随机过程笔记II：一维随机游走的首次返回概率

拉普拉斯算符
数学话题下的优秀答主

假设有一个粒子从原点出发在一维直线上做对称随机游走。为了描述这个运动，我们首先定义一列随机变量 $X_n, n \in \mathbb{Z}^+$ ，它的概率分布为 $P(X_n=1)=P(X_n=-1)=\frac{1}{2}$ ，其中 $n \geq 1$ 为粒子的移动步数。显然，经过 n 步后粒子的坐标为

$$Z_n=\sum_{k=1}^{n} X_k$$

现在要计算从原点出发，经过 $n \geq 1$ 步后粒子刚好第一次返回原点的概率。

第一节：粒子返回原点的概率

很明显，粒子一定要经过偶数步移动才有可能重回原点。因此，我们只需要计算粒子经过 $2n \geq 2$ 步后首次返回原点的概率。首次返回原点的概率不容易计算，但是我们可以很容易写出经过 $2n$ 步后，粒子刚好在原点的概率，为

$$P(Z_{2n}=0)=C_{2n}^{n}\left(\frac{1}{2}\right)^{2n}$$

我们可以从这个出发，计算经过 $2n$ 步后粒子首次返回原点的概率。

第二节：粒子返回原点的次数

为了计算粒子首次返回原点的概率，我们可以首先计算粒子经过 $2n$ 步后第 $m \geq 1$ 次重返原点的概率。将这个概率记做：

$$P^{(m)}(Z_{2n}=0)$$

对于任意的 $n \geq 1$ ， m 的取值范围为 $1 \leq m \leq n$. 当 $m \geq n+1$ 时，我们总有 $P^{(m)}(Z_{2n}=0)=0$.

对于概率 $P^{(m)}(Z_{2n}=0)$ ，我们有这样的一个递归公式：

$$P^{(m)}(Z_{2n}=0)=\sum_{k=m-1}^{n-1} P^{(m-1)}(Z_{2k}=0)P^{(1)}(Z_{2n-2k}=0), m \geq 2$$

这个递归公式的物理意义是，粒子经过 $2n$ 步返回原点 m 次的概率等于粒子在前 $2k<2n$ 步返回原点 $m-1$ 次的概率乘以粒子在剩下的 $2n-2k$ 步返回原点 1 次的概率，再遍历所有可能的 k .

记概率 $P^{(1)}(Z_{2n}=0), n \geq 1$ 的生成函数为 $f(x)$ ，也就是定义

$$f(x)=\sum_{n=1}^{\infty} P^{(1)}(Z_{2n}=0)x^n$$

如果我们可以得到生成函数 $f(x)$ 的表达式，我们就可以很容易计算出概率 $P^{(1)}(Z_{2n}=0), n \geq 1$. 接下来我们研究如何计算这个生成函数。

第三节：概率分布的生成函数

为了计算生成函数 $f(x)$ ，我们首先考虑从这个生成函数出发，我们可以得到什么信息。根据这个递归公式：

$$P^{(2)}(Z_{2n}=0)=\sum_{k=1}^{n-1} P^{(1)}(Z_{2k}=0)P^{(1)}(Z_{2n-2k}=0) ,$$

概率分布 $P^{(2)}(Z_{2n}=0)$ 的生成函数为

$$\begin{aligned} f^{(2)}(x) &= \sum_{n=2}^{\infty} P^{(2)}(Z_{2n}=0)x^n \\ &= \sum_{n=2}^{\infty}\sum_{k=1}^{n-1} P^{(1)}(Z_{2k}=0)P^{(1)}(Z_{2n-2k}=0)x^n \\ &= \sum_{k=1}^{\infty}\sum_{n=k+1}^{\infty} P^{(1)}(Z_{2k}=0)x^k P^{(1)}(Z_{2n-2k}=0)x^{n-k} \\ &= \left(\sum_{k=1}^{\infty} P^{(1)}(Z_{2k}=0)x^k\right)^2 \\ &= \Big(f(x)\Big)^2 \end{aligned}$$

使用数学归纳法，假设概率分布 $P^{(m-1)}(Z_{2n}=0), m-1 \geq 3$ 的生成函数为 $\Big(f(x)\Big)^{m-1}$ ，那么根据概率递归公式，我们有概率分布 $P^{(m)}(Z_{2n}=0)$ 的生成函数为

$$\begin{aligned} f^{(m)}(x) &= \sum_{n=m}^{\infty} P^{(m)}(Z_{2n}=0)x^n \\ &= \sum_{n=m}^{\infty}\sum_{k=m-1}^{n-1} P^{(m-1)}(Z_{2k}=0)P^{(1)}(Z_{2n-2k}=0)x^n \\ &= \sum_{k=m-1}^{\infty}\sum_{n=k+1}^{\infty} P^{(m-1)}(Z_{2k}=0)x^k P^{(1)}(Z_{2n-2k}=0)x^{n-k} \\ &= \sum_{k=m-1}^{\infty} P^{(m-1)}(Z_{2k}=0)x^k \sum_{n-k=1}^{\infty} P^{(1)}(Z_{2n-2k}=0)x^{n-k} \\ &= \Big(f(x)\Big)^{m-1} f(x) \\ &= \Big(f(x)\Big)^{m} \end{aligned}$$

我们可以根据粒子返回原点的次数对回归概率做分割，把粒子经过 $2n$ 步返回原点的概率写为：

$$P\Big(\text{经过}2n\text{步随机游走，粒子返回原点}\Big)=\sum_{m=1}^{n} P\Big(\text{经过}2n\text{步随机游走，粒子第}m\text{次返回原点}\Big)$$

用公式表示为

$$P(Z_{2n}=0)=\sum_{m=1}^{n} P^{(m)}(Z_{2n}=0)$$

记概率分布 $P(Z_{2n}=0)$ 的生成函数为

$$B(x)=\sum_{n=1}^{\infty} P(Z_{2n}=0)x^n$$

根据概率划分的定义，生成函数 $B(x)$ 可以用 $f(x)$ 表示为

$$B(x)=\sum_{m=1}^{\infty}\Big(f(x)\Big)^m=\frac{f(x)}{1-f(x)}$$

这里，我们假设了 $|f(x)|<1$ 在定义域内几乎处处成立。从这个公式出发，我们可以解出 $f(x)$ 为

$$f(x)=\frac{B(x)}{1+B(x)}$$

所以我们只需知道 $B(x)$ 就可以得到 $f(x)$. 根据定义，

$$\begin{aligned} B(x) &= \sum_{n=1}^{\infty} P(Z_{2n}=0)x^n \\ &= \sum_{n=1}^{\infty} C_{2n}^{n}\left(\frac{1}{2}\right)^{2n} x^n \\ &= \frac{1}{\sqrt{1-x}}-1, |x|<1 \end{aligned}$$

所以

$$f(x)=1-\sqrt{1-x}, |x|<1$$

显然，在定义域内 $|f(x)|<1$ 恒成立。

第四节：从概率生成函数的系数得到概率分布

现在已经知道粒子首次回归原点的概率生成函数为

$$f(x)=1-\sqrt{1-x}, |x|<1$$

对这个函数做泰勒级数展开得到

$$f(x)=\frac{1}{2}x+\sum_{n=2}^{\infty}\frac{(2n-3)!!}{(2n)!!}x^n, |x| \leq 1$$

对比幂级数的系数，我们有

$$P^{(1)}(Z_{2n}=0)=\begin{cases} \frac{1}{2}, n=1 \\ \frac{(2n-3)!!}{(2n)!!}, n \geq 2 \end{cases}$$

或者等价地，粒子刚好在第 $2n$ 步首次重返原点的概率为

$$P^{(1)}(Z_{2n}=0)=\frac{1}{2n-1}C_{2n}^{n}\left(\frac{1}{2}\right)^{2n}, n \geq 1$$

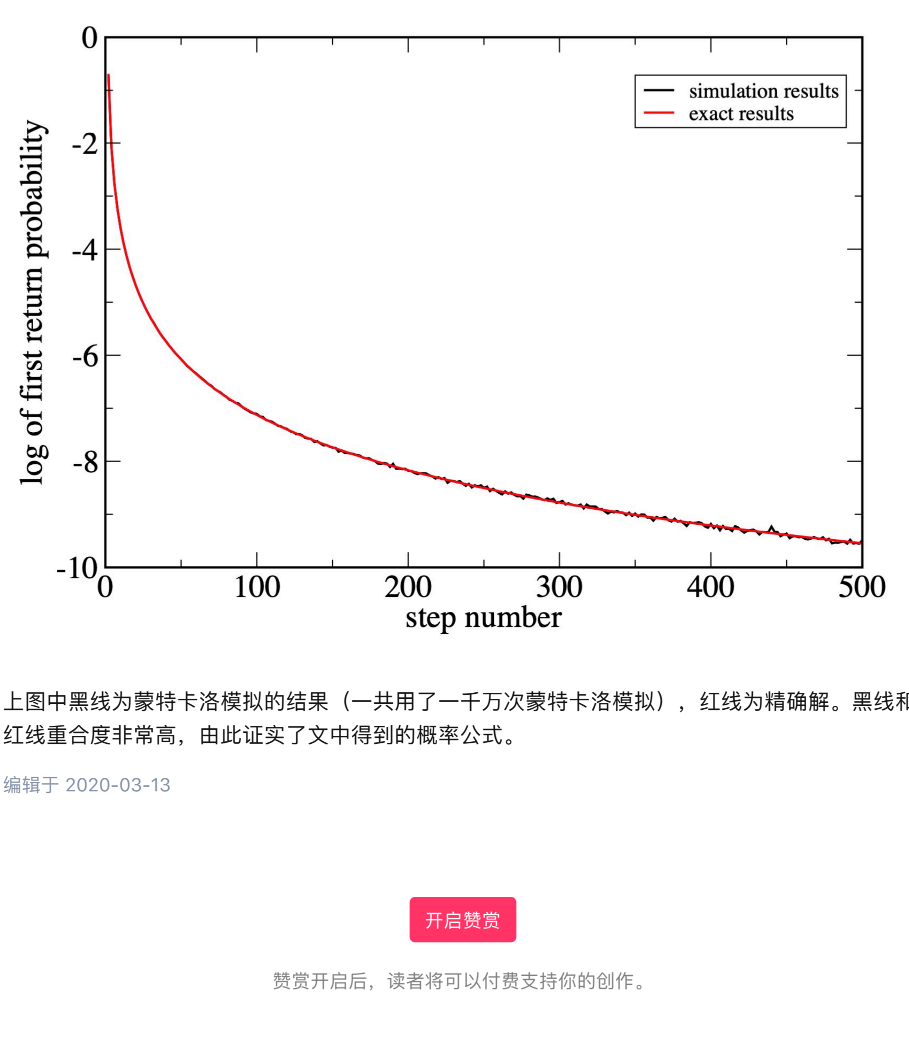
计算完毕。

第五节：蒙特卡洛模拟结果

这里，我使用蒙特卡洛模拟了一下粒子的首次回归频率，并且与精确解做了比较。蒙特卡洛模拟和对比代码如下：

```
import numpy as np
import random
import scipy.special as sc
import datetime

def random_move():
    r = random.uniform(0, 1)
    if r < 0.5:
        return -1
    else:
        return 1

def first_return_time():
    counter = 0
    z = 0
    counter += 1
    iter_max = 500
    x = random_move()
    z += x
    while z != 0:
        x = random_move()
        z += x
        counter += 1
        if counter > iter_max:
            return -1
    return counter

def random_walks(N):
    first_return_times = []
    print "Running Monte Carlo simulations ... "
    for i in range(N):
        print i, N, datetime.datetime.now()
        step_number = first_return_time()
        if step_number > 0:
            first_return_times.append(step_number)
    stats = dict()
    unit = 1.0/float(N)
    for i in range(len(first_return_times)):
        temp = first_return_times[i]
        if temp in stats:
            stats[temp] += unit
        else:
            stats[temp] = unit
    pairs = stats.iteritems()
    result = sorted(pairs, key = lambda (k, v): k)
    writer = open("simulation_results.txt", "w")
    for i in range(len(result)):
        writer.write(str(result[i][0]) + ", " + str(np.log(result[i][1])) + "\n")
    writer.close()

    writer = open("exact_results.txt", "w")
    for i in range(len(result)):
        step_number = result[i][0]
        assert(step_number%2 == 0)
        n = float(step_number/2)
        assert(n > 0)
        exponent = -np.log(2*n-1) - 2*n*np.log(2) + sc.gammaln(2*n + 1) - 2.0 * s
        #exact_result = np.exp(exponent)
        exact_result = exponent
        writer.write(str(step_number) + ", " + str(exact_result) + "\n")
    writer.close()

def main():
    import sys
    if len(sys.argv) != 2:
        print "simulation_times = sys.argv[1]. "
        return -1

    simulation_times = int(sys.argv[1])
    random_walks(simulation_times)
    return 0

if __name__ == "__main__":
    import sys
    sys.exit(main())
```

结果如下（横坐标为随机游走的步数，纵坐标为首次回归的概率的对数）：



上图中黑线为蒙特卡洛模拟的结果（一共用了一千万次蒙特卡洛模拟），红线为精确解。黑线和红线重合度非常高，由此证实了文中得到的概率公式。

编辑于 2020-03-13

微棠 2020-05-24
以前学的工科的随机过程，现在感觉和没学一样 。想问下作者用的什么教材或课程？非常感谢
1

拉普拉斯算符 (作者) 回复 微棠 2020-05-24
Stochastic processes by Sheldon Ross
1 推荐 删除

飘渺鸟 2020-03-22
用停时定理求会更简单。你这个算是古典方法吧？
1

拉普拉斯算符 (作者) 回复 飘渺鸟 2020-03-22
初学随机过程
赞 推荐 删除

拉普拉斯算符 (作者) 回复 飘渺鸟 2020-06-23
停时定理是不是martingale stopping theorem? 这个定理挺适合求期望，也可以用来计算概率吗？
1 推荐 删除

这里没有小熊 2020-11-16
答主，这个问题的高维推广有什么可以参考的资料吗。我模拟出来这个概率分布在一维和三维的情况下渐进性质高度一致，有没有什么深层原因
赞

拉普拉斯算符 (作者) 回复 这里没有小熊 2020-11-16
维数变高了问题就会很复杂，一维情况下的很多技巧都不能用了。
赞 推荐 删除

这里没有小熊 回复 拉普拉斯算符 (作者) 2020-11-16
好的谢谢你！ 如果我找到了结果就回来吱一声[大笑]
赞

展开其他 1 条回复

阿岛 2020-06-01
蛮有趣
赞