ECE:226 Project

Anis Chihoub, RUID: 192002438

Due: July 7th 2020

1 Background

Often in statistics and mathematics, it is important to use pseudo-random sampling methods in order to transform data to fit a specific distribution. Although there are many different methods to do so, it is important to understand how to do so efficiently and how to apply the proper method to a particular problem.

One particular example that can be discussed is Monte-Carlo simulation. Often, these processes require large amounts of random numbers. In order to generate random-numbers, it takes a massive amount of computational power. Therefore, methods that employ statistical transformations are needed to generate these numbers quickly. As a result, this has lead to numerous innovations, such as simulations[1] of radio flares using pseudo random number generation to develop radio signals.

It is important to note that the same sampling methods can be applied to the study of probability distributions. Often, it is important to transform various types of data to reveal their properties. Take for example a uniformly distributed density function. As a whole, not much can be revealed about the data. However, by employing a variety of transform techniques, one can accurately transform the data in such a way that more can be learned, such as the shape or the end behavior.

There are multiple methods to transform data such as Rejection Sampling and Histogram based Sampling. However, this report will discuss the use of Inverse Transform Sampling method. Inverse transform sampling was selected due to its memory-less properties, saving on physical memory space needed to allocate data, and its relative straight forwardness to implement in this situation.

Inverse Transform Sampling works by finding the inverse of the CDF function (the quantile function) and applies it on each element in the uniform random variable. Mathematically, this method works by plugging in a value from the uniform random variable into the inverse of the CDF, the corresponding x value in the CDF of the desired distribution is generated. Obviously, since it is possible to generate equally likely values from a uniform random variable, one can guarantee that the numbers in this transformed sample are random. Additionally, since each sample is transformed independently from the other, this gives the method its memory-less property and ensures that one sample does not impact the other. Therefore, Inverse Transform sampling allows for a relatively quick and simple way to transform data.

[1]https://arxiv.org/abs/1707.02212

2 Laplace Random Variable

2.1 Introduction

In order to explore the process of Inverse Transform Sampling, it is necessary to discuss the target variable. In this case, the target distribution is a laplacian variable with $Var[X] = \frac{2}{9}$ and $E[X] = 0$. It is also important to look at the nature of its distribution. The PDF is clearly two sided, which will come into play when the CDF is derived later on.

2.2 Scale Derivation

In order to derive the CDF, first we need to define the scale, b. We can set up the following to define the scale[2]:

$$Var[X] = 2b^2 \tag{1}$$

Solving for the scale, we find that $b = \frac{1}{3}$.

2.3 CDF Derivation

The PDF of the Laplacian Random Variable is defined as follows [3]: [4]

$$f_X(x) = \frac{e^{\frac{-|x-\mu|}{b}}}{2b} \tag{2}$$

Where b is the scale variable and μ the expected value. To find the CDF, we can set up an integral to do so and split it appropriately. This first derivation will be for all values $S = \{x \mid x \leq \mu\}$

$$F_X(x) = \int_{-\infty}^{x} \frac{e^{\frac{-(t-\mu)}{b}}}{2b} dt \tag{3}$$

Integrating yields:

$$\frac{e^{\frac{(x-\mu)}{b}}}{2} \tag{4}$$

Now, to account for any values greater than the expected value, we can set up the following integral and subtract 1:

$$F_X(x) = \int_{\mu}^{x} \frac{e^{\frac{-(t-\mu)}{b}}}{2b} dt + F_X(\mu) \tag{5}$$

$$1 - \frac{e^{\frac{-(x-\mu)}{b}}}{2} \tag{6}$$

More formally, this can be defined as follows:

$$\boxed{\mathbf{F_X(x)} = \begin{cases} \frac{e^{\frac{(x-\mu)}{b}}}{2} & x \leq \mu \\ 1 - \frac{e^{\frac{-(x-\mu)}{b}}}{2} & x > \mu \end{cases}} \tag{7}$$

[2]Probability and Random Processes,Yates and Goodman
[3]https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.laplace.html
[4]This is the implementation used for the algorithm

2.4 Inverse Derivation

Now, since we are dealing with Inverse Transform Sampling, it is also necessary to find the inverse of each part of this piece wise function. We define the inverse as follows, given that $y = F_X(x)$ is the CDF

$$F_X^{-1}(y) = x \tag{8}$$

Applying the inverse process to the piece wise functions in equation 7 we obtain [5]:

$$F_X^{-1}(y) = 2 * b * ln \mid 2 * x \mid : x \in x \leq \mu \tag{9}$$

$$F_X^{-1}(y) = -1 * b * ln \mid -2 * x + 2 \mid : x \in x > \mu \tag{10}$$

Using equations 9 and 10, we can write the algorithm we need as described below.

3 Algorithm

First, let InverseLess(x) be a function that calls equation (9) and InverseGreater(x) be a function that calls equation (10)

Algorithm 1: Inverse Transform Sampling

```
Result: A point from the Uniform Random Variable is transformed
        appropriately
Data: A uniform random variable distribution
Initialize: an empty array, data
for sample in uniform random sample do
    if $x \leq \mu$ then
        transformedSample = InverseLess(x);
    else
        transformedSample = InverseGreater(x);
    end
end
data(i) $\longleftarrow$ $transformedSample$
```

Using this algorithm, we can then accurately transform the data. A for loop was used to transform the data one by one since the Laplace random variable is two sided, so each element was transformed individually according to the Inverse CDF.

4 Results

4.1 A Note On Limitations

In this example, Python embeded in Jupyter Notebooks was used for organizational purposes as opposed to Matlab. However, due to how some external libraries may be imported, it is important to note that some rounding errors may have occurred.

[5]http://wiki.stat.ucla.edu/socr/index.php/AP$_{S}tatistics_{C}urriculum_{2}007_{L}aplace$

4.2 Results

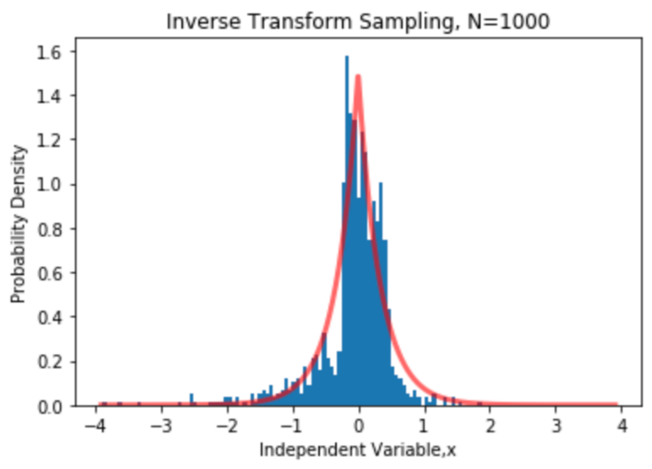

Figure 1: N = 1000 test

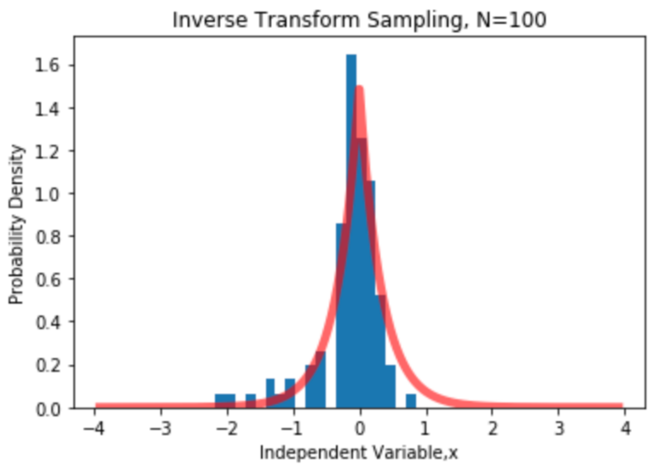

Figure 2: N = 100 Test

In order to best assess the results, the probability from the array with the transformed values was plotted as a histogram against the actual distribution. Overall, the data closely matches the desired distribution . When looking at error, since this is a random distribution that was generated, we cannot assess how effective this method was with error alone. However, it is important to note that when the number of samples was increased from N=100 to N=1000, the error in both the mean and variance went down from around 20% and 10%respectively to 11%and 6% respectively.

5 References

References

[1] "AP Statistics Curriculum 2007 Laplace." Socr RSS, wiki.stat.ucla.edu/socr/index.php/AP Statistics Curriculum 2007 Laplace.

[2] Route, Matthew. "Radio-Flaring Ultracool Dwarf Population Synthesis." ArXiv.org, 15 Aug. 2017, arxiv.org/abs/1707.02212.

[3] "Scipy.stats.laplace." Scipy.stats.laplace - SciPy v1.5.1 Reference Guide, docs.scipy.org/doc/scipy/reference/generated/scipy.stats.laplace.html.

[4] Yates, Roy D., and David J. Goodman. Probability and Stochastic Processes: a Friendly Introduction for Electrical and Computer Engineers. John Wiley amp; Sons, 2015.

6 Appendix

6.1 Python Code

```
#Relevant imports
import numpy as np
import scipy as sc
import matplotlib.pyplot as plt
from scipy.stats import laplace
from scipy import interpolate
import cProfile
import numpy as np
import scipy as sc
import matplotlib.pyplot as plt
import math
from scipy.stats import laplace
#The scale variable
b = (1/3)
#Set up the distribution and plot it
fig, ax = plt.subplots(1, 1)
mean,var,skew, kurt = laplace.stats(moments='mvsk')
x = np.linspace(laplace.ppf(0.01),laplace.ppf(0.99), 10000)

deku = []

test_2_container = []

#Generate the uniform random variable.
test_1 = np.random.uniform(-1,1,1000)
test_2 = np.random.uniform(-1,1,100)
```

```
#For loop to accurately transform the data
for x in test_1:
    if x<=0:
        add_me = 2*b*math.log(abs(2*x))
    else:
        add_me = -1*b*math.log(abs(-2*x+2))

    deku.append(add_me)

#For loop to accurately transform the data
for x in test_2:
    if x<=0:
        add_me_2 = 2*b*math.log(abs(2*x))
    else:
        add_me_2 = -1*b*math.log(abs(-2*x+2))

    test_2_container.append(add_me_2)

test1_as_np = np.asarray(deku, dtype=np.float32)
test2_as_np = np.asarray(test_2_container, dtype=np.float32)

#N=1000
fig, ax = plt.subplots(1, 1)
mean,var,skew, kurt = laplace.stats(moments='mvsk')
x = np.linspace(laplace.ppf(0.01),laplace.ppf(0.99), 1000)
plt.title("Inverse Transform Sampling, N=1000")
count2, bins2, ignored2 = plt.hist(test1_as_np,100,density=True)
ax.plot(x, laplace.pdf(x,0,b),'r-', lw=3, alpha=0.6, label='laplace pdf')

#N=100
fig, ax = plt.subplots(1, 1)
mean,var,skew, kurt = laplace.stats(moments='mvsk')
x = np.linspace(laplace.ppf(0.01),laplace.ppf(0.99), 1000)
count2, bins2, ignored2 = plt.hist(test2_as_np, 15, density=True)
plt.title("Inverse Transform Sampling, N=100")
ax.plot(x, laplace.pdf(x,0,b),'r-', lw=5, alpha=0.6, label='laplace pdf')
```