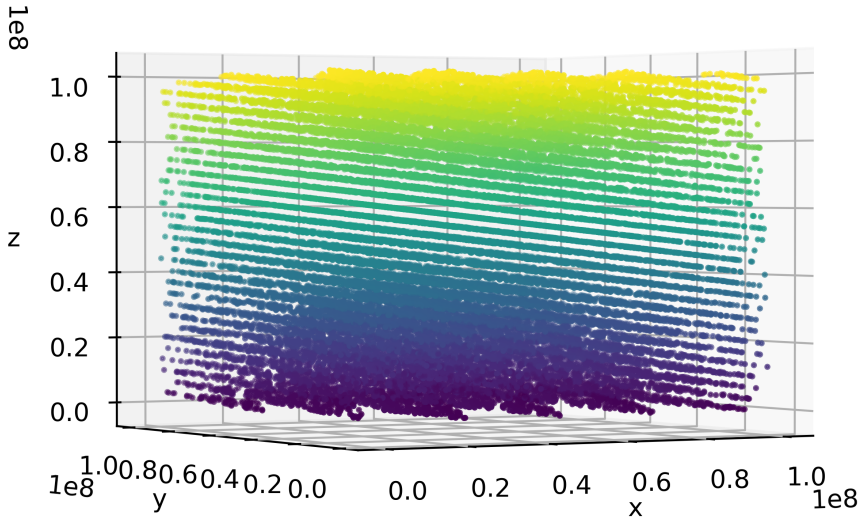
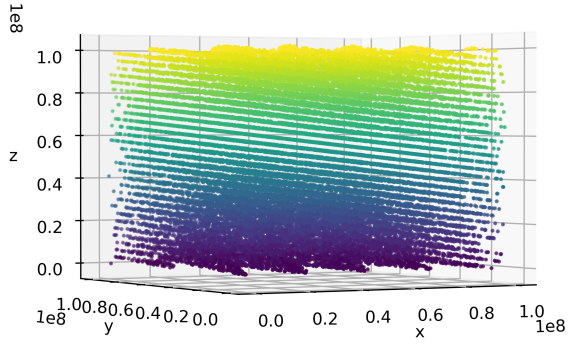
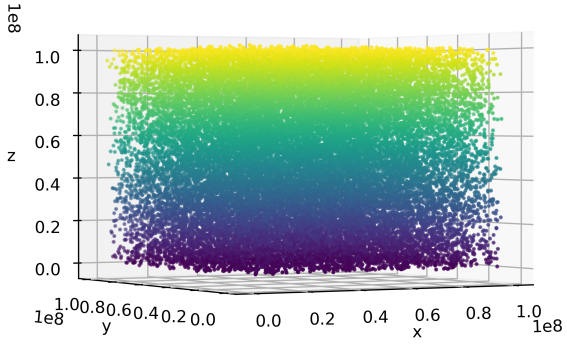
1. We can simply import the data from the "rand_points.txt" file and make a 3D plot in pyplot to see whether we observe planes or not. The same can be done for np.random.randint as well to check if we observe the same planes. The viewing angle of the 3D plot was adjusted to best show the planes, however as the value of a and b for $ax + by = z$ are not the same for each plane, a few planes merge together making it hard to discern a difference, as can be seen below. The viewing angle was kept the same for the numpy random numbers and for the default random number generator in the C standard library. We also make another plot for random numbers generated using this C standard library on my MacBook. No major changes needed to be applied to the script besides changing the name of the output file in "test_broken_libc.py". All the plots were made in the file "problem_1.py". The plots can be seen below.



(a) Random numbers from "rand_points.txt" (C standard library).



(b) Random numbers from C standard library produced on my MacBook.



(c) Random numbers generated using np.random.randint.

Figure 1: 3D plot of random numbers generated using different methods to see if we can make out any patterns.

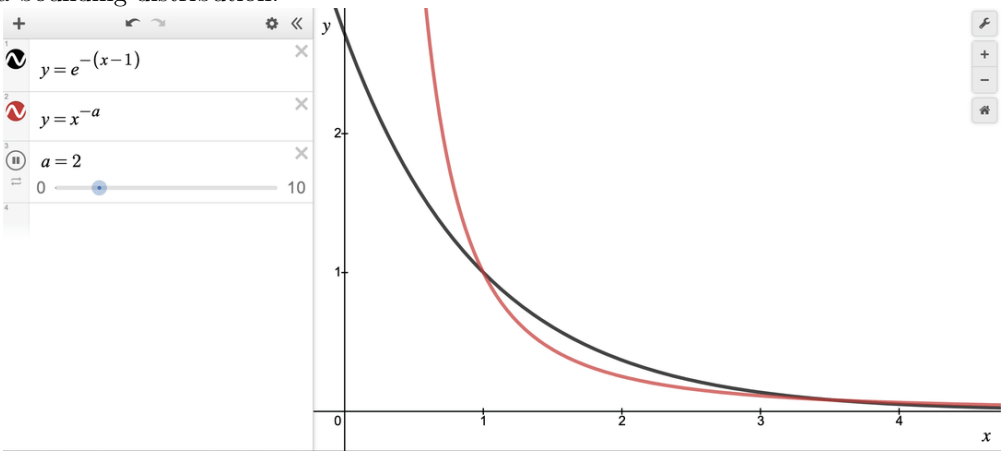
From Figure 1a, we count the number of observable planes to be exactly 30. This is the same for the random numbers generated using the same library on my MacBook. However, the numpy generated random numbers in Figure 1c do not have any observable planes, indicating that this is a better random number generator.

2. First, we shall look at how to randomly generate numbers for each distribution. We do so by calculating the inverse of the CDF for each distribution. Starting with the normalized Lorentzian, i.e. $1/\pi(1+x^2)$, and taking x to be our uniform generated random numbers, the CDF would be the integral of this from 0 up to y which is $\frac{1}{\pi}\arctan y$. Hence, we could generate random numbers by calculating the inverse CDF which gives us $\tan \pi x$. In order to better cover the range of tan and avoid the asymptotes we can substitute πx with $\pi(x-0.5)$ so that the input varies between $[-\pi/2, \pi/2]$. Therefore, our Lorentzian deviates can be generated as $\tan \pi(x-0.5)$. We can generate Gaussian deviates using the Box-Muller method with two random uniform deviates, θ_1 and θ_2. As we have already discussed this method in depth, we just state the result and generate the random numbers as $x = \sqrt{-2\ln\theta_1}\sin(2\pi\theta_2)$. Lastly, we can generate power law deviates, where α is our exponent, by integrating the normalised power law, $Cx^{-\alpha}$ (where $\alpha > 1$), from 0 to y to give $\frac{C}{1-\alpha}y^{1-\alpha}$. Note that C can only exist if we define our power law to exist where $x \geq x_{min}$ and $x_{min} \neq 0, -\infty, \infty$. Taking the inverse CDF, we can generate random numbers as $x_{min}(1-x)^{1/(1-\alpha)}$ where we have taken $x \to 1-x$ so that we do not have to worry about any negatives.

Now we decide which distributions we could use for the bounding distributions. The Lorentzian is an obvious candidate as it provides us with numbers inside our range for the exponential distribution (0 to ∞) and is always larger than the exponential. We can see this by writing out the Taylor series for our exponential: $e^{-x} = 1 - x + x^2/2 - x^3/6 + ...$ whereby if our random numbers are always between 0 and 1 (which they are), the leading order non-constant decay term (negative) is x, where as the leading order decay term for the Lorentzian is x^2. When x is between 0 and 1, $x^2 < x$ and therefore, the Lorentzian decays slower.

The Gaussian distribution is possible, however to be a bounding distribution it must always be greater than the distribution desired. At large x, the Gaussian will always be lower than the exponential distribution, regardless of the coefficient, and we can see this by calculating the point of intersection for a Gaussian: $Ae^{-x^2/2}$. Doing some algebra, we find that the point of intersection (at $x > 0$) is $x = 1 + \sqrt{1 + 2\ln A}$. As a result, the Gaussian cannot be used as a bounding distribution.

Lastly, we consider the power distribution. With suitable chosen α and x_{min}, the power distribution is always greater than the exponential. However, in order to accurately represent our exponential distribution, we would like to have $x_{min} = 0$ which we cannot have. Furthermore, only at $x_{min} \approx 0.9$ is the power distribution always greater than the exponential, so we cannot make x_{min} very small. This means that we cannot use the power distribution either. Therefore, lets consider the power distribution for another exponential centered at 1 with $x_{min} = 1$. In order for the power distribution to always be greater than the exponential in this case, $alpha = 1$, which was checked doing a quick plot on Desmos. This cannot be used as $\alpha > 1$, therefore we choose $\alpha = 1.1$ and just graph for the sake of analysis even though it cannot be used as a bounding distribution.



The method of rejections is explained in the python code in "problem_2.py". To quickly summarise, the method involves generating a uniformly distributed random number and if this number is less than the acceptance probability, characterised by dividing the exponential distribution by the respective probability distribution for the random numbers generated from that distribution, then we accept that number. In this way we can also calculate the ratio between the number of accepted numbers and the number of uniform deviates used to generate our distributions to see how efficient our generators are. Doing so we can also play around with the probabilities (by multiplying by some factor) to see how efficient we can make the generator. Obviously, we can achieve an acceptance of 1 if we multiply by a large enough factor, yet this makes our resulting distribution inaccurate.

3.

`output.png`

Figure 2: Python output for "problem_1.py".