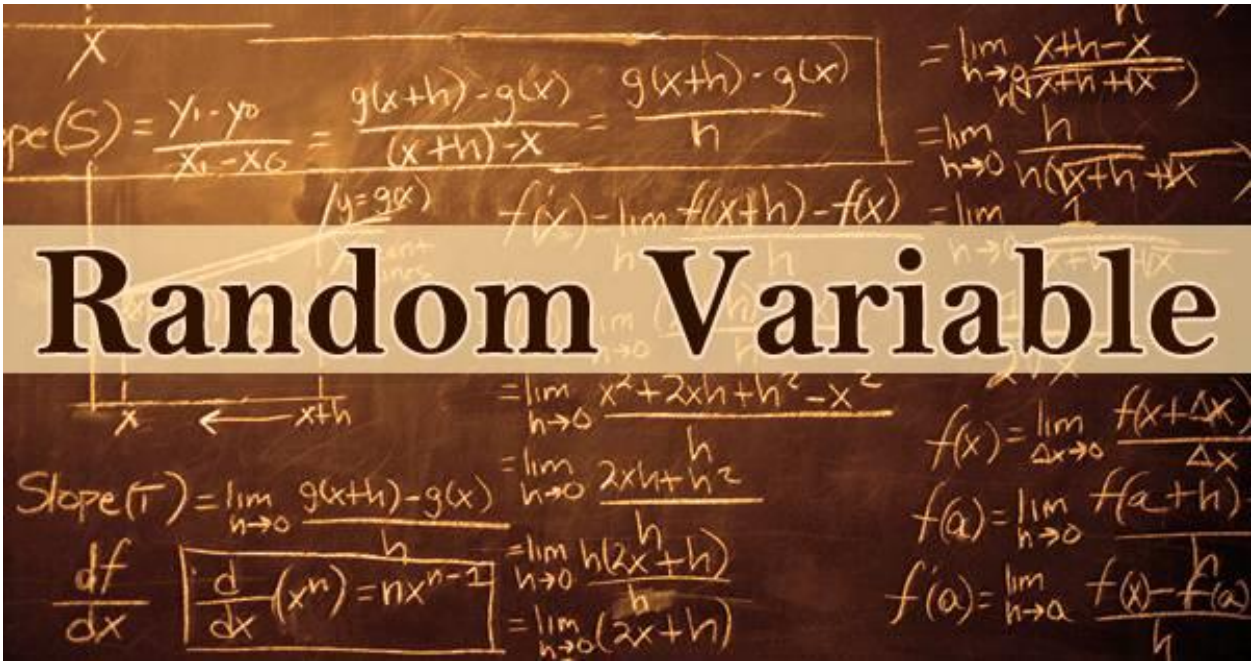

Random Variable Generation

By: Joshua McDonald & Ni Li

TABLE OF CONTENT

I. Abstract

Random variate generation is a fundamental aspect of simulation modeling and analysis. The objective of random variate generation is to produce observations that have the stochastic properties of a given random variable. To this end, we have developed a library to generate random variates using programing language go that are convenient to use. Please refer to our coding document for the library details.

II. Background

Given an experiment with a set of possible outcomes, a random variable is defined as a measurable function from possible outcomes to a measurable space. Random variate generation is the process of producing observations that have the stochastic properties of a given random variable. The ability to create stochastic simulation models for analysis and decision-making relies heavily on random variate generators. The topic of random variate generation appears in many papers throughout history and continues today.

III. Objective and Concepts

The objective of random variate generation is to produce sample observations that have the stochastic properties of a given random variable, X, having a distribution function: $F(x) = Pr(X \leq a)$ $(-\infty < a < \infty)$. The primary original concepts on which many random variate generators are developed described below:

- **The Inverse Transform Method**

The inverse transform method utilizes the inverse of the cumulative distribution function (CDF) of the random variable under consideration to generate observations. The general approach is as follows:

Given a random variable X with CDF F(x), generate a U(0,1) random number n corresponding to the ^{nth} fractile of the CDF, F(x) = n. The value of the random variate generated is the value of x such that $x = F^{-1}(n)$.

- **The Composition Method**

Situations arise in which a density function (f), can be written as a weighted sum of r other densities (Cheng 1998):
$f(x) = \sum_{i=1}^{r} pfi(x)$, where $p > 0$ and the sum of p equals to 1. The density f is referred to as a compound or mixture density.

- **The Acceptance–Rejection Method**

The acceptance–rejection method is often used when a closed-form cumulative distribution function does not exist or is difficult to calculate. In this method, variates are generated from one distribution and are either accepted or rejected in such a way that the accepted values have the desired distribution. Schmeiser presents the following general acceptance-rejection algorithm:

Given a random variable X, let f(x) denote the desired density function of X. Let t(x) be any majorizing function of f(x) such that t(x) ≥ f(x) for all values of x. Let g(x) = t(x)/c denote the density function proportional to t(x) such that c = $\int_{-\infty}^{\infty} t(x)dx$. The steps are:

1. Generate x ~ g(x).
2. Generate u ~ U(0,1).
3. If u > f(x)/t(x), then reject x and go to step 1.
4. Return x.

Based on the original concept, we have picked a programing language go to generate our findings. Go is a language that lacks many of the distribution functions that can be found in a language such as Python. Thus, the opportunity to add this library to the language made sense and presented more of a challenge than other languages.

IV. Key Findings

We have selected continuous and discrete distribution families that are common and generate the random variables, means, and variances for them. The distributions include:
Discrete:

- Bernoulli
- Negative Binomial
- Poisson
- Geometric

Continuous:

- Exponential
- Standard Normal
- Erlang
- Triangular
- Gamma
- Weibull

Bernoulli

Bernoulli's RandVar() takes a probability of p, and given a Unif(0,1) number returns a 0 if the random number is less than or equal to p, otherwise returns 1.

The function takes a float64, probability as a parameter. The function will return an error if the probability value is *less than* 0 OR *greater than* 1.

Example

```
var p = .25

d := BernoulliDistribution{

    DistributionType: "Bernoulli",

}

n, _ := d.RandVar(p)
```

Mean

`E(X) = p`

Variance

`Var[X] = pq = p(1-p)`

Distribution plot:

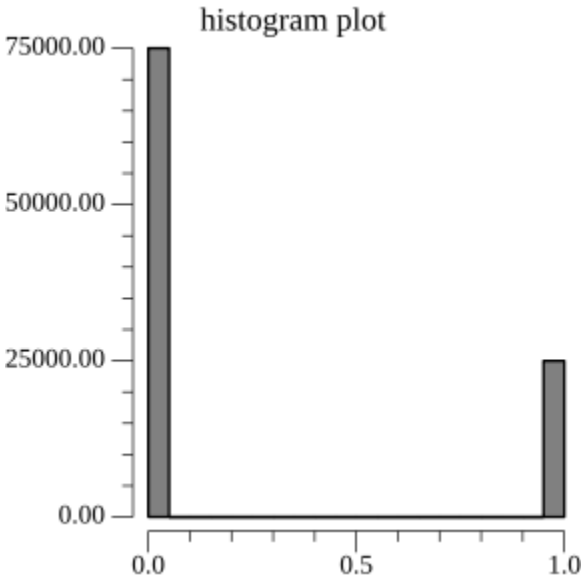

Figure 1: Bernoulli distribution

Negative Binomial

The negative binomial distribution is used to model the number of failures *x* before the *nth* success. The RandVar() function takes in *p*, which is the probability of success, and *n*, which is the number of successes. This function calculates the sum of *n* geometric variates G(p).

The parameter *p* must be `0 < p < 1` The parameter *n* must be a positive integer

```
    p := .25
    n := 4
    d := NegativeBinomialDistribution{
        DistributionType: "NegativeBinomialDistribution",
    }
    x, _ := d.RandVar(p, n)
```

Expected Value

`E(X) = n(1-p)/p`

Variance

`Var(X) = n(1-p)/p^2`

Distribution plot:

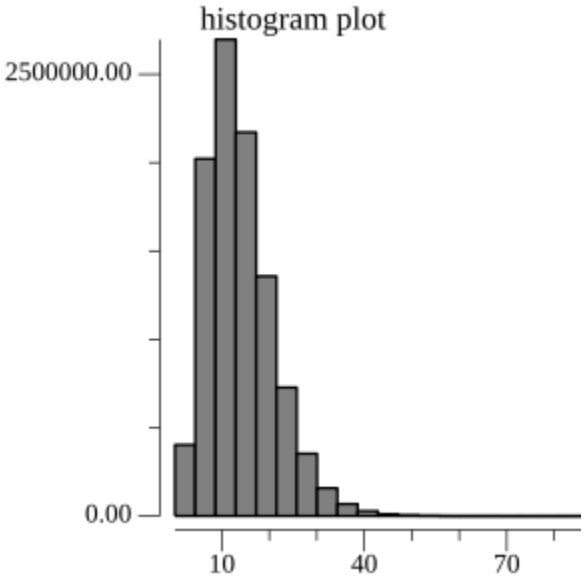

Figure 2: Negative binomial distribution

Poisson

The Poisson distribution is used to model the number of arrivals over a given interval.

Since the function is using the direct method the value of λ has been limited to 20.

The parameter λ must be > 0 and < 21

```
    lambda := float64(2)
    d := PoissonDistribution{
```

```
        DistributionType: "Poisson",
    }
    n, _ := d.RandVar(lambda)
```

Expected Value

$E(X) = \lambda$

Variance

$Var(X) = \lambda$

Distribution plot:

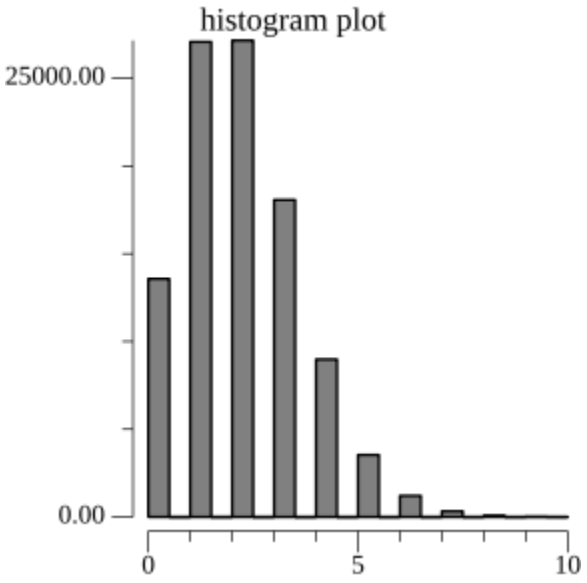

Figure 3: Poisson distribution

Exponential

The exponential distribution is commonly used to model the time between events or the time between failures.

The RandVar() function generates random numbers from a Unif(0,1) random number so that the returned value fits an exponential distribution given a scale parameter which is a float64.

The function will return an error if the scale parameter value is *less than* 0 OR *equal to* 0.

Example

```
    a := 1.0 // scale parameter
    d := ExponentialDistribution{
        DistributionType: "Exponential",
```

```
    }

    n, _ := d.RandVar(a)
```

Mean

`E[X] = 1/scale parameter`

Variance

`Var(X) = 1/scale parameter^2`

Distribution plot:

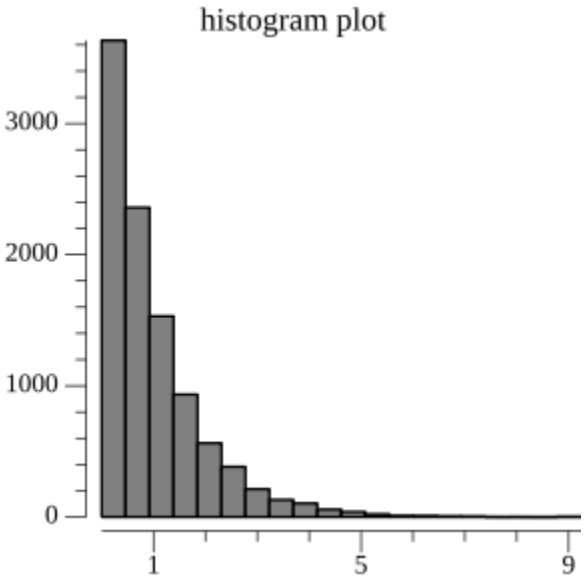

Figure 4: Exponential distribution

Standard Normal

The normal distribution's RandVar() function takes a mean and standard deviation and returns two random variables z1, z2 using the Box-Muller method.

Example

```
    mean := .5

    sd := 1

    d := NormalDistribution{

        DistributionType: "Normal",

    }

    n, _ := d.RandVar(m, sd)
```

Expected Value

`E[X] = μ`

Variance

`Var(X) = σ^2`

Distribution plot:

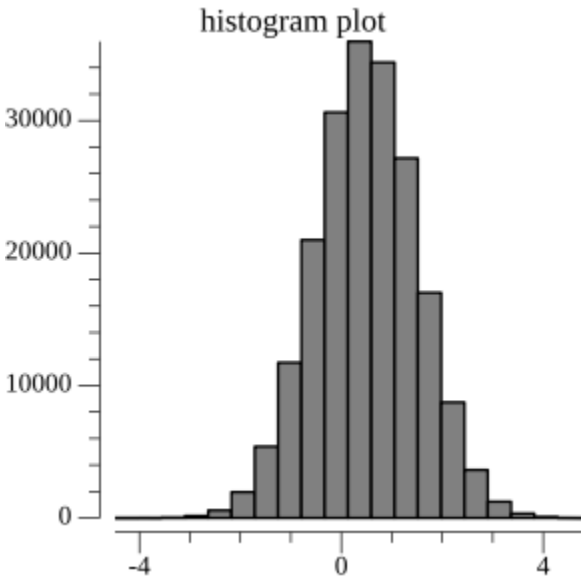

Figure 5: Standard normal distribution

Erlang

Where the events that occur can be modeled by the Poisson distribution, the waiting times between k occurrences of the event are Erlang distributed.

```
    lambda := float64(2)
    k := 1


    d := ErlangDistribution{
        DistributionType: "Erlang",
    }
    n, _ := d.RandVar(k, lambda)
```

Expected Value

`E(X) = k / λ`

Variance

```
Var(X) = k / λ^2
```

Distribution plot:

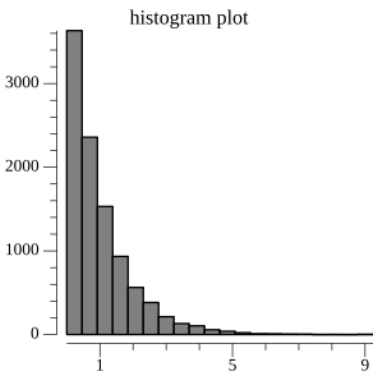

Figure 6: Erlang distribution

Triangular

A triangular distribution has a lower limit (min), an upper limit (max), and mode and is used to describe a population when there is a limited amount of sample data.

The *min* parameter must be lower than the *max* parameter

```
    min := float64(0)

    mode := .5

    max := float64(1)


    d := TriangularDistribution{

        DistributionType: "Triangular",

    }

    n, _ := d.RandVar(min, mode, max)
```

Expected Value

```
E(X) = (min + mode + max) / 3
```

Variance

```
Var(X) = (min^2 + max^2 + mode^2 - (min * max) - (min * mode) - (max * mode)) / 18
```

Distribution plot:

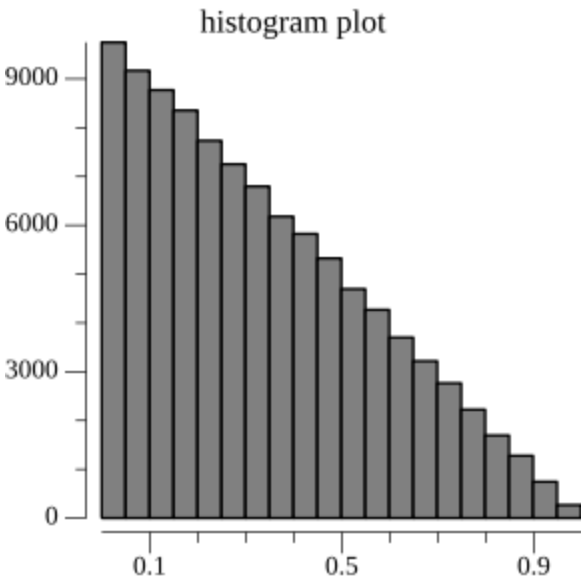

Figure 7: Triangular distribution

Weibull

The RandVar() function of the Weibull distribution take a scale parameter *a* and a shape parameter *b* and returns a random variable that fits the Weibull distribution.

The parameter *a* must be > 0 The parameter *b* must be > 0

Weibull is commonly used to model failure rates of electronics where the failure rate :

`_b_ < 1` ▶ increases over time

`_b_ > 1` ▶ decreases over time

`_b_ = 1` ▶ constant over time

Example

```
    a := 1.5

    b := float64(1)


    d := WeibullDistribution{

        DistributionType: "Weibull",

    }

    n, _ := d.RandVar(a, b)
```

Expected Value

`E(X) = a / b Γ (1/b)`

Variance

```
Var(X) = a^2 / b^2 [2 * b * Γ(2/b) - {Γ(1/b)}^2}]
```

Distribution plot:

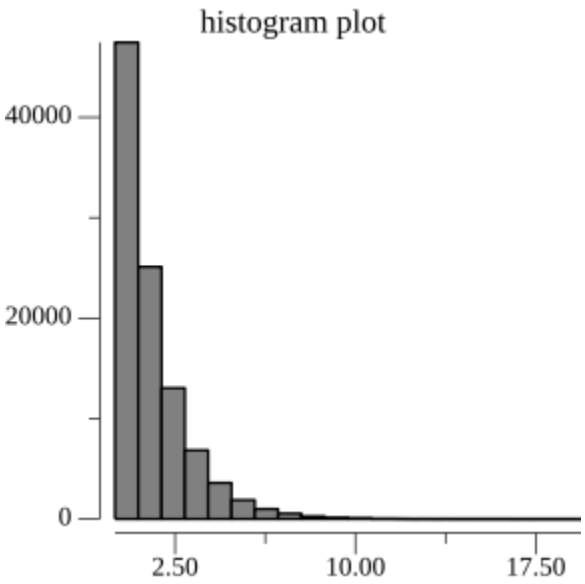

Figure 8: Weibull distribution

Gamma

The gamma distribution RandVar() function produces random variables given the shape parameter *k* and scale parameter *s*.

This generator relies on the Matlab example, which has been translated to Go and uses the standard normal random variate generator function from this same package.

```
    k := 5
    s := 1


    d := GammaDistribution{
        DistributionType: "Gamma",
    }
    n, _ := d.RandVar(k, s)
```

Expected Value

```
E(X) = k / s
```

Variance

```
Var(X) = k / s^2
```

Distribution plot:

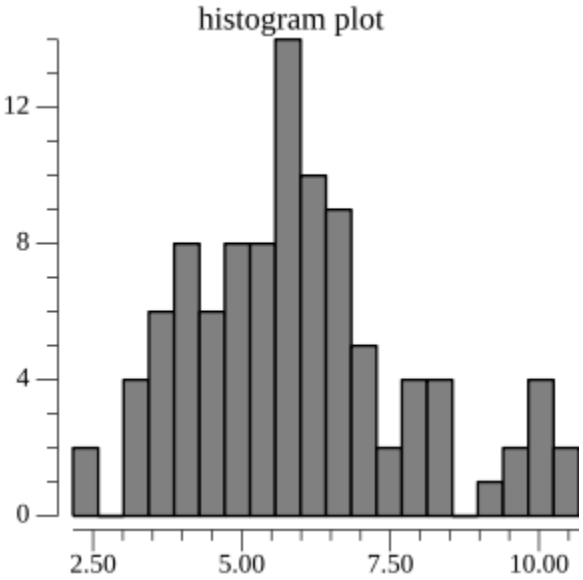

Figure 9: Gamma distribution

V. Conclusion

Random variate generation has had extensive and interesting importance to the field of simulation. Even though random variate generation has been studied extensively, opportunities and challenges still exist for developing random variate generations to accurately and efficiently simulate systems to solve complex problems. We hope our random number generation library could be used as a starting point to help to build up solutions to solving more complex problems in the future and provide more insight for further research in this field.

VI. References

1. Cheng, R. C. H. 1998. "Random Variate Generation". In Handbook of Simulation: Principles, Methodology, Advances, Applications, and Practice edited by J. Banks, 139–172. New York: Wiley.

2. Keeler, Author Paul. “Simulating Poisson Random Variables - Direct Method.” *H. Paul Keeler*, 23 Feb. 2021, hpaulkeeler.com/simulating-poisson-random-variables-direct-method/.

3. “Poisson Random Number Generator.” *Poisson Random Number Generator - MATLAB Answers - MATLAB Central*, www.mathworks.com/matlabcentral/answers/28161-poisson-random-number-generator.

4. “Simulating Discrete (Geometric, Poisson And Zero-Inflated Poisson, Negative Binomial And Zero-Inflated Negative Binomial) Random Variables | Stata Code Fragments.” *IDRE Stats*, stats.idre.ucla.edu/stata/code/imulating-discrete-geometric-poisson-and-zero-inflated-poisson-negative-binomial-and-zero-inflated-negative-binomial-random-variables/.

5. Hong, Liangjie, et al. “Hong, LiangJie.” *Hong, LiangJie | Director of Engineering - AI at LinkedIn Corporation*, 19 Dec. 2012, www.hongliangjie.com/2012/12/19/how-to-generate-gamma-random-variables/.

6. Xi, Bowei, et al. “Logarithmic Transformation-Based Gamma Random Number Generators.” *Journal of Statistical Software*, www.stat.purdue.edu/~xbw/research/jss102013.gamma.pdf.