第二次作业

刘子安 PB20000069

1. 实验环境

- 操作系统：Windows 11中wsl2的Ubuntu 20.04
- Python版本：Python 3.8.10
- matplotlib库版本：3.5.1
- numpy库版本：1.22.3

2. 实验内容

2.1 实验设计

本实验通过Python模拟了二维情况下的随机游走，分别模拟了向四个方向的无偏随机游走、有偏随机游走和有倾向的随机游走。

2.2 算法流程

开始时使用书*Python*编程导论中所使用的方法，但是发现当步数从1-10000并且每个步数重复1000次时代码运行时间过长，于是考虑优化书中的算法。

书中每次运行某一步数的代码时，都从头开始一步一步模拟，这是没有必要的。我们可以对每一步都记下该点的位置，再去求得每一步到原点的距离，这样我们只需要跑一万步就可以得到一万个有用的距离。

2.3 核心代码分析

进行随机游走模拟的`simWalks`函数：

```
def simWalks(walkLengths, numTrials: int, dClass: Drunk):
    """假设walkLengths是非负整数有序数列,numTrials是正整数,
        dClass是Drunk的一个子类。
        模拟numTrials次游走,每次游走walkLengths中的某一步数。
        返回一个列表,表示每次模拟的最终距离"""
    Homer = dClass()
    origin = Location(0, 0)
    r = []
    for _ in range(numTrials):
        f = Field()
        f.addDrunk(Homer, origin)
        t = 0
        distances = []
        for numSteps in walkLengths:
            distances.append(round(walk(f, Homer,numSteps - t), 1))
            t = numSteps
        r.append(distances)
    return r
```

我们对先使用一个for循环，即表示模拟`numTrials`次，对每一次模拟，我们都先实例化一个`Field`对象，再在其中添加一个名为`dClass`的`Drunk`对象，并且对于`walkLengths`中的每个步数进行模拟，第一次先模拟第一项并添加进`distances`列表，之后每次游走这个步数与前一个步数之间的差值，并将与原点的距离添加进`distances`中。最终我们得到了一个存有`walkLengths`里每一步数模拟到原点的距离的列表，并将其添加到`r`中，循环`numTrials`次后就在`r`中得到了`numTrials`个元素，每个元素都是一个列表，包含了每个步数对应的距离。

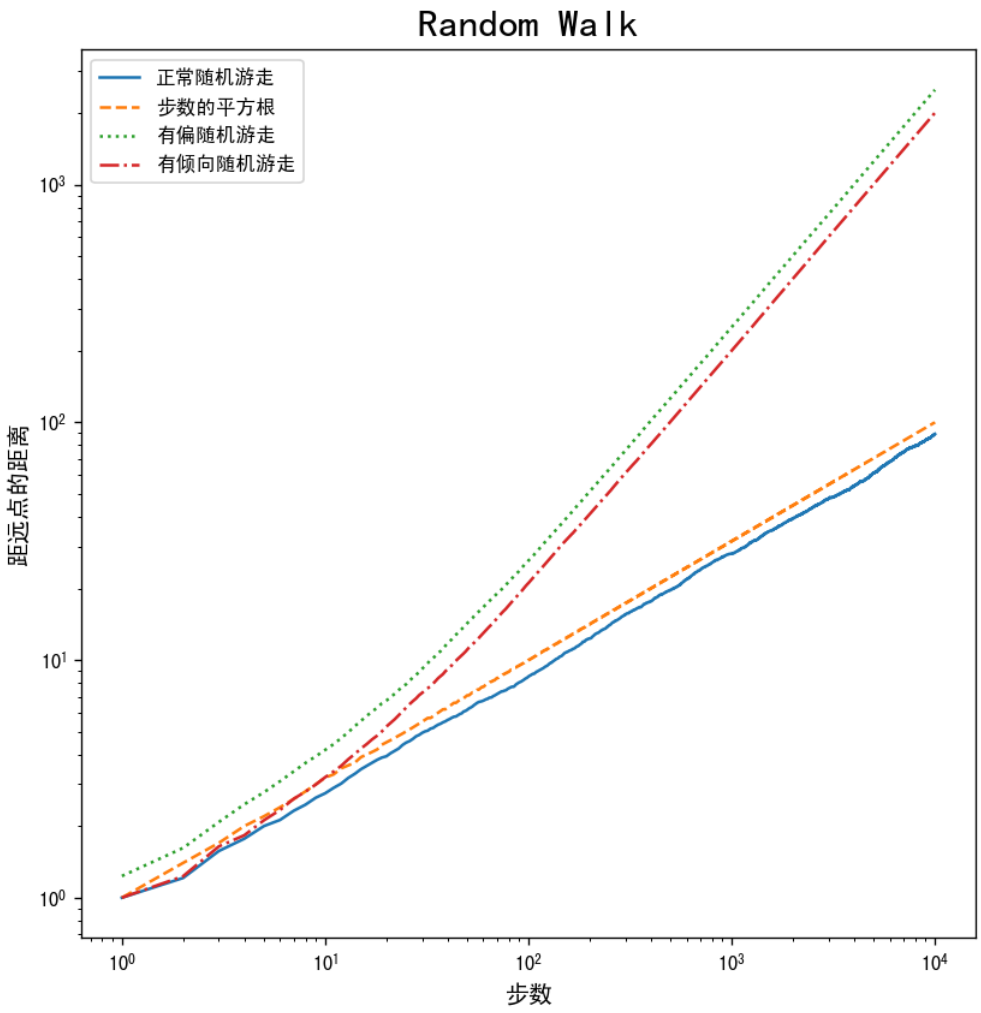
```
def drunkTest(walkLengths, numTrials, dClass: Drunk):
    """假设walkLengths是非负整数序列
        numTrials是正整数,dClass是Drunk的一个子类
        对于walkLengths的每个步数,运行numTrials次simWalk函数,并输出结果"""
    result = np.array(simWalks(walkLengths, numTrials, dClass))
    R = []
    for i in range(len(walkLengths)):
        distances = result[:, i]
        ave_dis = round(sum(distances) / len(distances), 4)
        R.append(ave_dis)
    return R
```

这里使用了上面所说的`simWalks`函数，并且使用了`numpy.array`来将二维数组中的某一列取出来，使用了`result[:, i]`来取出第`i`列并取平均得到每个步数的一千次距离的平均值，再添加到列表`R`中并返回。

在python的主函数中使用`matplotlib.pyplot`绘制了一张图，并且使用了对数坐标。其中有四条线，每一条线都有对应的标识。

```
    plt.rcParams['font.sans-serif'] = ['SimHei']
    plt.figure(num=1, figsize=(8, 8), dpi=120)
    plt.title('Random Walk', fontsize=20)
    walkLengths = range(1, 10000)

    sqrtwalkLengths = [round(i**0.5, 1) for i in walkLengths]
    plt.plot(walkLengths, sqrtwalkLengths, linestyle="--", label="步数的平方根")

    # 正常随机游走
    result1 = drunkTest(walkLengths, 1000, UsualDrunk)
    plt.plot(walkLengths, result1, label="正常随机游走")

    # 有偏随机游走
    result2 = drunkTest(walkLengths, 1000, ColdDrunk)
    plt.plot(walkLengths, result2, label="有偏随机游走", linestyle=":")

    # 有倾向随机游走
    result3 = drunkTest(walkLengths, 1000, TendencyDrunk)
    plt.plot(walkLengths, result3, label="有倾向随机游走", linestyle="-.")

    plt.xscale("log")  # 使用对数坐标
    plt.yscale("log")
    plt.xlabel("步数", fontsize=12)
    plt.ylabel("距远点的距离", fontsize=12)
    plt.legend()
    plt.savefig("RandomWalk.png")
    plt.show()
```

3. 实验结果


Random Walk
正常随机游走
步数的平方根
有偏随机游走
有倾向随机游走
距远点的距离
步数
10^0
10^1
10^2
10^3
10^4