Phys 512 Problem Set 7

Jan Viatteau (260857910)

1. For the C generated points, we just read in the prepared data and check by moving the 3D scatter plot of the generated points. We do see planes appearing for a certain viewing angle. To make this more apparent (ie. to see all the planes simultaneously), we check some points on a single plane to determine the a and b values. For example, if we read the coordinates for two points (x_1, y_1, z_1) and (x_2, y_2, z_2) while keeping $x_1 \sim x_2$, we get:

$$z_1 = ax_1 + by_1$$
$$z_2 = ax_2 + by_2$$
$$b \sim \frac{z_1 - z_2}{y_1 - y_2}$$

and a similar method for getting a. We then plot z versus $ax + by$ for each point, and end up with the figure below.

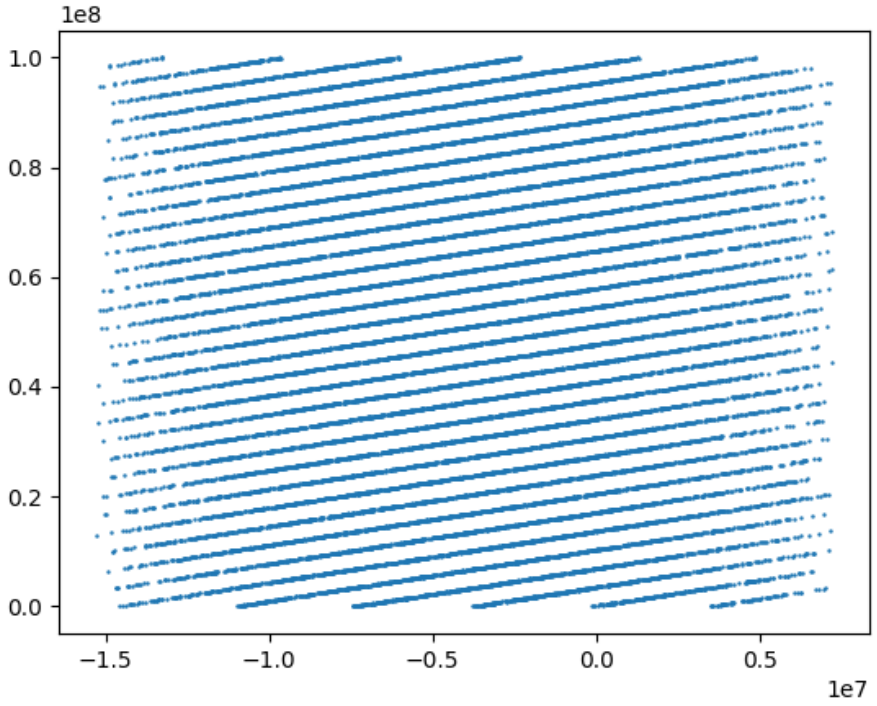

Figure 1: $ax + by$ vs z for $a = -0.153188$ and $b = 0.073168$ with the points from `rand_points.txt`. Clearly, even though the a and b values are not perfectly right, there is correlation between the coordinates generated for each point.

For the python generated random points, we just for-loop through generating points using the standard `random` library. Then, we do a 3D scatter plot of the results and get the figure below.

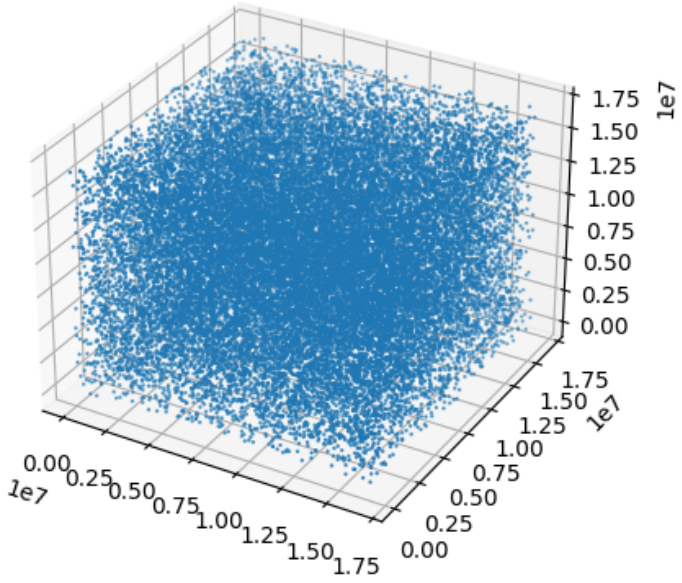

Figure 2: 3D scatter plot of randomly generated points using the python standard library. Moving it around, I can't find any planes or shape, so there's no evidence of correlation between randomly generated coordinates.

When I tried to run `test_broken_libc.py` with the standard Windows library, but with the library I specified my computer crashed. So instead, I made a friend run it on his Linux machine with libc.so as the library and send me back the output. The result is showed in the figure below.

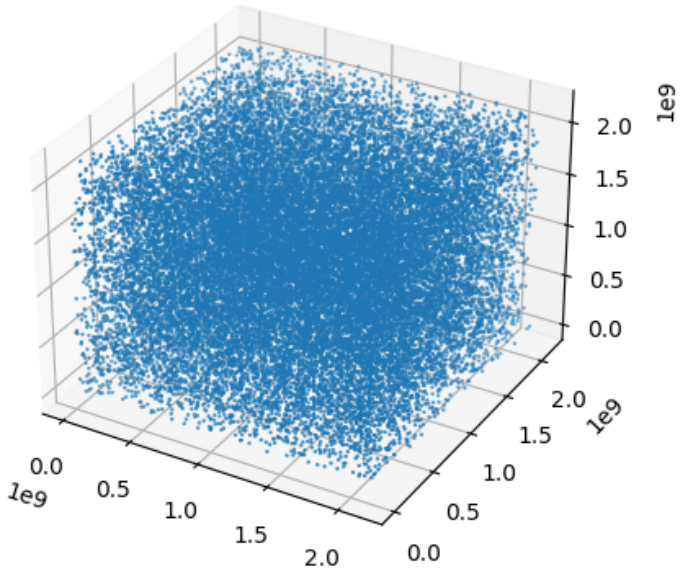

Figure 3: 3D scatter plot of randomly generated points using the Linux standard library. As for the python one, moving around didn't show any planes or obvious patterns.

2. For rejection, we want the used distribution to be greater than the distribution of interest over the whole interval of interest. Since we want to generate exponential deviates following a e^{-x} distribution, Gaussians are out of the question. Because Gaussians follow more or less e^{-x^2}, it dies off quicker than our exponential, so it is bound to get under our exponential at some point. Power laws $x^{-\alpha}$ look a bit like our exponential given $\alpha > 0$, so they are a possible choice. However, since they are not defined at 0, we can avoid this problem by choosing a Lorentzian (actually just the positive half of a Lorentzian centered at 0)! Lorentzians are of the form $\frac{1}{1+x^2}$, so they keep above exponentials (nothing dies off quicker than exponentials except other exponentials). The used Lorentzian and target exponential are shown in the figure

below.

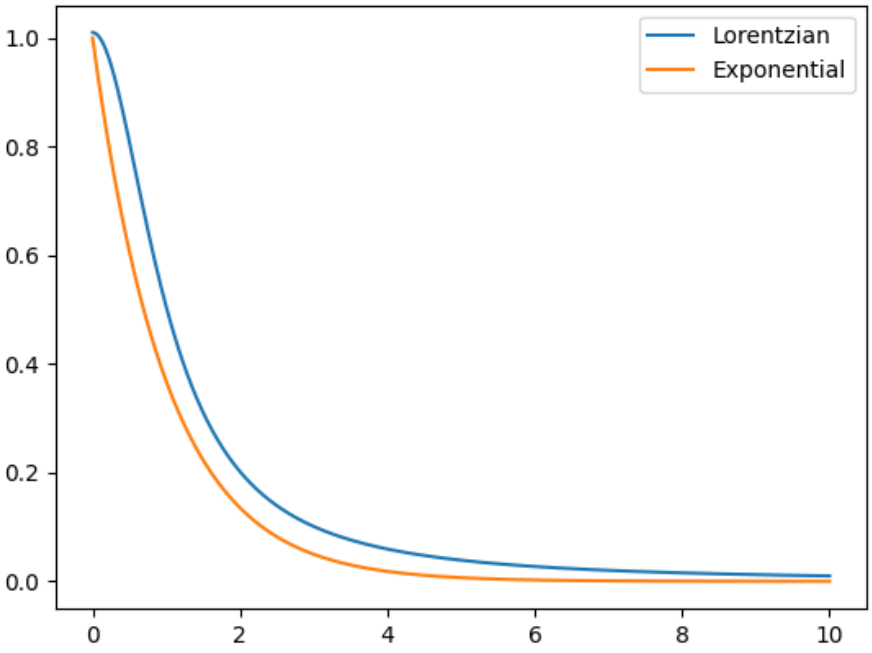

Figure 4: Lorentzian and target exponential. As can be seen, over the whole range, the exponential lays below the Lorentzian, and we know analytically that it stays that way to ∞.

So, we generate Lorentzian deviates since we know how to do that : from the slides, the CDF of a Lorentzian distribution is $\arctan(x)/\pi + 1/2$, therefore taking $x = \tan(\pi(y - 0.5))$ where y is uniformely distributed will give us a Lorentzian distribution. We take another uniformly distributed random number between 0 and 1, and compare that to the ratio of the exponential at x to the Lorentzian at x. We accept if the random number is lower than the ratio, and reject otherwise. That way, we "take off the top" of our Lorentzian to get the exponential distribution. We make a histogram of the accepted values, and end up with the figure below. The fraction of accepted samples using this

method is `0.630174` as returned by our code (and we use two uniform deviates for each random potential exponential deviate, so ultimately we need on average around $2/0.63 \approx 3.17$ uniform deviates to produce an exponential deviate). I tried lowering the coefficient of the Lorentzian so that it lies as close to the exponential as possible, but didn't get a significant improvement in efficiency.

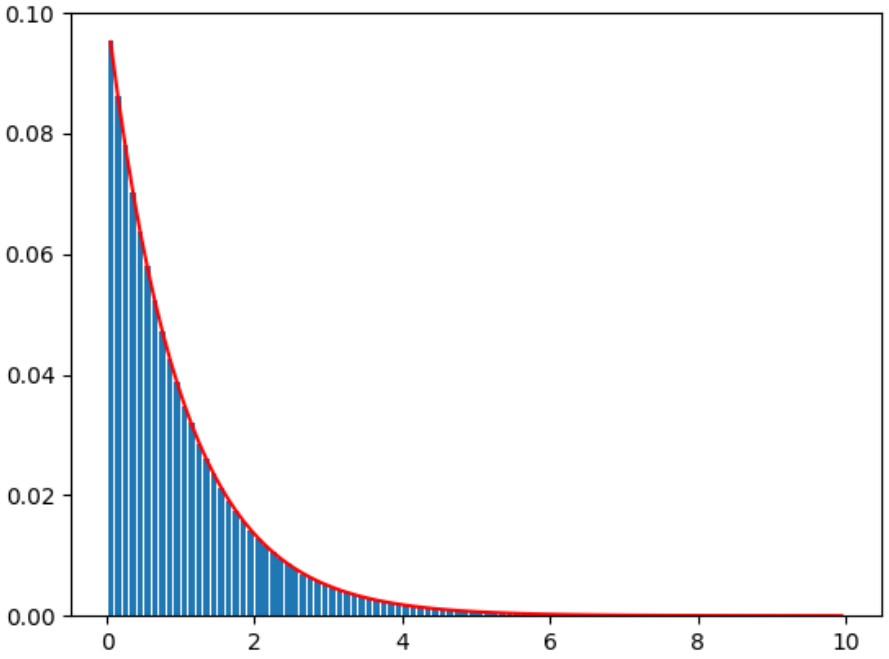

Figure 5: Histogram of the obtained sample and exponential prediction that we aimed for. As can be seen, the sample follows the exponential distribution quite closely.

3. For ratio of uniforms, we first determine the region we accept samples from. To do that, we plot v as a function of the maximum u we allow. Since we're generating exponential deviates, we accept $0 < u < \sqrt{e^{-v/u}}$ (since we don't want negative values, cut off both u and v at 0), which

gives the border of our sample region :

$$\begin{aligned} u &= \sqrt{e^{-v/u}} \\ &= e^{-v/2u} \\ v &= -2u\log(u) \end{aligned}$$

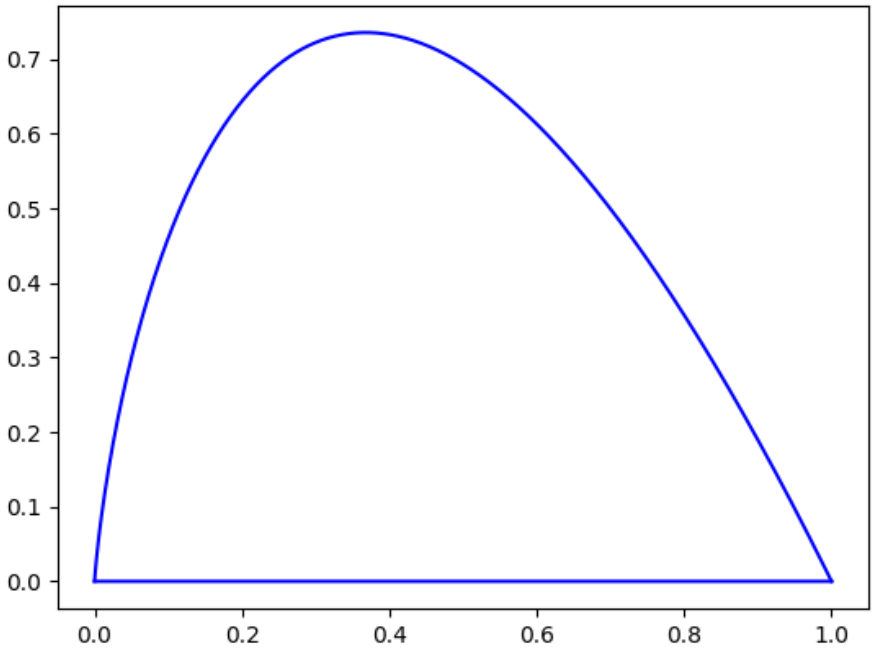

Figure 6: The region we accept samples from, u is represented on the horizontal axis, v on the vertical. Since we cut off our deviates at 0, we don't sample negative values. The maximum value for v in the sample is just below 0.75, we will take that into account when generating the v random values.

We then draw u and v values uniformly, between 0 and 1 for u and between 0 and 0.75 for v (to maximize the chances of falling into the accepted region). We reject the values of v/u that fall outside the region, and take $e^{-v/u}$ for the accepted ones. We histogram the result and end up with the plot below.

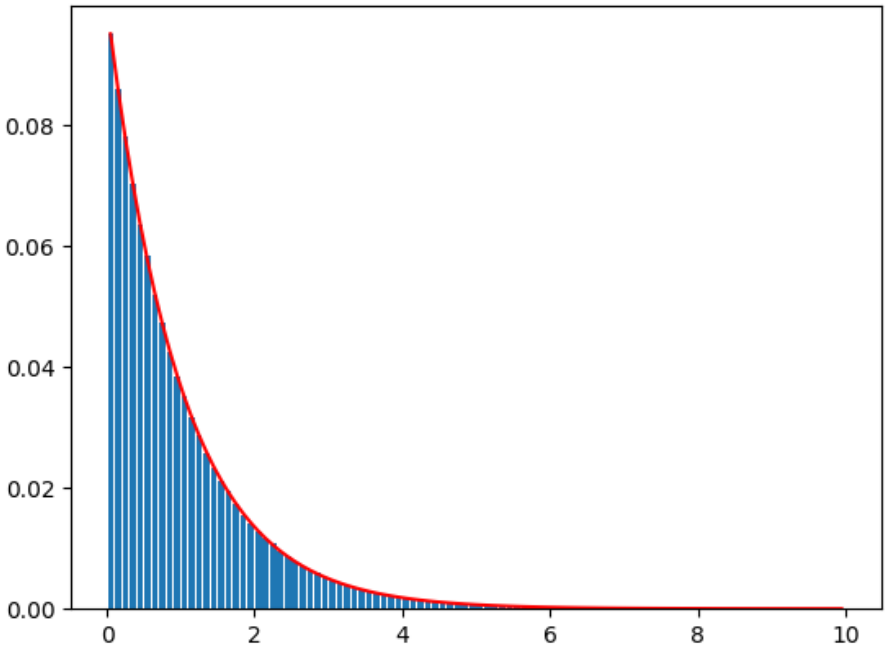

Figure 7: Histogram of the sample obtained through ROU and exponential prediction that we aimed for. Here as well, the sample comes very close to the prediction.

The accepted fraction here is `0.667246`. This is slightly better than what we had for the rejection method, but ultimately the best point of the ratio of uniforms method is its simplicity.