利用蒙特卡洛估算积分

吕艺

517021910745

December 27, 2018

1 适用范围

在概率论与数理统计中，我们常常使用数学期望来描述随机变量的数字特征，其中涉及到函数在区间上的积分。在高等数学中，我们通常采用寻找原函数的方法来计算积分值，然而在实际生活中，很多随机变量的分布函数都是超越函数，不存在原函数，故寻找原函数这一方法失效。当随机变量维度较低时，可以采用数值插入的方法。可当随机变量维度较高时，数值插入的方法就显得力不从心。此时通过蒙特卡洛方法，我们可以将问题转换为随机取样求平均值，从而进行积分的近似计算。

2 数学原理

假设我们有一个函数 $k(x)$, 当 $k(x)$ 满足 $k(x) \geq 0, a \leq x \leq b$, 且 $\int_a^b k(x) = C < \infty$, 则我们可以定义 k(x) 对应的概率密度函数

$$p(x) = \frac{k(x)}{C} \tag{2.1}$$

根据这一原理，我们可以将复杂的分布函数转换为较为简单的分布函数。由于计算机的内部算法实现，大多计算机内部只能生成满足均匀分布或者高斯分布的随机数，这一转换对于利用计算机进行随机模拟有着很大的帮助。

利用类似思想，假设我们要计算的积分为 $\int_a^b g(x) = I$, 我们可将 g(x) 分解为 $h(x)$ 和 $p(x)$(其中 $p(x)$ 通常为较为简单的概率密度函数，以方便取样模拟)。之后我们取一组相互独立，且在区间 [a,b] 上满足 $p(x)$ 分布的随机变量 X_i，那么 $h(X_i)$ 也为独立同分布的随机变量。

$$E(h(X_i)) = \int_a^b h(X_i)\dot{p}(x) = \int_a^b g(X_i) = C \tag{2.2}$$

则根据 Khintchine 定理，X_i 满足大数定律，

$$\lim_{N to \infty} P(|\frac{1}{N}\sum_{i=1}^{N} h(X_i) - \frac{1}{N}\sum_{i=1}^{n} E(X_i)| < \epsilon) = 1 \tag{2.3}$$

因此，当 N 充分大的时候我们可以近似认为:

$$\frac{1}{N}\sum_{i=1}^{N} h(X_i) \approx \frac{1}{N}\sum_{i=1}^{n} E(X_i) \tag{2.4}$$

$$C \approx \frac{1}{N}\sum_{i=1}^{N} h(X_i) \tag{2.5}$$

3 具体实现

设待求积分为 $\int_a^b g(x)$, 根据 a，b 取值的不同，我们可以将模拟近似估计积分分为以下两种情况，分别为定积分和无穷积分。

1) 定积分计算：$\int_0^1 xe^x$

首先通过分部积分确定该积分的值为 1，之后利用 MCMC 方法模拟结果进行比较。由于在计算机中**定区间均匀分布**的随机数比较容易获得，这里将 g(x) 拆分为:

$$h(x) = xe^x \qquad p(x) = 1$$

之后通过生成一满足服从 (0,1) 上均匀分布的随机序列，并将它们通过 h(x) 进行映射后的结果取平均值，从而得出随机模拟的估计值。

估算结果如下图所示。通过观察，在随机序列中的元素个数小于 10^3 时，MCMC 估算误差较大，当随机序列个数大于 10^3 时，MCMC 估算较为精确。由此可见，MCMC 方法适用于估算定区间上的积分，且元素个数越大，MCMC 的估计值与积分的实际值更接近。

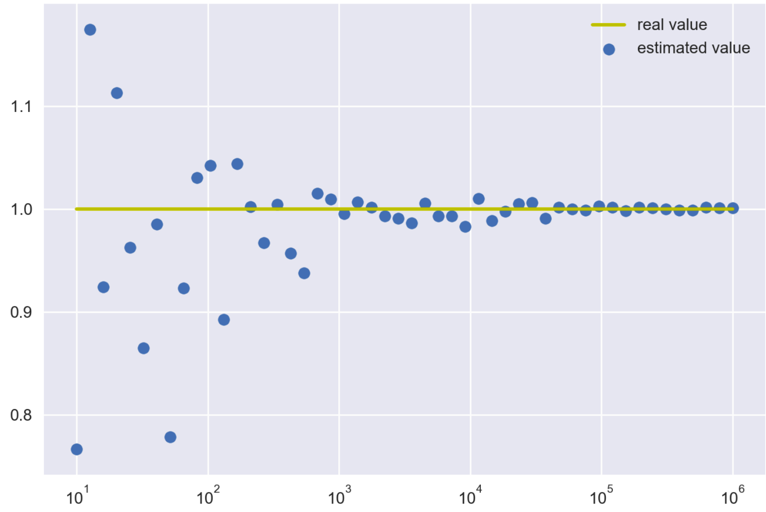

Figure 3.1: 利用 MCMC 估算 $\int_0^1 xe^x$

2）无穷积分计算: $\int_{-\infty}^{\infty} e^{-3x^2}$

通过寻找原函数的方法，求出该积分的值为 $\sqrt{\frac{\pi}{3}}$。之后利用 MCMC 方法模拟结果进行比较。由于在计算机中比较容易取得**满足正态分布的无穷分布随机数**, 这里将 g(x) 拆分为:

$$h(x) = \sqrt{2\pi}e^{\frac{-5x^2}{2}} \qquad p(x) = \frac{1}{\sqrt{2\pi}}e^{\frac{x^2}{2}}$$

之后通过生成一满足服从 $-\infty \sim \infty$ 的随机序列，并将它们通过 h(x) 进行映射后的结果取平均值，从而得出随机模拟的估计值。

模拟结果如下图。当随机序列元素个数在 10^5 个左右时，MCMC 估算 $\int_{-\infty}^{\infty} e^{-3x^2}$ 与实际值已十分接近。这说明利用 MCMC 方法估算积分上下限都为无穷时的无穷积分是有效的。

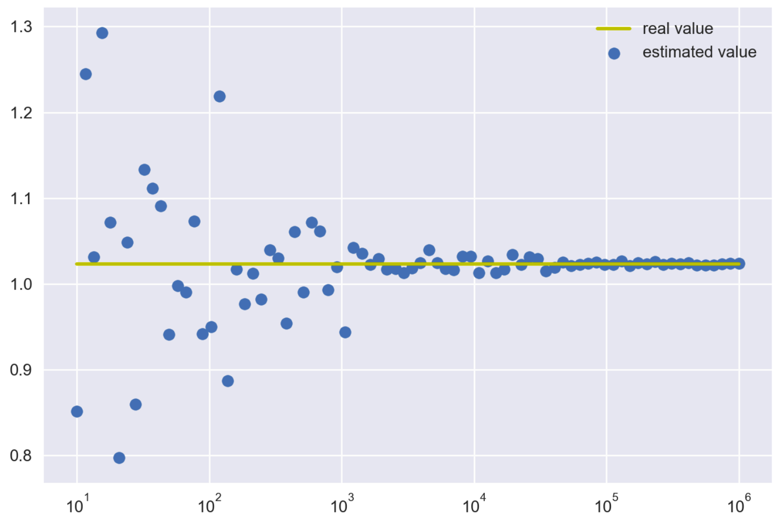

Figure 3.2: MCMC 估计 $\int_{-\infty}^{\infty} e^{-3x^2}$

3）无穷积分计算：$\int_0^{\infty} e^{-3x^2}$

通过寻找原函数的方法，求出该积分的值为 $\sqrt{\frac{\pi}{3}}$。之后利用 MCMC 方法模拟结果进行比较。由于在计算机中比较容易取得**无穷分布的随机数满足正态分布**，这里将 g(x) 拆分为:

$$h(x) = \sqrt{2\pi} e^{\frac{-5x^2}{2}} \qquad p(x) = \frac{1}{\sqrt{2\pi}} e^{\frac{x^2}{2}}$$

之后通过生成一满足服从 $0 \sim \infty$ 的随机序列。由于取值区间为 $0 \sim \infty$，故将小于零的随机数映射后的值置为零。即将 h(x) 设为分段函数：

$$h(x) = \begin{cases} \sqrt{2\pi} e^{\frac{-5x^2}{2}} & x \geq 0 \\ 0 & otherwise \end{cases} \tag{3.1}$$

并将它们通过 h(x) 进行映射后的结果取平均值，从而得出随机模拟的估计值。由于取值区间在

模拟结果如下。当随机序列元素个数在 10^4 个左右时，MCMC 估算 $\int_{-\infty}^{\infty} e^{-3x^2}$ 与实际值已十分接近。这说明利用 MCMC 方法估算积分上下限中有一个为无穷时的无穷积分是有效的。

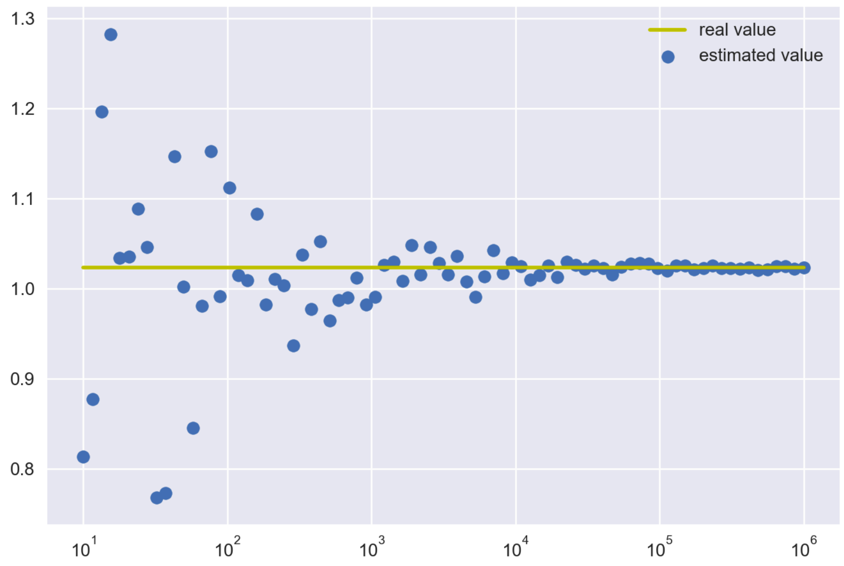

Figure 3.3: MCMC 估计 $\int_0^\infty e^{-3x^2}$

4 MCMC 高维拓展

当被积函数的维度逐渐增加时，利用传统寻找原函数求积分值的复杂度也大幅上升，而利用 MCMC 方法模拟估算出积分值则是一种很好的选择。我们可以采用平均期望法来解决这一问题。

假设我们需要解决的高维积分函数为 $f(x_1, x_2, ..., x_n)$, 积分区域为 D，我们可以在区域 D 中随机生成 N 个点并利用下式进行计算：

$$\hat{f} = \frac{|D|}{N} \sum_{i=1}^{N} f(x_1^{(i)}, x_2^{(i)}, ..., x_n^{(i)}) \tag{4.1}$$

其中在这里 D 为积分区域的体积，在这里 $|D| = \prod_{i=1}^{n}(b_i - a_i)$

5 平均期望法探究

接下来我们来探究这种方法的准确性，分别从是否无偏和是否为一致估计量来考虑。为运算简便考虑，这里仅考虑一维情况。

$$E(\hat{I}) = E(\frac{b-a}{N} \sum_{i=1}^{N} f(X_i)) = \frac{b-a}{N} \sum_{i=1}^{N} E(f(X_i)) = \int_a^b f(x) = I \tag{5.1}$$

故这种估算方法采用的是无偏估计量。

之后我们再来研究其是不是一致估计量。

$$\sigma_{\hat{i}}^2 = \frac{1}{N}((b-a)\int_a^b f^2(x) - I^2)) = \frac{1}{N}(E(f^2) - I^2) \propto N^{-1} \tag{5.2}$$

故这种估算方法采用的是一致估计量。

6 原函数法与 MCMC 法对比

原函数法的优点是计算求得的是真实值，没有误差，但局限在于可求出原函数的积分较少，适用范围小，且计算复杂度高，不易操作。

MCMC 方法的优点在与适用性广，不局限于几何形状与问题维数的限制，且容易编程模拟。然而 MCMC 方法求解的值毕竟是估算，与真实值之间存在着一定误差，且需要大量模拟次数才能得到较为精确的估算值，收敛速度较慢。

7 附录

1）定积分计算：$\int_0^1 xe^x$ 具体代码实现

```
import matplotlib.pyplot as plt
import numpy as np

plt.style.use('seaborn')
N = np.arrange(10, 1000, 1000000)
result = []
for item in N:
    X = np.random.rand(item)
    Y = X*np.exp(X)
    result.append(sum(Y)/item)

plt.scatter(N, result, label='estimated value')

plt.plot([10, 10 ** 6], [np.sqrt(2*np.pi)/np.sqrt(6),
np.sqrt(2*np.pi)/np.sqrt(6)], 'y', linewidth=2,
label='real value')

plt.legend()
plt.show()
```

2）无穷积分计算：$\int_{-\infty}^{\infty} e^{-3x^2}$ 具体代码实现

```
import matplotlib.pyplot as plt
import numpy as np

plt.style.use('seaborn')
N = np.logspace(1, 6, 80)
result = []
for item in N:
    item = int(item)
    X = np.random.randn(item)
    Y = np.sqrt(2*np.pi)*np.exp(-X*X*5/2)
    result.append(sum(Y)/item)

plt.scatter(N, result, label='estimated value')
plt.xscale('log')

plt.plot([10, 10 ** 6], [np.sqrt(2*np.pi)/np.sqrt(6),
np.sqrt(2*np.pi)/np.sqrt(6)], 'y', linewidth=2,
label='real value')

plt.legend()
plt.show()
```

3）无穷积分计算：$\int_{0}^{\infty} e^{-3x^2}$ 具体代码实现

```
import matplotlib.pyplot as plt
import numpy as np
plt.style.use('seaborn')
N = np.logspace(1, 6, 80)
result = []
for item in N:
    item = int(item)
    X = np.random.randn(item)
    Y = np.sqrt(2*np.pi)*np.exp(-X*X*5/2)
    for i in range(item):
```

```
        if X[i] < 0:
            Y[i] = 0
    result.append(sum(Y)/item)

plt.scatter(N, result, label='estimated value')
plt.xscale('log')

plt.plot([10, 10 ** 6], [np.sqrt(2*np.pi)/np.sqrt(6)/2,
np.sqrt(2*np.pi)/np.sqrt(6)/2], 'y', linewidth=2,
label='real value')

plt.legend()
plt.show()
```