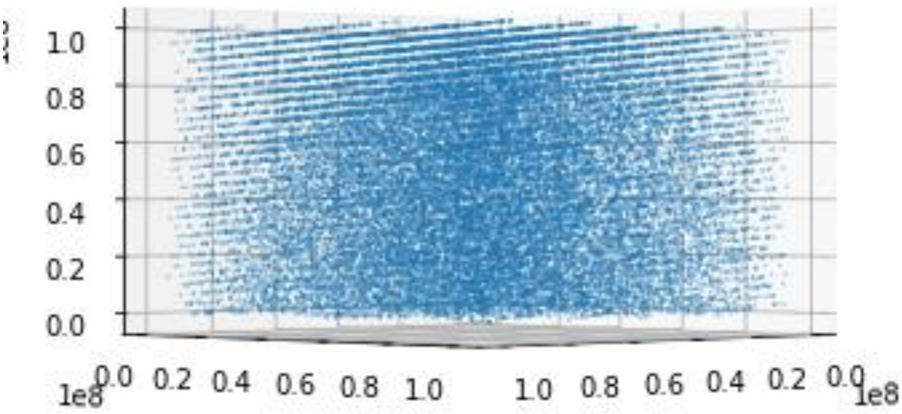
Assignment 6

Yifei Gu

260906596

Q1)

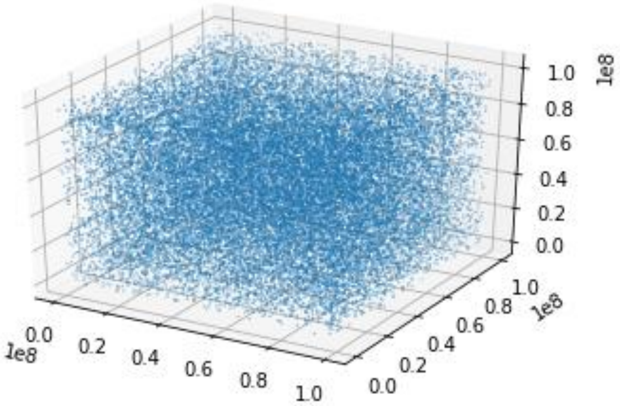
Here are the random points I loaded from 'rand_points.txt' which was generated by the broken lib. When rotate the 3D plot, I saw an obvious pattern of these random points.



There are 30253 points in the 'rand_points.txt'. So, I generate same number of random points with the python generator. Here is how I did it:

```
# random points from python generator
X = []
Y = []
Z = []

for i in range(30253):
    x = random.randrange(0,10**8)
    y = random.randrange(0,10**8)
    z = random.randrange(0,10**8)
    X.append(x)
    Y.append(y)
    Z.append(z)
fig2 = plt.figure()
ax = fig2.add_subplot(111, projection='3d')
ax.scatter(X, Y, Z, marker='o',s=0.1)
plt.savefig('python_rand_points_generator.png')
plt.show()
```

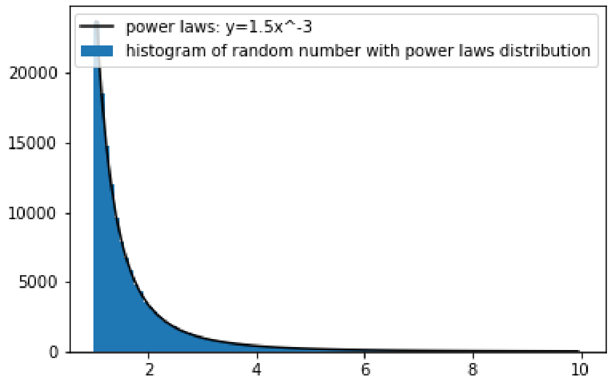


In the 3D plot of the random number generated by python, I don't see any pattern when rotating the plot. I couldn't get *libc.dylib* work on my windows system.

Q2)

I chose **power laws** for the bounding distribution. Because it is close to the shape of the exponential distribution so it's more efficient to do the transformation. Specifically, I used ***y= 1.5x^-3***. The 1.5 scale factor is to keep the power law distribution above the exponential distribution. Here are the **100000** random numbers with a power laws distribution I generate and the function I wrote:

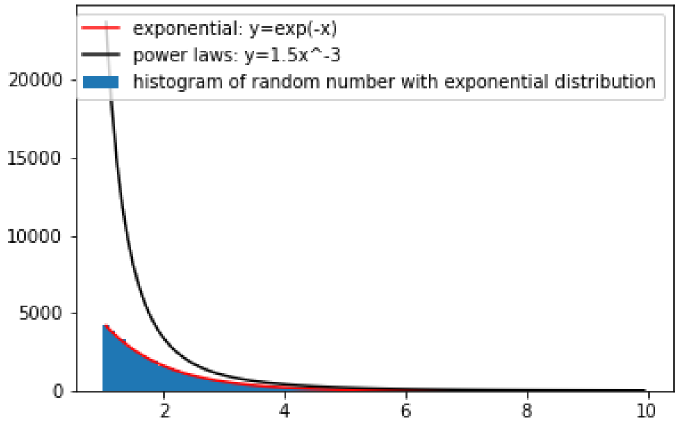
```
def random_power_law(n,a):
    r=np.random.rand(n)
    alpha = a
    x = (1-r)**(1/(1+alpha))
    return x
```



Here is the function I wrote for the rejection method, which return the exponential distribution and the accepted fraction from the power laws distribution.

```
def pl_to_exp(power_law):
    accpt_prob=1/1.5*np.exp(-power_law)/(power_law**-3)
    assert(np.max(accpt_prob<=1))
    accept=np.random.rand(len(accpt_prob))<accpt_prob
    exponential=power_law[accept]
    accpt_fraction=len(exponential)/len(power_law)
    return exponential,accpt_fraction
```

The random number with an exponential distribution that transferred from the power laws distribution is shown in the histogram below. The histogram is **matching up the expected exponential curve** in red color. However, there are only 49153 numbers accepted among the 100000 points from the power laws distribution, which is **an accept fraction of 49.153%**. It doesn't seem very efficient.



Comparing with the rejection method, **the transformation method takes 0.0039secs, which is 10 times faster than the rejection method.** The generator with transformation method is below:

```
def random_exp(tau,n):
    r=np.random.rand(n)
    exponential = -tau*np.log(1-r)
    return exponential
```

The results of efficiency I got:

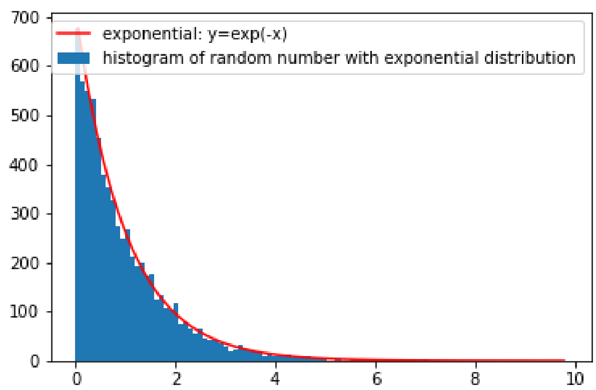
```
To generate 49153 random number,
The rejection method takes: 0.023139476776123047
and the accept fraction is 0.49153
The transformation method takes:
0.003988027572631836
```

Q3)

I tried a series of limits on v from 0 to 1 in a space of 0.5. Keep the distribution matching up with the expected exponential curve and the highest accept fraction, the limits I can get is 0.7. Here is how I get the exponential distribution with a ratio-of-uniforms generator.

```
n = 10000
v = np.random.rand(n)*0.7
u = np.random.rand(n)
ratio = v/u
accept = u<np.sqrt(np.exp(-1*ratio))
exponential = ratio[accept]
```

The distribution is matching up with the expected curve:



The efficiency (accept fraction) is 0.7028 and it takes about 0.000996 secs to generate 7028 random number.

```
The accept fraction is 0.7028
To generate 7028.0 random number with exponential
distribution, a ratio-of-uniforms generator takes:
0.000995635986328125 second
```