Volume of a sphere using MC

December 2, 2020

1 Estimating the Volume of a Sphere using MC techniques

To estimate the volume of a sphere I employed the fairly standard hit or miss practice using MC integration. I generated 1×10^7 Monte Carlo Samples using $np.random.uniform$ from -1 to 1. In the simulation I created a sphere with radius 1 and a cube with side of length 2. Because the points inside the cube were random they had a chance to be in the sphere or not. To count the number of points inside the sphere I found the sum of the distance from the origin and counted them. This is then compared to the total number of points as a fraction. The fraction inside multiplied by the volume of the cube will equal the volume of the sphere.

To calculate pi is simply done by rearranging the formula for the volume of a sphere and solving for pi

However, to calculate the uncertainties I ran the monte carlo simulation 200 times and found the average. From this I used the known formula of the standard error to find the error in the mean

$$S_E = \frac{\sigma}{\sqrt{n_{sims}}}$$

[18]:
```
import numpy as np
import matplotlib.pyplot as plt
from scipy import random
import scipy.special


N = int(1e7) # Monte Carlo samples
R=1

all = np.random.uniform(-1,1, 3*N).reshape(N,3)
distance = np.sum(all**2,axis=1)**0.5

Ninside = np.sum(distance<=1) # number of points in sphere
fractioninside = Ninside/N # fraction of points in sphere
vol_cube = 2**3 # Volume of cube
vol_sphere = fractioninside * cube # Volume of sphere

solution = 4./3. * np.pi*R**3 #true volume using pi
diff= np.abs(vol_sphere-solution)/solution # Error
```

```
print("N=%i\tN_in=%06d\tV=%.5f" %(N,Ninside,vol_sphere))
print("True value vol_sphere =%.5f" %solution)
print("Difference = %.5f" %(np.abs(solution-vol_sphere)))
print("percentage diff = %f percent" %diff)

pi_est = (vol_sphere*3)/(4*(R**3))

print("-------")
print("True pi val: %f" %(np.pi))
print("Est Pi: %f" %(pi_est))
```

```
N=10000000      N_in=5235829    V=4.18866
True value vol_sphere =4.18879
Difference = 0.00013
percentage diff = 0.000030 percent
-------
True pi val: 3.141593
Est Pi: 3.141497
```

[33]:
```
n_sims=200
import time

times=np.zeros(n_sims)
vol_list = np.zeros(n_sims)
pi_est_list = np.zeros(n_sims)
for i in range(0,n_sims):
    t0=time.time()
    N = int(1e7) # Monte Carlo samples
    R=1

    all = np.random.uniform(-1,1, 3*N).reshape(N,3)
    distance = np.sum(all**2,axis=1)**0.5

    Ninside = np.sum(distance<=1) # number of points in sphere
    fractioninside = Ninside/N # fraction of points in sphere
    vol_cube = 2**3 # Volume of cube
    vol_sphere = fractioninside * cube # Volume of sphere
    pi_est_list[i] = (vol_sphere*3)/(4*(R**3))
    vol_list[i] = vol_sphere
    t=time.time()-t0
    times[i]=t
    if np.mod(i,20)==0: #keep track of sim
        print(i,t)


print("Total time of sim = %f" %(np.sum(times)))
```

```
0 0.634119987487793
```

```
20 0.6301989555358887
40 0.6240849494934082
60 0.6158668994903564
80 0.6180229187011719
100 0.6163930892944336
120 0.6282470226287842
140 0.6265480518341064
160 0.6127808094024658
180 0.6191859245300293
Total time of sim = 124.454432
```

[30]:
```
std_vol = np.std(vol_list)
pf_vol = np.mean(vol_list)
stdError_vol = std_vol/np.sqrt(n_sims)

std_pi = np.std(pi_est_list)
pf_pi = np.mean(pi_est_list)
stdError_pi = std_pi/np.sqrt(n_sims)



print("Volume of 3D Sphere estimate: %.5f +- %.5f" %(pf_vol, stdError_vol))
print("Pi Estimate: %.5f +- %.5f" %(pf_pi, stdError_pi))
```

```
Volume of Sphere estimate: 4.18875 +- 0.00009
Pi Estimate: 3.14156 +- 0.00007
```

[ ]:
```

```

Question 2

December 2, 2020

1 Question 2

1.1 Part 1

$$L(s,b) = \frac{(s+b)^n e^{-(s+b)}}{n!}$$

To calculate the p value for the background only i.e $s = 0$, with $n = 15, b = 2.8$ we have to use the following formula with the cumulative χ^2 distribution function.

$$p = 1 - \sum_{n=0}^{m}(n;v) = 1 - [1 - F_{\chi^2}(2\nu; n_{dof})]$$

In the case of this question $m = 14$ because we are going from $n = 15 \to \infty$ to our sum is then $n = 0 \to m = 14$. We also have $\nu = s + b = b$ and the degrees of freedom $n_{dof} = 2(m+1) = 30$. This can also be explictly verified using a chi2 applet for $F_{\chi^2}(\nu = 5.6, n_{dof} = 30)$ where $2\nu = 2b = 5.6$

$$\text{p-value} = 2.9 \times 10^{-7}$$

1.2 Part 2

For the second part we have to calculate how many standard deviations (=std) from a standard gaussian. To calculate this we use

$$\sigma = \sqrt{1-p}$$

It is easy to see that this is just the square root of the poisson function or the square root of 1-p-value. The standard normal distribution has $\mu = 0, \sigma = 1$ so we can use the ppf (percent points function or Quartile function) to return the significance. The signifiance is greater than 5 so it is a new discovery.

$$Z = 5.132042$$

1.3 Part 3

For part 3 we introduce a new function in the code which is the Likelihood ratio, simply likelihood for s over likelihood for $\hat{s}$. To get $\hat{s}$ we have to maximise $L(s,b)$ i.e $\frac{\partial L}{\partial b} = 0$. Analytically this is

$$\frac{\partial L}{\partial b} = \frac{ne^{-(s+b)}(s+b)^{n-1} - e^{-(s+b)}(s+b)^n}{n!}$$

We know that $b > 0, s > 0$ so

$$(s+b)^{n-1}(n-(s+1)) = 0$$

So either $(s+b) = 0$ or $n - (s+b) = 0$ and $\hat{s} = n - b$. This is verified graphically (desmos.com) and introduced as a function for which the likelihood ratio depends on. In the code below I plotted $-2\ln\lambda(\hat{b}, n)$ vs s like in the hint and used *scipy* to find the confidence intervals matching the roots of the plot for when $-2\ln\lambda = 1$.

1.4 Part 4

Part 4 was particularly tricky but the hint and reference helped. I now introduced a new function for the likelihood ratio using a b as a uniformly distributed random number between $[b - \sigma_b], [b + \sigma_b]$. Note that a 1 σ confidence interval is 68.3% for a standard gaussian distributed. So to find the CI I find the ratio of the likelihood function which is less than or equal to 1 by introducing another function which finds the amount under 1 and then finding fraction of them. This fraction was plotted and shows that it has very similar roots to part 3 with very similar confidence interval. I am not sure if I avoided undercoverage or not with this method.

1.5 Code

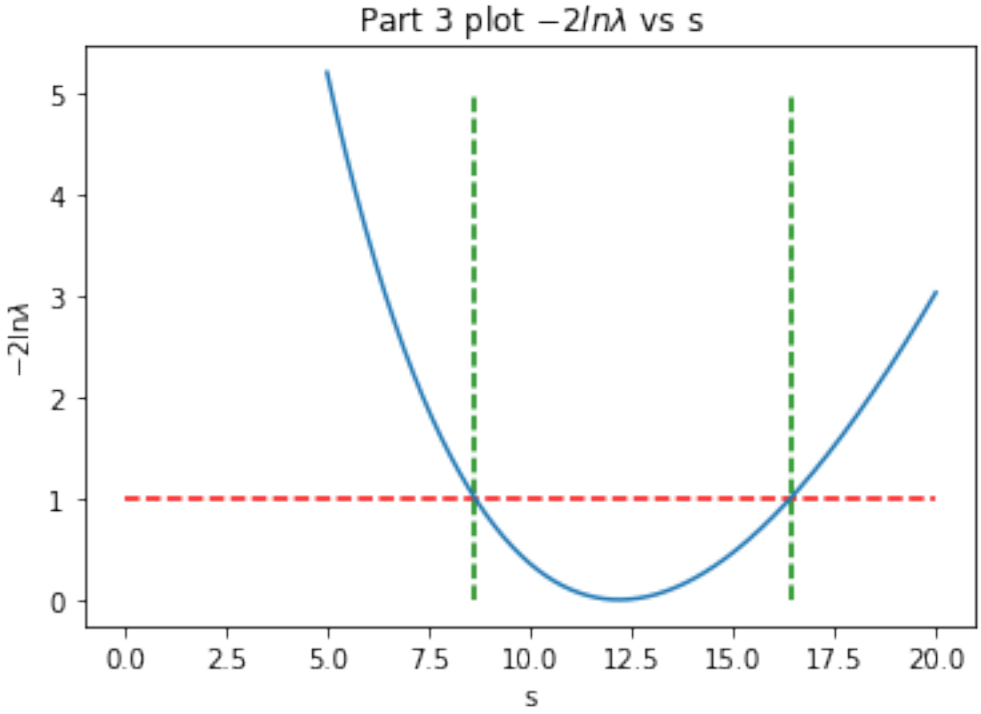
[64]:
```
import matplotlib.pyplot as plt
import numpy as np
import scipy.optimize
import scipy.stats

#functions
def likelihood(s, b, n):
    return ((s+b)**n * np.exp(-(s+b))) / np.math.factorial(n)


def poisson_func(m, v):
    n_dof = 2*(m+1)
    return 1 - scipy.stats.chi2.cdf(2*v, n_dof)


def s_hat(b, n):
    return n - b


def likelihood_ratio(s, b, n):
    return likelihood(s, b, n) / likelihood(s_hat(b, n), b, n)


def likelihood_ratio_b_uniform(s, b, n, b_hat):
    return likelihood(s, b_hat, n) / likelihood(s_hat(b, n), b_hat, n)


def likelihood_ratio_under_limit(s, b, n,sigma_b): #n_b = 10000, limit=1
    b_arr = np.random.uniform(b-sigma_b,b+sigma_b,10000)
    frac_sat_con = np.sum(-2*np.log(likelihood_ratio_b_uniform(s, b, n, b_arr))␣
 ↪<= 1) / 10000
```

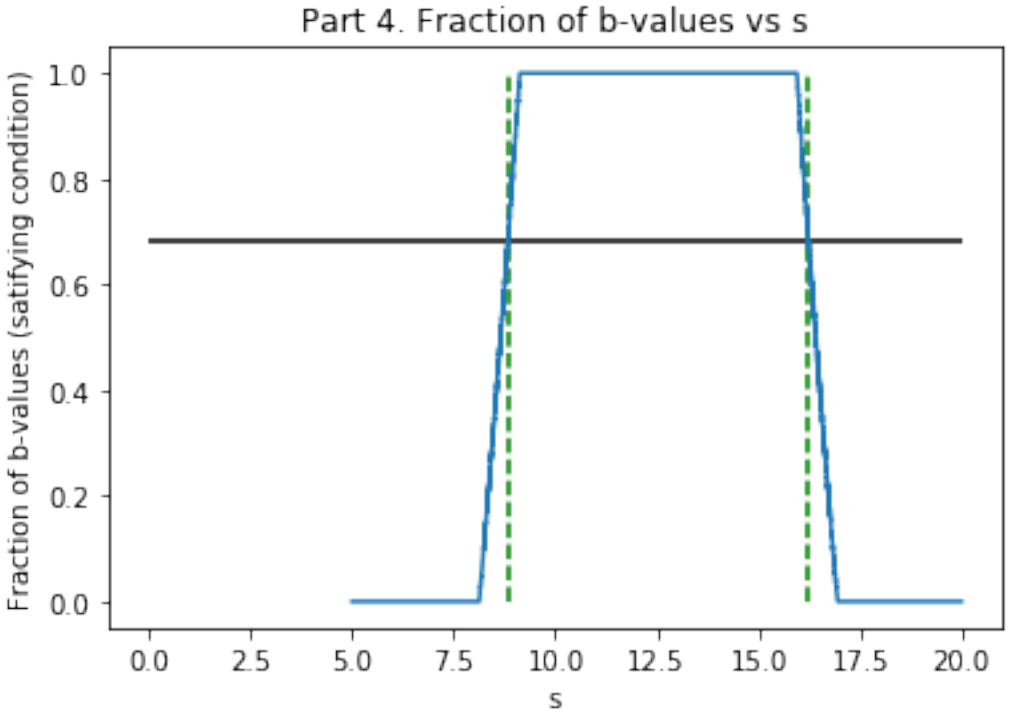
```
    return frac_sat_con


#init
b=2.8
n_obs = 15
m = n_obs-1

s=0
v = s+b
Poi= poisson_func(m, v)
p=1-Poi
print("p-value:", p)

std = np.sqrt(Poi)
n_sigma = scipy.stats.norm.ppf(std)
print("P-value corresponds to %f standard deviations" %n_sigma)


plt.figure(1)

s_arr = np.linspace(5,20, 10000)
plt.plot(s_arr, -2*np.log(likelihood_ratio(s_arr, b, n_obs)))
plt.title("Part 3 plot $-2ln \lambda$ vs s")
plt.xlabel("s")
plt.ylabel(r"$-2\ln \lambda$")


sol1_min = scipy.optimize.minimize_scalar(lambda s: -2*np.
 ↪log(likelihood_ratio(s, b, n_obs)), bracket=show_range)
print("sol1 x = %.5f" %sol1_min.x)
#print(sol1_min)

sol1_lower = scipy.optimize.root_scalar(lambda s: -2*np.log(likelihood_ratio(s,␣
 ↪b, n_obs)) - 1, bracket=[show_range[0], sol_min.x])
sol1_upper = scipy.optimize.root_scalar(lambda s: -2*np.log(likelihood_ratio(s,␣
 ↪b, n_obs)) - 1, bracket=[sol_min.x, show_range[1]])
#print(sol1_lower)
#print(sol1_upper)

plt.hlines(1,0,20,  colors='Red', linestyle='dashed')
plt.vlines(sol_lower.root, 0, 5, colors='Green', linestyle='dashed')
plt.vlines(sol_upper.root, 0, 5, colors='Green', linestyle='dashed')
print("-------------------------------------------------")

print("s for part 3 with +- uncertainties from scipy")
```

```
print("s = %.3f + %.3f, - %.3f" %(s_hat(b, n_obs),sol_upper.root - s_hat(b,␣
 ↪n_obs),s_hat(b, n_obs) - sol_lower.root))



sigma_b = 0.5



frac_arr = np.zeros_like(s_arr)
for i in range(fracs.size):
    frac_arr[i] = likelihood_ratio_under_limit(s_arr[i], b, n_obs, sigma_b)
plt.figure(2)
plt.plot(s_arr, frac_arr)
plt.title("Part 4. Fraction of b-values vs s")
plt.xlabel("s")
plt. ylabel("Fraction of b-values (satifying condition)")

frac_limit = scipy.stats.norm.cdf(1) - scipy.stats.norm.cdf(-1)
plt.hlines(frac_limit, 0,20)

sol2_lower = scipy.optimize.root_scalar(
    lambda s: -2*np.log(likelihood_ratio_under_limit(s, b, n_obs, sigma_b)) -␣
 ↪frac_limit,bracket=[show_range[0], sol_min.x])
sol2_upper = scipy.optimize.root_scalar(
    lambda s: -2*np.log(likelihood_ratio_under_limit(s, b, n_obs, sigma_b)) -␣
 ↪frac_limit,bracket=[sol_min.x, show_range[1]])
#print(sol2_lower)
#print(sol2_upper)

plt.vlines(sol2_lower.root,0,1, colors='Green', linestyle='dashed')
plt.vlines(sol2_upper.root,0,1, colors='Green', linestyle='dashed')
print("-------------------------------------------------")
print("s for part 4 with +- uncertainties from scipy")

print("s = %.3f + %.3f, - %.3f" %(s_hat(b, n_obs),sol2_upper.root - s_hat(b,␣
 ↪n_obs),s_hat(b, n_obs) - sol2_lower.root))
```

```
p-value: 2.866153162583984e-07
P-value corresponds to 5.132042 standard deviations
sol1 x = 12.20000
-------------------------------------------------
s for part 3 with +- uncertainties from scipy
s = 12.200 + 4.213, - 3.547
-------------------------------------------------
s for part 4 with +- uncertainties from scipy
s = 12.200 + 4.010, - 3.343
```

```
/usr/local/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:95:
RuntimeWarning: divide by zero encountered in log
/usr/local/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:97:
RuntimeWarning: divide by zero encountered in log
```




Part 4. Fraction of b-values vs s
Fraction of b-values (satifying condition)
1.0
0.8
0.6
0.4
0.2
0.0
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
s