CS6410 Assignment 3

Robert Stanton 113542767

March 31, 2020

This report outlines the operation of a random number generator (RNG) developed for CS6410. The RNG presented in this report can sample numbers from both the Logarithmic and Kumaraswamy distributions. The report is presented in two sections. Section 1 contains information on the distributions available to the RNG, and the different sampling methods used. Section 2, the usage guide, contains information on the prerequisite libraries/packages needed, and instructions for generating random variates.

Contents

1 Distribution Information

Logarithmic Distribution

The Logarithmic distribution is a discrete probability distribution of one parameter, p; such that $0 < p < 1$. The probability mass function (PMF) is given as:

$$f(k) = \frac{-1}{\ln(1-p)} \frac{p^k}{k}$$

The cumulative distribution function (CDF) is given as:

$$F(k) = 1 + \frac{\mathrm{B}(p; k+1, 0)}{\ln(1-p)}$$

Where B is the incomplete beta function.
The CDF of the Logarithmic distribution can alternatively represented as:

$$F(k) = \sum_{k_i \leq k} f(k_i)$$

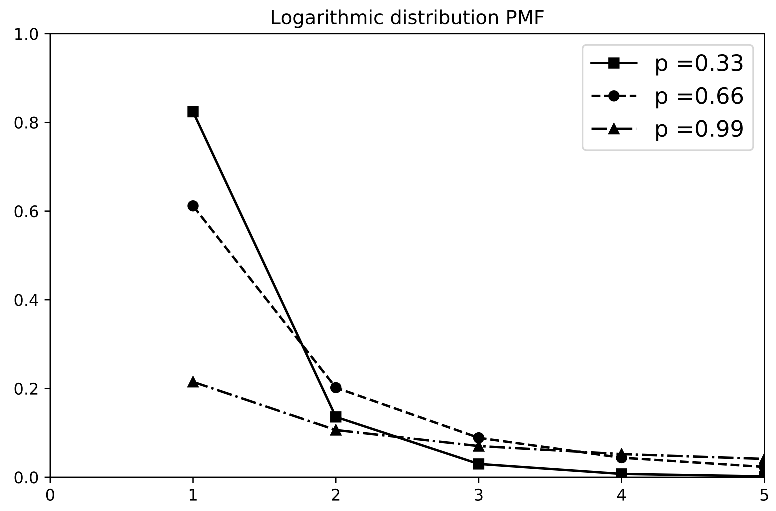

Figure 1: The probability mass function of the Logarithmic distribution.

Which is the sum of the PMF, $f(k_i)$, for each k_i up to, and including, the k of interest. This is method by which the CDF is calculated in this tool.
Figure 1 shows the PMF of the Logarithmic distribution and Figure 2 shows its CDF. As a discrete distribution, the PMF and CDF are only defined at the integer values $k \in \{1, 2, 3, ...\}$; the lines shown in both Figures do not imply continuity.

The method used to sample pseudo-random numbers from the Logarithmic distribution is as follows: first, a set of intervals, $S = \{[F(0), F(1)), [(F(1), F(2)), ..., [F(k-1), F(k))\}$, and a random uniform variate, u, between [0, 1], are generated. The index of the interval in S, in which u lies, is the Logarithmic distributed random variate returned by the RNG.

Kumaraswamy Distribution

The Kumaraswamy distribution is a continuous probability distribution. The probability density function (PDF) and CDF of the Kumaraswamy distribution can be represented in closed form. The distribution takes two parameters, α and β, such that: $\alpha > 0$, and $b > 0$. The PDF of the Kumaraswamy distribution is:

$$f(x) = \alpha\beta x^{\alpha-1}(1 - x^{a})^{\beta-1}$$

The CDF of the Kumaraswamy distribution is:

$$F(x) = 1 - (1 - x^{\alpha})^{\beta}$$

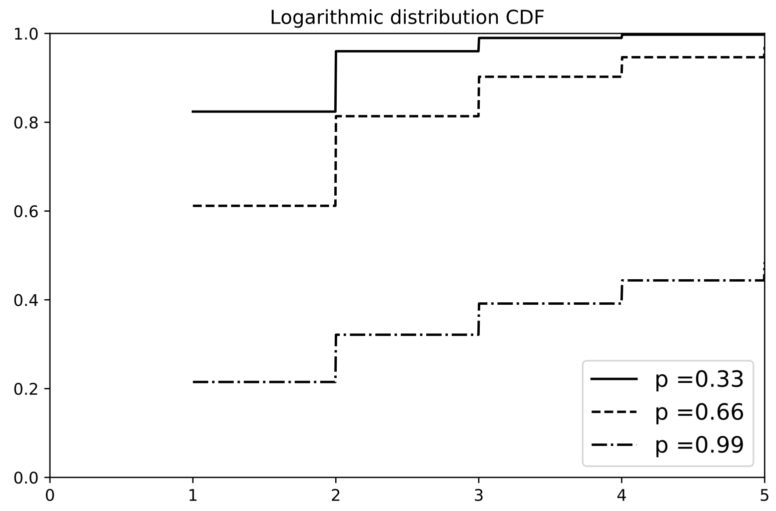

Figure 2: The cumulative distribution function of the Logarithmic distribution.

Figures 3 and 4 show the PDF and CDF of the Kumaraswamy distribution. The values of α and β for which the PDF and CDF have been calculated are also shown.

As the Kumaraswamy distribution is continuous (and has an invertible CDF), the RNG uses inverse transform sampling to generate the desired random variates. Necessary for this method of pseudo-random number sampling is the quantile function, or inverse CDF. The quantile function of the Kumaraswamy distribution is:

$$F^{-1}(x) = \sqrt[\alpha]{1 - \sqrt[\beta]{1-x}}$$

To sample Kumaraswamy distributed random variates by inverse transform sampling: first, a random uniform variate, u, from the interval [0, 1], is generated. Then, $K = F^{-1}(u)$ is calculated. K is the result of the random uniform variate, u, being passed as an argument to the inverse CDF, F^{-1}. K can be shown to be Kumaraswamy distributed with the parameters α and β as specified in $F^{-1}(u)$.

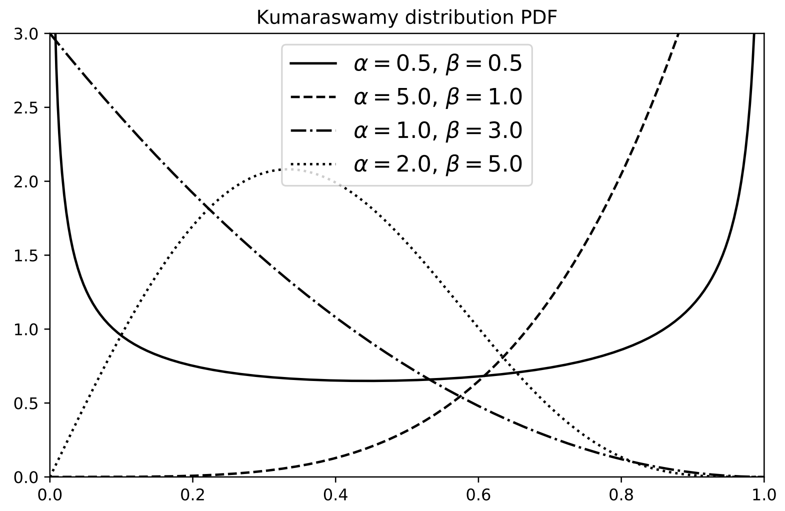

Figure 3: The probability density function of the Kumaraswamy distribution

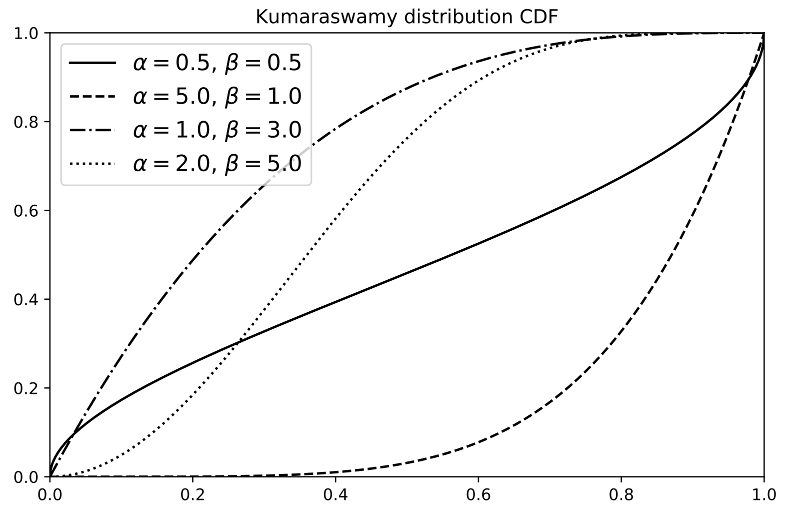

Figure 4: The cumulative distribution function of the Kumaraswamy distribution

2 Usage Guide

Prerequisites

Several packages from the SciPy [5, 3] library are necessary to for the RNG to function correctly. These are:

- Matplotlib [2]:

  Provides the core graphical functionality of the RNG.

- Numpy [4, 6]:

  Necessary for the correct functionality of Matplotlib.

All necessary packages, and detailed instructions on their installation, can be found on the SciPy website [1].

Instructions

Inputs

The RNG is operated from the command line. Using a command-line interface, navigate to the folder containing the RNG. It is recommend to store the RNG in a folder of it's own as it generated several output files. The RNG has the following signature that must be adhered to when called:

```
n seed dist par1 [par2]
```

- int n

  The number of random variates to generate. *n* should be a positive integer; however the RNG will accept 0 as a valid input.

- int seed

  The seed value used to initialise the inbuilt pseudo-random, uniform, number generator. Setting a seed value ensures that the random sequence generated by the tool is reproductible. *seed* should be an integer.

- str dist

  The desired distribution from which to generate random variates. *dist* takes the case-insensitive string 'LOG' or 'KUM' as input; which refer to the Logarithmic or Kumaraswamy distributions respectively.

- float par1

  The meaning of *par1* is dependent on the chosen distribution. In the Logarithmic case, *par1* refers to the parameter p. For the Kumaraswamy distribution, *par1* refers to α. In both cases, *par1* should obey the parameter constraints of the respective distribution; $0 < p < 1$ or $\alpha > 0$.

- float par2

  Refers the parameter β of the Kumaraswamy distribution. Should *par2* be given as an argument to the Logarithmic distribution, it will be ignored.

An example of a correctly formulated script call is as follows:

```
python myRng.py 1000 5 LOG 0.66
```

This code generates 1000 Logarithmic distributed random variates with parameter $p = 0.66$, and a seed value of 5.

Outputs

The script generates three output files, which can be found in the same folder as the RNG:

- seq_.txt

  This file contains the n random variates generated by the RNG.

- cdfData_.txt

  Contains ordered tuples in the form $(x,\ F(x))$, where $F(x)$ is the CDF of the distribution specified in *dist*. There are a significantly different number of tuples generated depending on the distribution chosen. In the Logarithmic case, six tuples, representing $(k,\ F(k))$ for $k \in \{1, 2, ..., 6\}$, are presented. This reflects the discrete nature of the Logarithmic distribution that is only defined at integer values $k > 0$. The CDF of the Kumaraswamy distribution is evaluated for 1000 floating point values on the interval [0, 1]. This level of accuracy should be sufficient for most statistical purposes.

- cdf_.pdf

  Contains a plot of the empirical and theoretical CDFs of the chosen distribution. The empirical CDF is derived from the variates generated by the RNG. Increasing the size of n typically results in the empirical curve aligning closer to the theoretical. Figure 5 shows an example of a CDF plot generated by the script.

Figure 6 shows the CDF of Kumaraswamy distribution with the parameters $\alpha = 5$ and $\beta = 1$. The empirical CDF is shown as the grey shaded area, while the theoretical CDF is given by the dashed black line. The parameters of this distribution can be seen in the title of the plot.

Figure 7 shows the CDF of the Kumaraswamy distribution distribution where $n = 100$. As can be seen, smaller values of n can lead to large differences in the empirical and theoretical CDFs. Increasing the size of n typically leads to better convergence of both curves.

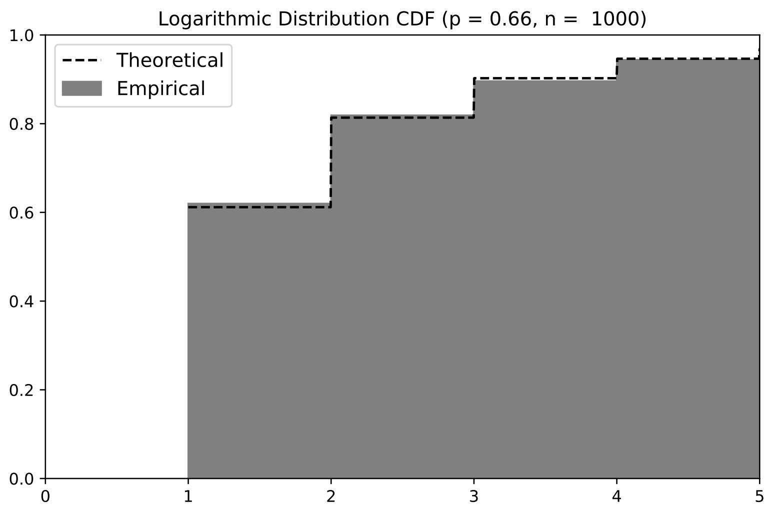

Figure 5: An example of the empirical and theoretical CDFs produced by the script. The arguments used to generate this CDF can be seen in the figurehead.

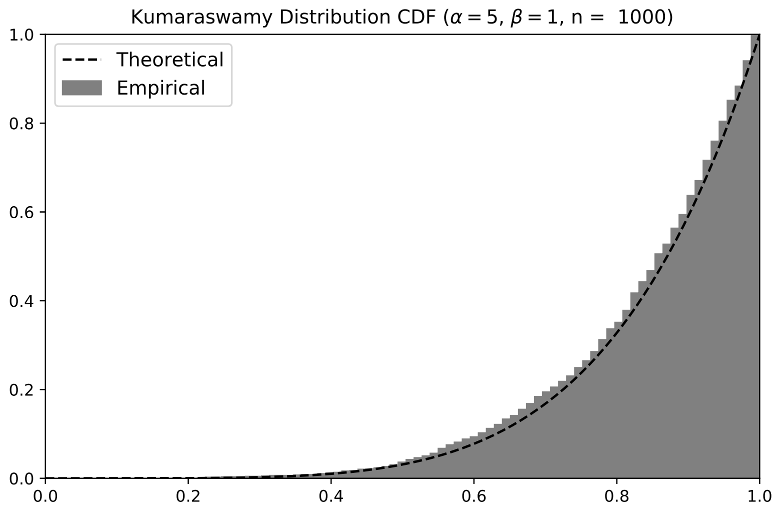

Figure 6: The Kumaraswamy distribution for the given parameters. The relatively large n gives better convergence between the empirical and theoretical curves.

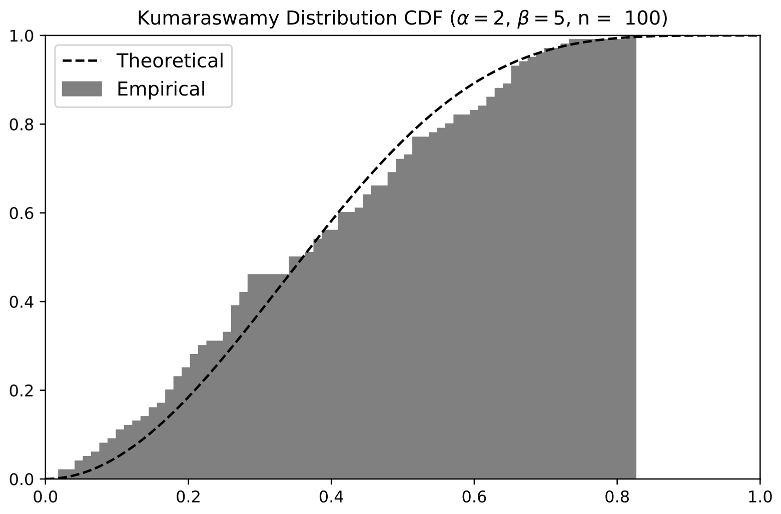

Figure 7: As can be seen here, generating a smaller number of random variates results in greater residuals between the expected and observed distributions. From a scientific point of view, it is necessary to generate and experiment with large amounts of random data to ensure that random inputs truly reflect the distribution from which they are drawn.

References

[1] Scipy.org.

[2] John D Hunter. Matplotlib: A 2d graphics environment. *Computing in science & engineering*, 9(3):90–95, 2007.

[3] K Jarrod Millman and Michael Aivazis. Python for scientists and engineers. *Computing in Science & Engineering*, 13(2):9–12, 2011.

[4] Travis E Oliphant. *A guide to NumPy*, volume 1. Trelgol Publishing USA, 2006.

[5] Travis E Oliphant. Python for scientific computing. *Computing in Science & Engineering*, 9(3):10–20, 2007.

[6] Stéfan van der Walt, S Chris Colbert, and Gael Varoquaux. The numpy array: a structure for efficient numerical computation. *Computing in Science & Engineering*, 13(2):22–30, 2011.