U - \lambda_L}, & 1 < j < m. \\ (b_m - a_m) \left(\prod_{i=1}^{m-1} a_i \right) \cdot \dfrac{1}{\lambda_U - \lambda_L}, & j = m. \end{cases} \tag{4.54}$$

其中

$$\lambda_L = \prod_{i=1}^{m} a_i, \lambda_U = \prod_{i=1}^{m} b_i.$$

而具体的 MC 算法相应修改为如下算法（4.4）。

算法 4.4 在上下界估计下的 MC 方法

目标 估算 $\lambda(\mathscr{R})$.
输入 包含在 $\mathscr{J}^m$ 中的区域 $\mathscr{R}$, $\lambda(\mathscr{R})$ 的下界估计 λ_L 和上界估计 λ_U, 样本尺寸 n, 置信水平 $1-\delta$.
输出 无偏点估计：$\bar{\lambda}_n(\mathscr{R})$, 方差 $\mathrm{var}\bar{\lambda}_n(\mathscr{R})$ 以及置信区间.

1. $j=1$, $S=0$.
2. While $j \le n$:
   (a) 生成 $\vec{X}^{(j)}$ 服从概率密度函数 $f(\vec{x}, \lambda_L, \lambda_U)$.
   (b) $\phi(\vec{X}^{(j)})=0$.
   (c) If $\vec{X}^{(j)} \in \mathscr{R}, \phi(\vec{X}^{(j)})=1$.
   (d) $S=S+\phi(\vec{X}^{(j)})$.
   (e) j = j + 1.
3. 做统计分析:
   (a) 计算 $\bar{\lambda}_n(\mathscr{R})=\lambda_L+(\lambda_U-\lambda_L)S/n$ 作为 $\lambda(\mathscr{R})$ 的点估计.
   (b) 计算 $V[\bar{\lambda}_n(\mathscr{R})]=(\lambda_U-\lambda_L)^2(S/n)(1-S/n)/(n-1)$ 作为 var $\bar{\lambda}_n(\mathscr{R})$ 的点估计.
   (c) 根据置信水平计算百分之 $100\times(1-\delta)$ 的置信区间：

$$I_i(S,n,\delta)=\lambda_L+(\lambda_U-\lambda_L)\tilde{I}_i, i=1,2.$$

   这里 $\tilde{I}_i$ 是原 MC 中对应的置信区间界.

4.5 网络可靠性

接下去我们讨论一个 MC 方法的应用例子。首先我们需要一点点图论的知识。令 G 表示一个无向网络（undirected network），它的顶点（nodes）编号的集合是 $\mathscr{V}$，而全部边（edges）编号的集合是

$$\mathscr{E}=\{1,2,\cdots,m\}.$$

我们现在考虑这个由顶点和边组成的网络，假设全部的顶点都是可靠的，但边的可靠性却是一个独立的随机事件。也即 $\forall i \in \mathscr{E}$，这条边连通的概率是 q_i，而有 $1-q_i$ 的概率是不连通的。现在用随机变量 y_i 表示一条边是否连通的状态，也即

$$y_i=\begin{cases}1, & \text{边 } i \text{ 是连通的;}\\ 0, & \text{其它.}\end{cases}$$

则 $\vec{y}=(y_1,y_2,\cdots,y_m)$ 表示整个网络 m 条边的状态。用 $\mathscr{Y}$ 表示全部 $\vec{y}$ 可能状态的集合。

$\forall s,t \in \mathscr{V}$，令

$$\phi(\vec{y})=\begin{cases}1, & \text{若 } s \text{ 和 } t \text{ 在 } \vec{y} \text{ 状态下是连通的.}\\ 0, & \text{其它.}\end{cases}$$

再令

$$P(\vec{y},\vec{q})=\prod_{i\in\mathscr{E}} q_i^{y_i}(1-q_i)^{1-y_i},$$

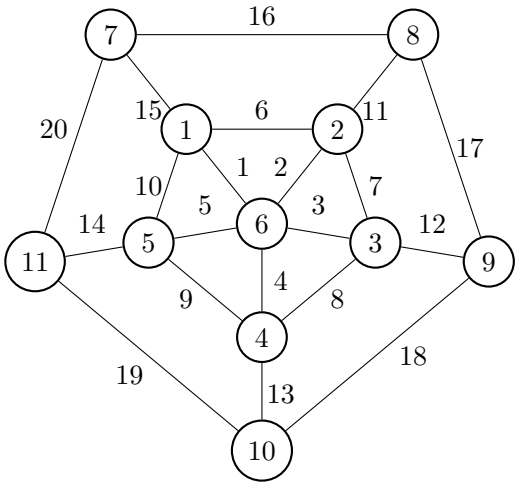

图 4.4: 一个网络的示意图

其中

$$\vec{q} = (q_1, \cdots, q_m),$$

则我们关心的是 s 和 t 之间连通的概率，也即

$$g(\vec{q}) = \sum_{\vec{y} \in \mathscr{Y}} \phi(\vec{y}) P(\vec{y}, \vec{q}). \tag{4.55}$$

如果 $\vec{q}$ 是已知的，那么 $g(\vec{q})$ 是可计算的（computable）。它应该就严格等于

$$g(\vec{q}) = \sum_{\vec{y} \in \mathscr{Y}_1} P(\vec{y}, \vec{q}), \tag{4.56}$$

其中

$$\mathscr{Y}_1 = \{\vec{y} \in \mathscr{Y} \,|\phi(\vec{y}) = 1\}.$$

然而，如何确定 $\mathscr{Y}_1$ 是一个 NP-Hard 的问题 [25]。在这种情况下，MC 方法也成为一个进行计算模拟的有效手段。

用 Γ 表示网络 G 中点 $s-t$ 的最小切割集（minimal $s-t$ cutsets）全体。一个 $s-t$ 的切割集是一些边的集合，在 G 中去除这些边，就会使 $s-t$ 不再连通。而最小切割集则表示该集合的任何子集都不会切割 $s-t$。于是

$$\begin{aligned} \phi(\vec{y}) &= \prod_{\mathscr{C} \in \Gamma} \left[1 - \prod_{i \in \mathscr{C}} (1 - y_i) \right] \\ &= \prod_{\mathscr{C} \in \Gamma} \left[1 - \min_{i \in \mathscr{C}} (1 - y_i) \right]. \end{aligned}$$

令 $x_i \in [0,1]$，$i \in \mathscr{E}$，则可取

$$y_i = 1 - \lfloor 1 - q_i + x_i \rfloor.$$

于是 $s-t$ 网络可靠性（4.56）等价于计算区域

$$\mathscr{R} = \left\{ (x_1, \cdots, x_m) \in \mathscr{I}^m \left| \min_{\mathscr{C} \in \Gamma} \left[1 - \min_{i \in \mathscr{C}} (\lfloor 1 - q_i + x_i \rfloor) \right] = 1 \right. \right\}.$$

我们可以用 MC 方法求解这个面积。同样地，为了提高采样精度，我们可以来考虑 λ_L 和 λ_U。也就是 $\mathscr{R}$ 的上下界估计，同时这种估计还要能够满足简单采样的要求。

我们用 $\mathscr{P}_1, \mathscr{P}_2, \cdots, \mathscr{P}_I$ 表示网络 G 的全部 I 个 $s-t$ 不相交最短路径集（edge-disjoint paths），即 $\forall i \in \{1, 2, \cdots, I\}$，$\mathscr{P}_i$ 中的边构成 s 到 t 在 G 中的最短路径，且 $\mathscr{P}_i \cap \mathscr{P}_j = \emptyset$，$\forall i \neq j$。类似地，记 $\mathscr{C}_1, \mathscr{C}_2, \cdots, \mathscr{C}_J$ 为 J 个 G 中的 $s-t$ 不相交最小切割集（edge-disjoint minimal $s-t$ cutsets）。令

$$\phi_L(\vec{y}) = \begin{cases} 1, & \text{若 } \vec{y} \text{ 在至少在一个不相交最短路径集上全部连通,} \\ 0, & \text{其他.} \end{cases}$$

（若 $\phi_L(\vec{y}) = 1$ 则 $s-t$ 必然连通。）类似地，令

$$\phi_U(\vec{y}) = \begin{cases} 1, & \text{若 } \vec{y} \text{ 在每个不相交最小切割集中都至少有一条边连通,} \\ 0, & \text{其他.} \end{cases}$$

（若 $\phi_U(\vec{y}) = 0$ 则说明至少有一个不相交最小切割集完成了切割，因此 $s-t$ 必然分离。）等价地，

$$\begin{aligned} \phi_L(\vec{y}) &= 1 - \prod_{j=1}^{I} \left(1 - \prod_{i \in \mathscr{P}_j} y_i \right), \\ \phi_U(\vec{y}) &= \prod_{j=1}^{J} \left(1 - \prod_{i \in \mathscr{C}_j} (1 - y_i) \right). \end{aligned} \tag{4.57}$$

注意到必然有 $\phi_L(\vec{y}) \leq \phi(\vec{y}) \leq \phi_U(\vec{y})$，于是可构建

$$\begin{aligned} g_L(\vec{q}) &= \sum_{\vec{y} \in \mathscr{Y}} \phi_L(\vec{y}) P(\vec{y}, \vec{p}), \\ &= 1 - \prod_{j=1}^{I} \left(1 - \prod_{i \in \mathscr{P}_j} q_i \right) \end{aligned} \tag{4.58}$$

和

$$\begin{aligned} g_U(\vec{q}) &= \sum_{\vec{y} \in \mathscr{Y}} \phi_L(\vec{y}) P(\vec{y}, \vec{p}), \\ &= \prod_{j=1}^{J} \left(1 - \prod_{i \in \mathscr{C}_j} (1 - q_i) \right) \end{aligned} \tag{4.59}$$

作为 MC 方法估算 $g(\vec{q})$ 的上下界。这个估算是重要的，因为显然，满足 $\phi_U(\vec{y})=1$ 的 $\vec{y}$ 要比 $\vec{y}$ 的全部 2^m 种组合少得多。因此这种界的约束可以极大地降低模拟结果的方差。类似（4.49），我们可以构建概率密度函数

$$P(\vec{y},\vec{q},\lambda_L,\lambda_U)=\frac{\phi_U(\vec{y})-\phi_L(\vec{y})}{g_U(\vec{q})-g_L(\vec{q})}P(\vec{y}-\vec{q}),\vec{y}\in\mathscr{Y}. \tag{4.60}$$

若随机向量 $\vec{Y}$ 服从 $P(\vec{y},\vec{q},\lambda_L,\lambda_U)$，则

$$\psi(\vec{Y})=g_L(\vec{q})+(g_U(\vec{q})-g_L(\vec{q}))\phi(\vec{Y}) \tag{4.61}$$

的期望是 $g(\vec{q})$ 且

$$\begin{aligned}\mathrm{var}\phi(\vec{Y}) &= (g_U(\vec{q})-g(\vec{q}))(g(\vec{q})-g_L(\vec{q}))\\ &\le \frac{(g_U(\vec{q})-g_L(\vec{q}))^2}{4}.\end{aligned} \tag{4.62}$$

现在总结一下 MC 模拟网络可靠性的算法，即算法4.5。

算法 4.5 估算 $s-t$ 网络可靠性 $g(\vec{q})$

目标 估算 $g(\vec{q})$.
输入 网络 $G=(\mathscr{V},\mathscr{E})$, 起点 s, 终点 t, 网络可靠性向量 $\vec{q}$, 置信水平 δ.
输出 $g(\vec{q})$ 的无偏点估计 $\hat{g}_n(\vec{q})$, $\mathrm{var}\hat{g}_n(\vec{q})$, 以及关于 δ 的置信区间.

1. 计算 $g_L(\vec{q})$ 和 $g_U(\vec{q})$.
2. $j=1$, $S=0$.
3. While $j\le n$:
   (a) 生成 $\vec{Y}=(Y_1,\cdots,Y_m)$ 服从概率密度函数 $P(\vec{y},\vec{q},\lambda_L,\lambda_U)$.
   (b) $\phi(\vec{X}^{(j)})=0$.
   (c) If s 和 t 连通，$\phi(\vec{Y})=1$.
   (d) $S=S+\phi(\vec{Y}$.
   (e) j = j + 1.
4. 做统计分析:
   (a) 计算 $\hat{g}_n(\vec{q})=g_L(\vec{q})+[g_U(\vec{q})-g_L(\vec{q})]S/n$ 作为 $g(\vec{q})$ 的点估计.
   (b) 计算 $V[\hat{g}_n(\vec{q})]=(g_U(\vec{q})-\hat{g}_n(\vec{q}))(\hat{g}_n(\vec{q})-g_L(\vec{q}))/(n-1)$ 作为 $\mathrm{var}\hat{g}_n(\vec{q})$ 的点估计.
   (c) 根据置信水平计算百分之 $100\times(1-\delta)$ 的置信区间:

$$I_i(S,n,\delta)=g_L(\vec{q})+(g_U(\vec{q})-g_L(\vec{q}))\tilde{I}_i,i=1,2.$$

   这里 $\tilde{I}_i$ 是原 MC 中对应的置信区间界.

算法4.5还有几个细节没有说明，首先是如何产生 P_i 和 C_i 集合，以及如何判定在 G 中的两点 s 和 ts 是否连通。这些是涉及图论的算法，资料很多，大家自己查阅。其次是如何产生分布 $P(\vec{y},\vec{q},\lambda_L,\lambda_U)$，由于问题复杂，参考书建议直接采用 AR 方法，也即先根据 $\vec{q}$ 随机产生 $\vec{y}$，然后拒绝 $\phi_U(\vec{y})=0$ 和 $\phi_L(\vec{y})=1$ 的样本。如何直接产生满足要求的样本是一个可以考虑的问题。但这个问题有一定的深度和难度。

参考文献

[1] M Abramowitz and I Stegun. *Handbook of Mathematical Functions with Tables*. U.S. Govt. Print. Off.„ 1970.

[2] Joachim H Ahrens and Ulrich Dieter. Efficient table-free sampling methods for the exponential, cauchy, and normal distributions. *Communications of the Acm*, 31(11):1330–1337, 1988.

[3] Colin R. Blyth and Harold A. Still. Binomial confidence intervals. *Publications of the American Statistical Association*, 78(381):108–116, 1983.

[4] G. E. P. Box and Mervin E. Muller. A note on the generation of random normal deviates. *Annals of Mathematical Statistics*, 29(2):610–611, 1958.

[5] Richard P. Brent. Uniform random number generators for supercomputers. In *Proc Fifth Australian Supercomputer Conference*, pages 95–104, 1992.

[6] Lih Yuan Deng and Hongquan Xu. A system of high-dimensional, efficient, long-cycle and portable uniform random number generators. *Acm Transactions on Modeling & Computer Simulation*, 13(4):299–309, 2003.

[7] U. Dieter and J. H. Ahrens. A combinatorial method for the generation of normally distributed random numbers. *Computing*, 11(2):137–146, 1973.

[8] Gregory W. Fischer, Ziv Carmon, Ariely Dan, Gal Zauberman, and Pierre L'Ecuyer. Good parameters and implementations for combined multiple recursive random number generators. *Operations Research*, 47(1):159–164, 1999.

[9] George S. Fishman. *Monte Carlo: Concepts, Algorithms, and Applications. Springer Series in Operations Research*, volume 75. Springer-Verlag, 1995.

[10] Jr Hastings, Cecil. *Approximations for digital computers*. Princeton University Press„ 1955.

[11] Wassily Hoeffding. Probability inequalities for sums of bounded random variables. *Publications of the American Statistical Association*, 58(301):13–30, 1963.

[12] Pierre L'Ecuyer and Tezuka Shu. Structural properties for two classes of combined random number generators. *Mathematics of Computation*, 57(196):735–746, 1991.

[13] D. H Lehmer. Mathematical methods in large-scale computing units. *Proc. of 2nd Symp. on Large-Scale Digital Calculating Machinery*, 26:141–146, 1949.

[14] Rudolf Lidl and Harald Niederreiter. *Introduction to finite fields and their applications*. Cambridge University Press, 1986.

[15] G. Marsaglia, M. D. Maclaren, and T. A. Bray. *A fast procedure for generating normal random variables*. ACM, 1964.

[16] George Marsaglia. Random numbers fall mainly in the planes. *Proceedings of the National Academy of Sciences of the United States of America*, 61(1):25–28, 1968.

[17] George Marsaglia. The squeeze method for generating gamma variates. *Computers & Mathematics with Applications*, 3(4):321–325, 1977.

[18] George Marsaglia and Wai Wan Tsang. The 64-bit universal rng. *Statistics & Probability Letters*, 66(2):183–187, 2004.

[19] George Marsaglia and Zaman Wai Wan Tsang. Toward a universal random number generator. *Statistics & Probability Letters*, 9(1):35–39, 1990.

[20] M Matsumoto. Dynamic creation of pseudorandom number generator. *Monte Carlo and Quasi-Monte Carlo methods 1998*, 48(6):R4211–R4214, 2000.

[21] Makoto Matsumoto and Yoshiharu Kurita. Twisted gfsr generators. *Acm Transactions on Modeling & Computer Simulation*, 2(3):179–194, 1992.

[22] Makoto Matsumoto and Yoshiharu Kurita. Twisted gfsr generators ii. *Acm Transactions on Modeling & Computer Simulation*, 4(3):254–266, 1994.

[23] Makoto Matsumoto and Takuji Nishimura. Mersenne twister:a 623-dimensionally equidistributed uniform pseudo-random number generator. *Acm Transactions on Modeling & Computer Simulation*, 8(1):3–30, 1998.

[24] Harald Niederreiter. *Random number generation and quasi-Monte Carlo methods*. Society for Industrial and Applied Mathematics, 1992.

[25] J. Scott Provan and Michael O. Ball. Computing network reliability in time polynomial in the number of cuts. *Operations Research*, 32(3):516–526, 1984.

[26] Gregory G. Rose. Kiss: A bit too simple. *Cryptography & Communications*, 2011:1–15, 2014.

[27] Rotenberg. A new pseudo-random number generator. *Journal of the Acm*, 7(1):75–77, 1960.

[28] Mutsuo Saito and Makoto Matsumoto. Simd-oriented fast mersenne twister: a 128-bit pseudorandom number generator. In *and Quasi-Monte Carlo Methods 2006*, pages 607–622, 2008.

[29] Tezuka Shu. *Uniform Random Numbers: Theory and Practice*. 1995.

[30] Tezuka Shu, Pierre L'Ecuyer, and Raymond Couture. On the lattice structure of the add-with-carry and subtract-with-borrow random number generators. *Acm Transactions on Modeling & Computer Simulation*, 3(4):315–331, 1993.

[31] Olga Taussky and John Todd. Generation and testing of pseudo-random numbers. *Symposium on Monte Carlo Methods University of Florida*, pages 15–28.

[32] John Von Neumann. 13. various techniques used in connection with random digits. *Appl. Math Ser*, 12(36-38):5, 1951.

[33] 布鲁斯 · 谢克特. 我的大脑敞开了. 上海译文出版社, 2016.

[34] 康崇禄. 蒙特卡罗方法理论和应用. 科学出版社, 2015.