Here is the report of PHYS512 problemset 6 written by Jiaxing MA.

problem1

I use 3D plot to show the (x,y,z) triples, here is the code I use:

```
fig1 = plt.figure()
ax1 = fig1.add_subplot(111, projection='3d')


# load some test data for demonstration and plot a wireframe
datas =np.loadtxt('rand_points.txt')
X, Y, Z = datas[:,0], datas[:,1], datas[:,2]
ax1.scatter3D(X,Y,Z)

# rotate the axes and update
for angle in range(0, 360,5):
    ax1.view_init(45, angle)
    plt.draw()
    plt.pause(.001)
```

Figure 1

By rotating the figure, I found at certain angle, these triples are in same planes. Here is the figure:

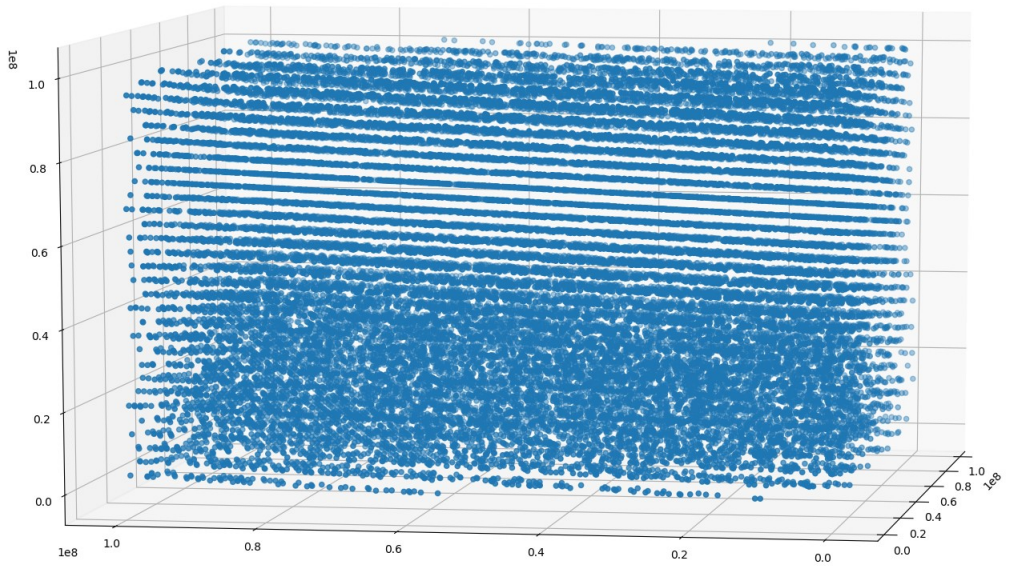

Figure 2

Then I generate random triples using the generator from the python, I couldn't find the triples in the same planes, here is the code and image:

```
#  Same proecess with python random number genetrator
A = []
B = []
C = []
for i in range(20000):

    x = random.randrange(0,10**8)
    y = random.randrange(0,10**8)
    z = random.randrange(0,10**8)
    A.append(x)
    B.append(y)
    C.append(z)
fig2 = plt.figure()
ax2 = fig2.add_subplot(111, projection='3d')
ax2.scatter3D(A,B,C)
```

Figure 3

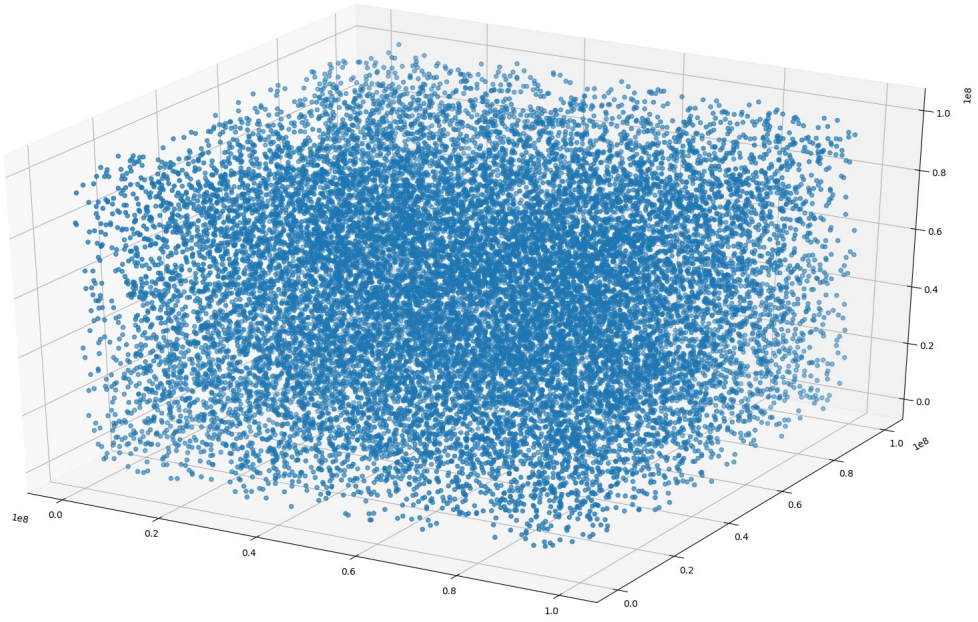

Figure 4

I cannot figure out libc.so for window.

problem2

I used power laws as bounding function, I choose to use $y = x^{-3}$ to generate random exponential deviate $y = e^{-x}$ Here is the code I generate random power law,

```
def random_power(a=-3):
    r=np.random.rand(100000)
    alpha = a
    x = (1-r)**(1/(1+alpha))
    return x
```

Figure 5

I adjust the ratio of the power law function to make the power law larger than exponential function, here is the figure.

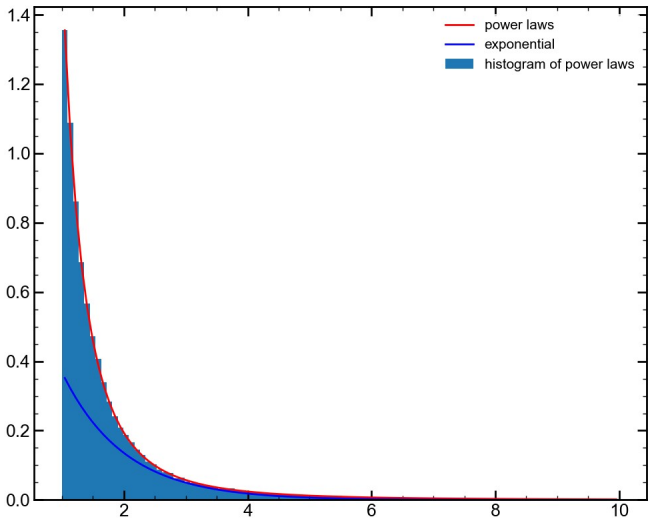

Figure 6: The histogram is random power law function, the red curve is the function $y = 1.355x^{-3}$, the blue curve is the function $y = e^{-x}$.

Here is the code that I used to generate exponential from power law function.

```
def expfrompower(x):
    accept_prob=(1/1.355)*np.exp(-x)/(x**-3)
    print(np.max(accept_prob))
    assert(np.max(accept_prob<=1))
    accept=np.random.rand(len(accept_prob))<accept_prob
    return x[accept]
```

Figure 7

Here is the result:

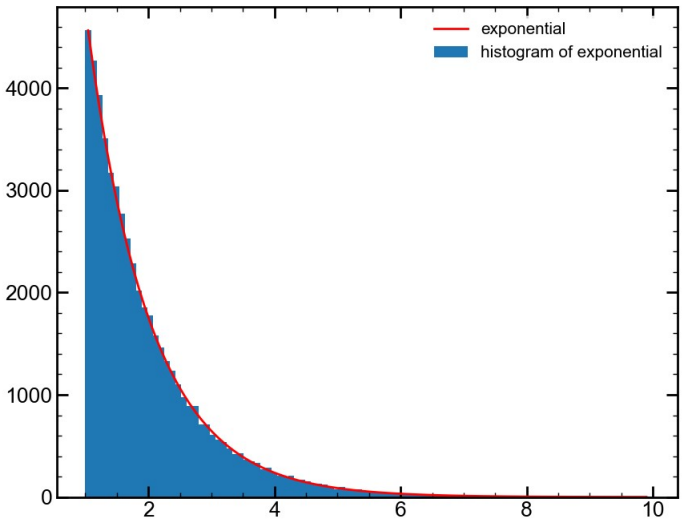

Figure 8

To calculate the efficiency of the generator, I try to change the constant in the expfrompower function, and I find that with the constant 1/1.355, the maximum accept probability is slightly smaller than 1, and the efficiency is the largest I could find, which is 0.542 accept rate. I also compare the speed of rejection method and transformation method to generate random exponential function, the rejection method is much slower. How is the code I used to calculate how many uniform deviates that transform to exponential deviates. I divided the lens of exponential deviates array to the len of uniform deviates.

```
e = expfrompower(x)
print('accept fraction is', len(e)/len(x))
```

Figure 9

Here is the result I got:

```
accept fraction is 0.54186
Time from rejection method: 0.01795363426208496
Time from transformation method: 0.0009953975677490234
```

Figure 10

problem3

To generate the exponential deviates using ratio-of-uniform method, I choose the limits on v from 0 to 1, and after optimize the efficiency, I changed it to from 0 to 0.75. The efficiency I got is 0.6667. Here is the ratio-of-uniform code I got.

```
n = 1000000
u = np.random.rand(n)
v = np.random.rand(n)*0.75
rat = v/u
accept = u<np.sqrt(np.exp(-1*rat))
myexp = rat[accept]
```

Figure 11

Here is the code that i calculate the efficiency

```
print("Accept fraction is ", np.mean(accept))
```

Figure 12

Here is the result I got: the histogram and the efficiency.

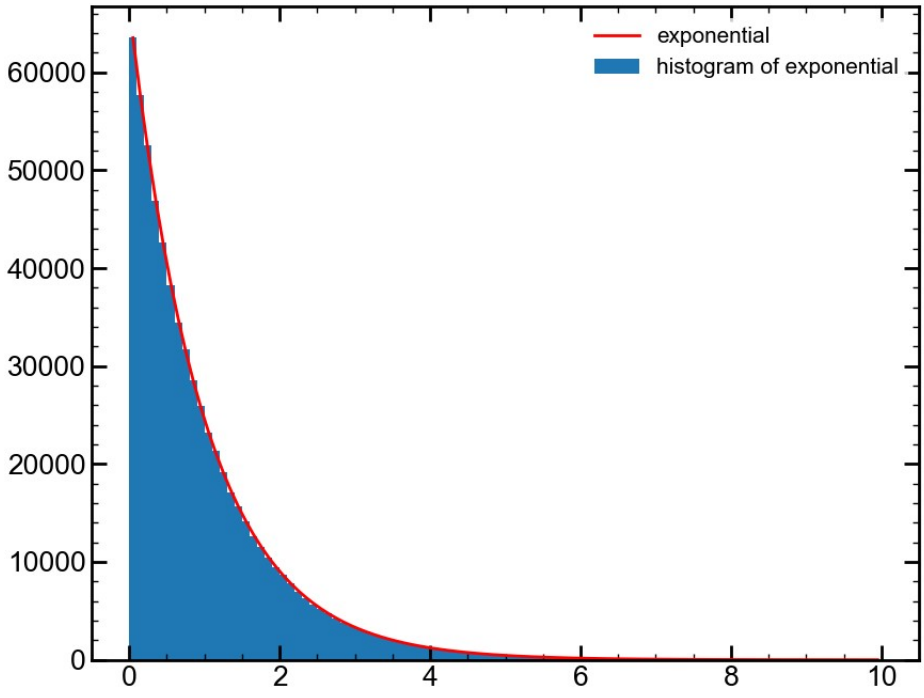

Figure 13

Figure 14