بنام خدا

گزارش تمرین کامپیوتری اول

آرشام لولوهری

۹۹۱۰۲۱۵۶

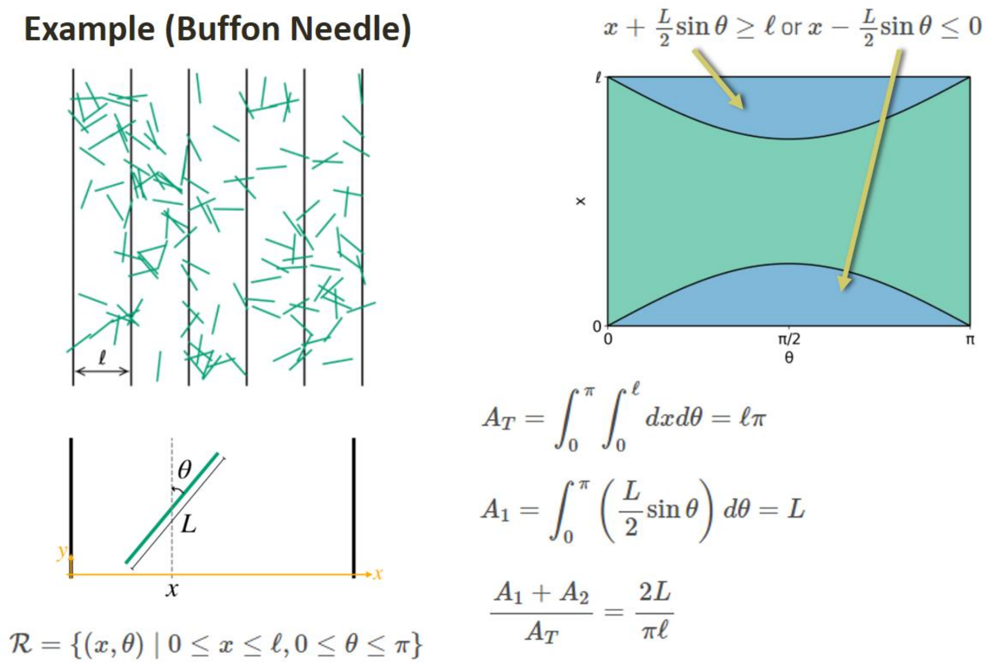
۱)

کد به صورت زیر است:

```
In [23]: from math import pi as pi
         from math import sin as sin
         import numpy as np
         from scipy.stats import bernoulli as ber

         total = 100000 # total experiments
         l = 5 #lines distance
         L = 1 #needle length
         result = np.array([0]*total) # array of experiments results, one if lies across a line, zero otherwise

         x = np.random.uniform(0,5,total) #needles center from previous line
         theta = np.random.uniform(0,pi,total) #angle with lines
         for i in range(total):
             if (x[i]+L/2*sin(theta[i]))>=l or (x[i]-L/2*sin(theta[i]))<=0: #if needle and line intersect
                 result[i] = 1
         p = sum(result)/total #the probability

         pi_app = 2*L/(l*p) #our approximation of pi number
         print(pi_app)

3.137500980469057
```

توضیحات تا جای ممکن در کامنت ها آورده شده است. L فاصله ی خطوط عمودی، L طول هر سوزن، و x,theta نیز مشابه آنچه در صورت سوال گفته شده تعریف شده اند. X,theta هریک صدهزار مقدار دارند که مقدار i ام آنها، به ترتیب فاصله ی مرکز سوزن از خط سمت چپی، و زاویه ی سوزن با خط قائم، در آزمایش i ام است. هرکدام به صورت توزیع تصادفی و یکنواخت در محدوده ی تغییر خودشان مقداردهی میشوند. Result نیز یک آرایه است که نتایج آزمایش ها در آن ریخته میشود. با استفاده از یک حلقه تعیین شده است که اگر در آزمایش i ام، سوزن با خطوط تقاطع داشت، مقدار result[i] برابر ۱ شود. در غیر این صورت مطابق تعریف اولیه، این مقدار برابر با صفر باقی میماند.

شرطی که داخل حلقه برای تقاطع نوشته شده است، بر اساس شکل زیر است:



مطابق این شکل، ابتدای سوزن در نقطه ی

$$x - \frac{L}{2} * \sin(\theta)$$

و انتهای آن در نقطه ی

$$x + \frac{L}{2} * \sin(\theta)$$

قرار دارد.شرط برخورد این است که ابتدای سوزن، عقب تر از خط سمت چپی، و یا انتهای سوزن، جلوتر از خط سمت راستی(واقع در نقطه‌ی l) باشد. این شروط در داخل حلقه نوشته شده اند.

اگر مثل شکل بالا، در یک صفحه دو بعدی، ناحیه آبی رنگ را محدوده ی x,theta های تقاطع سوزن و خطوط، و محدوده ی سبز رنگ را محدوده‌ی عدم تقاطع قرار دهیم، احتمال برخورد برابر با نسبت مساحت آبی رنگ به

مساحت کل مستطیل خواهد بود که مطابق انتگرال گیری بالا، حاصل احتمال برابر با :

$$p = \frac{2L}{\pi l}$$

خواهد بود. پس اگر احتمال را بتوانیم حساب کنیم، عدد پی از رابطه ی

$$\pi = \frac{2L}{lp}$$

بدست می آید. حاصل احتمال در صدهزار آزمایش، برابر با نسبت تعداد ۱ ها(تقاطع ها) در result، به تعداد کل آزمایش هاست که این مقدار در متغیر p ریخته شده است. سپس تخمین ما از عدد پی مطابق فرمول قرمز رنگ بالا، در pi_app ریخته و چاپ شده است. در انتهای تصویر هم یکی از تخمین های عدد پی بعنوان نمونه چاپ شده است.

۲)

احتمال های خواسته شده را به ترتیب p1,p2,p3 مینامیم. ابتدا به صورت دستی، هر احتمال را مطابق جداول حساب میکنیم:

$$p1 = P(A = +a|B = +b) = 0 \cdot 94$$

$$p2 = P(A = +a|B = -b) = 0 \cdot 01$$

$$P(A = -a) = P(A = -a|B = +b)P(B = +b) + P(A = -a|B = -b)P(B = -b)$$

$$p3 = P(A = -a) = 0 \cdot 06 \times 0 \cdot 01 + 0 \cdot 99 \times 0 \cdot 99 = 0 \cdot 9807$$

کد به صورت زیر است:

```
In [49]: import numpy as np
         from scipy.stats import bernoulli as ber

         total = 100000 #total experiments
         x = ber.rvs(p=0.01,size=total) #burglary event
         y = np.array([0]*total) #alarm event
         temp = 0
         for i in range(total):
             if x[i]==1: #if burglary==true,
                 temp = np.random.randint(1,101)
                 if temp<=94:
                     y[i]=1 #alarm is true with probability 0.94
             else:
                 temp = np.random.randint(1,101)
                 if temp==100:
                     y[i]=1 #if burglary == flase, alarm is true with probability 0.01
         s1,s2,s3=0,0,0 #si: total desired outcomes for probability pi
         for i in range(total):
             if x[i]==1: # if burglary is true,
                 if y[i]==1: #if alarm is true,
                     s1=s1+1 #it's desired outcome for p1
                 else:
                     s3=s3+1 #it's desired outcome for p3
             else:
                 if y[i]==1:
                     s2=s2+1 #it's desired outcome for p2
                 else:
                     s3=s3+1 #it's desired outcome for p3
         p1=s1/sum(x) #P(A=+a|B=+b)
         p2=s2/(total-sum(x)) #P(A=+a|B=-b)
         p3=s3/total #P(A=-a)

         print(f"P(A=+a|B=+b) = {p1}")
         print(f"P(A=+a|B=-b) = {p2}")
         print(f"P(A=-a) = {p3}")

         P(A=+a|B=+b) = 0.9540566959921799
         P(A=+a|B=-b) = 0.010042737201572083
         P(A=-a) = 0.9803
```

با توجه به اینکه دزدی (x=1) به احتمال ۰/۰۱ رخ میدهد، آرایه ی x را به صورت مقادیر یک متغیر تصادفی برنولی با احتمال ۰/۰۱ میگیریم که تکرر آزمایش هم به اندازه ی total=100000 بار است.

مقادیر یک یا صفر (روشن شدن یا نشدن) زنگ خطر، که در y ریخته میشوند، بر اساس مقدار x در هر آزمایش مشخص میشود. اگر x[i]=1، به احتمال ۰/۹۴ زنگ روشن میشود. بنابراین برای تعیین مقدار y ، یک متغیر temp را به صورت تصادفی از بین اعداد یک تا صد مقداردهی میکنیم و اگر مقدار آن کمتر مساوی ۹۴ بود(احتمال ۰/۹۴)، مقدار y را از صفر به یک تغییر میدهیم. اگر هم دزدی رخ نداده بود (x[i]=0)، باز به احتمال یک درصد، y[i]=1 است که این موضوع نیز به طور مشابه با استفاده از شرط، اعمال شده است.

حال برای محاسبه احتمال های p1,p2,p3 ، تعداد حالات مطلوب در هر احتمال(s1,s2,s3) را حساب میکنیم. اگر دزدی رخ داده بود، و سپس دیدیم که زنگ خطر روشن شده ، به حالات مطلوب برای p1 افزوده میشود:

S1=s1+1. اگر هم روشن نشده بود به s3 افزوده میشود. اگر هم دزدی نشده بود، سپس دیدیم که زنگ روشن شده، به s2، و در غیر این صورت به s3 افزوده میشود.

تعداد کل حالات برای p1، با توجه به شرط مشخص شده برای آن، تعداد دزدی های رخ داده میباشد ((sum(x)). برای p2 نیز تعداد آزمایش های بدون دزدی است، که باید مقدار حالت قبل را از کل کم کنیم. برای p3 نیز فضای نمونه ، اصلا کوچکتر نشده است. از تقسیم تعداد حالات مطلوب به کل حالات، هر احتمال بدست می آید. یک نمونه از مقادیر حاصل از ران کردن برنامه در انتهای تصویر آمده است که میبینیم بسیار نزدیک به محاسبات دستی خودمان هستند.

۱-۳)

کد به صورت زیر است:

```
In [49]: import numpy as np
         from math import pi as pi
         from random import sample

         def p_centerContain(n):
             total = 10000 #total number of experiments
             result = 0 #number of successful experiments
             nodesList = list(range(0,n)) #list of polygon nodes, from 0 to n-1
             for i in range(0,total):
                 selectedNodes = sample(nodesList,3) #choosing 3 random nodes from list
                 selectedNodes.sort()
                 n1 = selectedNodes[0] #the smallest number from selected nodes
                 n2 = selectedNodes[1] #the medium number
                 n3 = selectedNodes[2] #the largest number
                 theta12 = (n2-n1)*pi/n  #angle between n1,n2
                 theta23 = (n3-n2)*pi/n
                 theta13 = (n-(n3-n1))*pi/n
                 if theta12<pi/2 and theta13<pi/2 and theta23<pi/2: #triangle includes center, if all 3 angles are less than pi/2
                     result = result+1
             probability = result/total
             return probability
         print(p_centerContain(5))

         0.5037
```

درون حلقه، فرایند آزمایش را ۱۰۰۰۰ بار تکرار میکنیم. راس های چندضلعی منتظم را به ترتیب از صفر تا n-1 شماره گذاری میکنیم و سه مورد از آنها را به صورت رندوم انتخاب میکنیم. سپس لیست سه تایی ساخته شده را sort میکنیم تا n1,n2,n3 همواره از شماره های کوچک به بزرگ باشند. هر یک از thetaij ها، زاویه ی بین دو راس ni,nj است.

برای توضیح روند حل، مثالی که در صورت سوال برای n=9 آمده را بررسی میکنیم. اگر رئوس را از بالاترین راس و به صورت ساعتگرد، از صفر تا ۸ شماره گذاری کنیم، رئوس منتخب در این مثال، شماره های ۱ و ۴ و ۸ هستند. پس از سورت کردن داریم: n1=1,n2=4,n3=8 . در یک n ضلعی منتظم محاط داخل دایره، زاویه‌ی هر کمان روی دایره برابر با 2*pi/n است و زوایای مثلث، همگی زوایای محاطی اند که نصف کمان روبروی خود هستند. Theta12 زاویه ی بین n1,n2 است که برابر است با :

(فاصله ی n1,n2 روی دایره، از نظر تعداد کمان)*pi/n

پس:

$$theta12 = (n2 - n1) * \frac{pi}{n} = 3 * \frac{pi}{9}$$

که این را در کد لحاظ کرده ایم. برای theta23 نیز به همین ترتیب داریم:

$$theta23 = (n3 - n2) * \frac{pi}{n} = 4 * \frac{pi}{9}$$

اما برای theta13 باید توجه کنیم که فاصله ی بین n1,n3 ، باید از سمتی حساب شود که با n2 تقاطع نداشته باشد. در واقع در مثال صورت سوال، فاصله ی n1,n3 برابر با ۲ کمان است ، نه ۷ تا. پس برای theta13 داریم:

$$theta13 = (n - (n3 - n1)) * \frac{pi}{n} = 2 * \frac{pi}{9}$$

این هم در کد لحاظ شده است. در نهایت اگر هر سه زاویه محاسبه شده کمتر از ۹۰ درجه باشند، مثلث مرکز دوست است و به تعداد موفقیت های ما در آزمایش، یکی افزوده میشود. پس از ۱۰۰۰۰ بار تکرار آزمایش، احتمال نهایی برابر با تعداد موفقیت تقسیم بر ۱۰۰۰۰ است. در تصویر بالا یک نمونه تست به ازای n=5 انجام شده و احتمال نیز محاسبه شده است.

۳-۲:

کد با توجه به بخش قبل، به صورت زیر است:

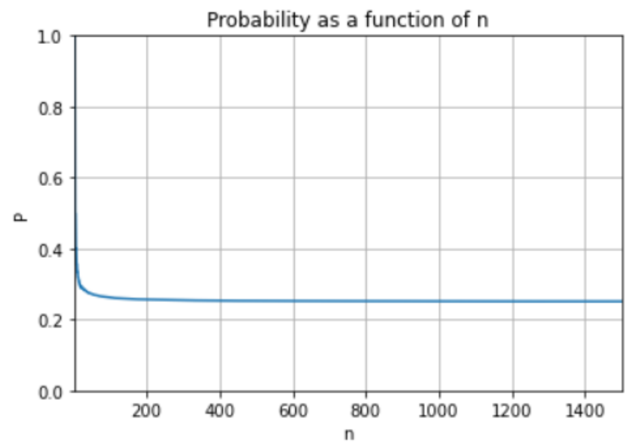
```
In [12]: #3.2
def centerContainCaller():
    MC_Result = np.zeros(1500-3+1) #initializing MC_Result
    MC_Avg = np.zeros(1500-3+1) #initializing MC_Avg
    for i in range(0,1500-3+1):
        MC_Result[i] = p_centerContain(i+3) #recalling previous function, assigning the result in MC_Result
        if i==0:
            MC_Avg[i] = MC_Result[i] #for the first loop
        else:
            MC_Avg[i] = (MC_Avg[i-1]*i+MC_Result[i])/(i+1) #(sum of previous values + new value)/new size
    return MC_Avg
```

ابتدا همه مقادیر دو آرایه ی مذکور را صفر قرار میدهیم. برای تکرار تابع بخش قبل از n=3 تا ۱۵۰۰، یک for زده شده که مقدار MC_Result را مشخص میکند و سپس در هر حلقه، با ضرب میانگین حلقه قبلی در تعداد حلقه قبلی، مجموع مقادیر قبلی را حساب میکند. این مقدار با مقدار جدید ریخته شده در MC_Result جمع شده، تقسیم بر تعداد جدید میشود تا میانگین جدید بدست آید. در حلقه اول هم میانگین برابر با مقدار MC_Result[0] است.

۳-۳:

کد و نتیجه به صورت زیر است:

In [20]:
```
#3.3
from matplotlib import pyplot
MC_avg = centerContainCaller()

pyplot.figure()
pyplot.plot(list(range(3,1500+1)),MC_avg)
pyplot.title('Probability as a function of n')
pyplot.xlabel('n')
pyplot.ylabel('P')
pyplot.grid(True)
pyplot.axis([3,1500+1,0,1])

pyplot.show()

print(MC_avg[-1])
```




```
0.2510405874499331
```

تابع بخش دوم فراخوانی شده و مقدارش در MC_avg ریخته شده است. به ازای n های از ۳ تا ۱۵۰۰، نودار را رسم کرده ایم و در زیر آن هم میانگین احتمال به ازای تمام n ها چاپ شده است که نشان میدهد با بزرگ شدن n به احتمالی در حدود ۰/۲۵ میرسیم. معنای این موضوع برای n های خیلی بزرگ، این است که احتمال اینکه سه نقطه دلخواه از دایره انتخاب کنیم و مثلث حاصل از آنها مرکزدوست باشد، ۰/۲۵ است.