Transforming Renewal Processes for Simulation of Nonstationary Arrival Processes

Jacob Shusko

December 10, 2022

Abstract

Simulating arrival processes is an important task in modeling real-life systems. Often, exponential inter-arrivals with a constant rate are assumed due to the convenient mathematical properties of stationary Poisson Processes. In previous literature, methods have been developed for transforming stationary Poisson Processes into non-stationary Poisson Process i.e those with time varying rates. In this paper, I replicate techniques developed by Ira Gerhardt and Barry Nelson [3] that leverage existing methods for transforming stationary Poisson Processes to simulate general non-stationary renewal processes. In the final section, I present a use case of these methods for modeling high value ransomware Bitcoin heists.

1 Introduction

For many applications of simulation models for real-life systems, we assume there are Poisson arrivals. The Poisson arrival process has useful mathematical properties that allow for easy simulation. As with all models, the Poisson Process (PP) comes with some assumptions about the underlying process. For one, it is assumed that there are no "batch" arrivals. There is also the assumption that interarrival times are independent and identically distributed (stationary) i.e there is a constant arrival rate. Clearly, there are many applications for which these assumptions do not hold, but fortunately we can modify PP to account for these cases.

In this paper, we exploit two known methods for transforming a stationary PP into a non-stationary Poisson Process ($NSPP$): inversion and thinning. In many applications, these transformations would be sufficient for modeling the underlying non-stationary arrival process, but for some systems this transformation can provide an inaccurate representation of the arrival process i.e the actual arrivals may be less or more variable than the $NSPP$. To solve this problem, Barry Nelson and Ira Gerhardt leverage these known techniques that transform a stationary PP into a $NSPP$ by transforming a stationary renewal process (RP) to obtain a non-stationary non-Poisson ($NSNP$) process as desired [3]. The results indicate that the desired arrival rate is achieved and that when the base RP is more or less variable than PP then the transformation generates a $NSNP$ that is also more or less variable than PP.

First, in Section 2, I formally define RP and demonstrate the inversion and thinning algorithms for transforming this process to its non-stationary form. In this section, the key difference between RP and PP is highlighted: RPs allow general inter-arrivals distributions while PP requires exponential inter-arrivals. For this reason, RPs can be

used for a wider range of applications as long as there is a way to effectively simulate these processes. In particular, exponential inter-arrivals are "memoryless" which is a strong assumption. An example where this is unrealistic is the lifetime of components. If one knew that a component has a lifetime of s years, the memoryless property would imply that the probability this component lasts an additional t years is the same as the probability a new component lasts t years [5]. One can see, that at least in this example one would like to have more general inter-arrivals distribution, potentially a Weibull distribution. In Sections 2.3 and 2.3, I state the two modified algorithms for transforming a RP into a $NSNP$ process with a specified inter-arrivals distribution $G(t)$ and a rate function $r(t)$.

In Section 3, I specify these algorithms for Weibull inter-arrivals. The Weibull distribution is often used to model failure times in reliability analysis [2]. It also has applications in industrial engineering to represent manufacturing and delivery times as well applications in modeling grinding, milling and crushing operations.

Finally, in Section 4, I explore applying these methods to a dataset. Currently, my research is focused on modeling cyber threat scenarios. It is critical to be able to model the arrival process of cyber threats to a system to get a realistic simulation model. The main application of my research is modeling ransomware attacks on semiconductor fabs, but due to the lack of publicly available data, I chose to use a more rich dataset on ransomware attacks called the 'BitcoinHeistRansomwareAddressDataset' [1]. This dataset was created by parsing the entire Bitcoin transaction graph for a decade, recording for each Bitcoin transaction, the address, date of transaction, Satoshi amount, whether the address was known to be ransomware and from what ransomware family, and other graph features such as number of neighors.

2 Methods

2.1 Renewal Processes

Before defining RP, first note we can consider both inter-arrivals and inter-events as having the same mathematical meaning. The choice of which term to use mainly rests on the application at hand, but we will stick to inter-events as this seems to be the more general choice. The following definition is based primarly on Kulkarni's work [5]. Consider a process of events occurring over time. Let $S_0 = 0$ and S_n be the time of occurrence of the n^{th} event for $n \geq 1$. Then, assume that

$$0 \leq S_1 \leq S_2 \leq S_3 \leq \dots$$

and define

$$X_n = S_n - S_{n-1}, n \geq 1$$

Therefore, I have $\{X_n, n \geq 1\}$ as a sequence of inter-event times and clearly $X_n \geq 0$. Now I can define our counting process:

$$N(t) = \sup\{n \geq 0 : S_n \leq t\}, t \geq 0$$

Definition 1. *The sequence $\{S_n, n \geq 1\}$ is called a renewal sequence and the process $\{N(t), t \geq 0\}$ is called a renewal process generated by $\{X_n, n \geq 1\}$ if $\{X_n, n \geq 1\}$ is a sequence of non-negative i.i.d. random variables.*

I consider a common cumulative distribution function G for each inter-event time in the sequence $\{X_n, n \geq 1\}$. In Gerhardt and Nelson's paper [3], the algorithms allow X_1

to have a different distribution say G_e, but for our use case I will not end up needing this generalization. For the rest of the paper, I will use the following shorthand throughout:

$$G(t) = \Pr(X_1 \leq t), \tau = E[X_1], \sigma^2 = \text{Var}(X_1), \text{ and } \text{cv}^2 = \frac{\sigma^2}{\tau^2}$$

I assume that $\lim_{h\to 0} G(h) = 0$ and that I can explicitly compute the density from G as such $g(t) = \frac{d}{dt}G(t)$ for $t \geq 0$. Based on our assumptions on G, the stationary of N, and $\tau > 0$, I conclude that N is regular, which means that only one renewal may occur at a time [5].

Finally, I will consider a integrable rate function $r(t)$, and let $R(t) = \int_0^t r(u)du$ be the integrated rate function.

2.2 The Inversion Method

Now, I will present an inversion algorithm for generating inter-event times for an $NSNP$ process, and then analyze the mean and variance of said process. There is plenty of literature on using this method when the base process is Poisson with rate 1, but not when we have other stationary renewal base processes. To use the following algorithm, we must specify the distribution G such that $\tau = 1$ and consequently $\text{cv}^2 = \sigma^2$. We will make use of the following quantity in the algorithm: $R^{-1}(s) \equiv \inf\{t : R(t) \geq s\}$. Essentially, this method, explicitly stated in Algorithm 1, uses the following ideas:

1. Sample the inter-event times X_n from G and update the event times $S_n = S_{n-1} + X_n$.
2. Then adjust the event times as such $V_n = R^{-1}(S_n)$ and update the adjusted inter-event times $W_n = V_n - V_{n-1}$

Algorithm 1 The Inversion Method for transforming RP to $NSNP$ processes.

1: $V_0 \leftarrow 0$, $n \leftarrow 1$, $S_1 \sim G$, and $V_1 \leftarrow R^{-1}(S_1)$
2: $W_n \leftarrow V_n - V_{n-1}$
3: $n \leftarrow n + 1$, $X_n \sim G$, $S_n \leftarrow S_{n-1} + X_n$, and $V_n \leftarrow R^{-1}(S_n)$
4: Go to step 2.

The algorithm benefits from being particularly simple to implement as long as one specifies the distribution G correctly. We maintain four arrays X, S, V, and W and update each array for a specified number of samples. The main difficulty with implementing this algorithm is calculating R^{-1} given we have to first integrate $r(t)$ and then perform a search to find the infimum of $\{t : R(t) \geq s\}$.

Although for many distributions, we can find an explicit form for $R(t)$, I chose to utilize Python code `scipy.integrate.quad` which handles numerical integration for single-dimensional integrals by using Fortran code `QUADPACK` in the backend. `QUADPACK` uses a variety of automatic procedures to numerically integrate, most using adaptive quadrature [8].

In a subroutine, I search for $\inf\{t : R(t) \geq s_n\}$ by increasing t from some starting point t_0 by steps of ϵ, evaluate $R(t)$ numerically, and check if $R(t) \geq s_n$. To make this process faster, for each iteration n, I use $t_0 = V_{n-1}$ to reduce the search space. We can decrease the step size ϵ to get better samples at the expense of computational efforts. For most of the results I choose $\epsilon = 0.01$, but a more intelligent choice would

use properties from the rate function. Additionally, we could instead use explicit forms of the integrals, but for my use case this was not necessary.

As mentioned in Gerhardt and Nelson's paper [3], the sequence $\{W_n, n \geq 1\}$ is a potential realization of inter-event times for the non-stationary process from the inversion method and one with desirable properties. In particular, let $\{I(t), t \geq 0\}$ denote the counting process generated by the inversion method, then the $E\{I(t)\} = R(t)$ for all $t \geq 0$ and $\text{Var}\{N(t)\} \approx \text{cv}^2 R(t)$ for large t.

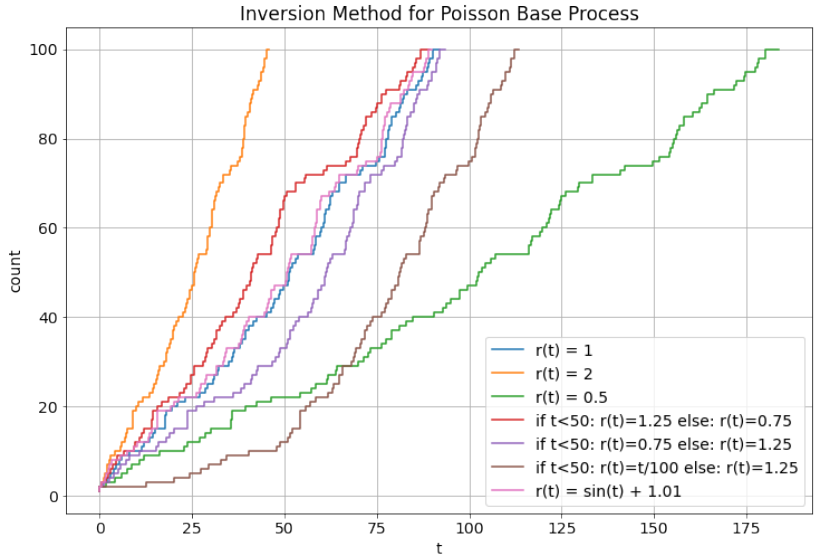

Figure 1: 100 samples from Inversion Method when the base process is Poisson under varying rate functions.

An important sanity check is to confirm that when N is Poisson with rate 1, the resulting process produced from the inversion method is an $NSPP$ with the desired properties. In this case, we have $\text{cv}^2 = 1$ and $G(t) = 1 - e^{-t}$ for $t \geq 0$. Typically, since it is desirable for $R(t)$ to be invertible, the following forms for the rate function are used: piecewise-linear, trigonometric, and piecewise-constant [3]. Sample paths for the inversion method when Poisson is the base process with rate 1 with a variety of rate functions is plotted in Figure 1. It is apparent that changing the rate function has large consequences on the behavior of the process. For instance, when the rate function is $r(t) = 2$ and $r(t) = 0.5$, the sample averages for 100 adjusted inter-event times are 0.459 and 1.837 with sample variances 0.221 and 3.527 respectively.

The last discussion point for this method is that cv^2 provides a measure of the deviation of the stationary renewal process from Poisson. In particular, we can say that when $\text{cv}^2 > 1$ the process is over-dispersed and when $\text{cv}^2 < 1$ the process is under-dispersed. Gerhardt and Nelson also demonstrate that the deviation measure is preserved by the inversion method for a large enough t, and thus cv^2 provides a single parameter by which to construct a process that is more or less variable than Poisson [3]. In the next section, I will discuss a method that is unaffected by analytical issues with $R(t)$.

2.3 The Thinning Method

As we saw in Section 2.2, I will present another algorithm for generating inter-event times by thinning a stationary renewal process to obtain a non-stationary process. First, for

this method, we choose a value $r^* \geq \max_{t\geq 0} r(t)$ and assign the stationary RP the arrival rate r^*. Then, we specify the distribution G with $\tau = (r^*)^{-1}$ and $\sigma^2 = \frac{\text{cv}^2}{\tau^2}$. In the case where the RP base is Poisson with rate λ, we have $\tau = \sigma^2 = \lambda$. This method, explicitly stated in Algorithm 2, uses the following ideas:

1. Sample the inter-event times X_n from G and update the event times $S_n = S_{n-1} + X_n$.
2. Then adjust the event times by thinning the sample i.e accept n^{th} event with event time S_n with probability $r(S_n)/r^*$. If accepted, update adjusted event time $T_k = S_k$ and adjusted inter-event time $Y_k = T_k - T_{k-1}$.

Algorithm 2 The Thinning Method for transforming RP to $NSNP$ processes.

$n \leftarrow 1$
$k \leftarrow 1$
$T_0 \leftarrow 0$
$S_1 \sim G$
$U_1 \sim$ Uniform$[0, 1]$
if $U_1 \leq r(S_1)/r^*$ **then**
    $T_1 \leftarrow S_1$
    $Y_1 \leftarrow T_1 - T_0$
    $k \leftarrow 2$
end if
$n \leftarrow n + 1$
$X_n \sim G$
$S_n = S_{n-1} + X_n$
$U_n \sim$ Uniform$[0, 1]$
if $U_n \leq r(S_n)/r^*$ **then**
    $T_k \leftarrow S_n$
    $Y_k \leftarrow T_k - T_{k-1}$
    $k \leftarrow k + 1$
end if
Go to step 11.

Thus, the sequence $\{Y_k, k \geq 1\}$ is a potential realization of inter-event times for the non-stationary process generated from the thinning method. Let $\{M(t), t \geq 0\}$ denote the counting process generated by this thinning method. Then, we have the following properties: $E[M(t)] = R(t)$ for all $t \geq 0$. The variance will depend on the expression for Var$(M(t))$ [3].

As before, we explore different rate functions for Poisson processes. Sample paths for the thinning method when Poisson is the base process with rate 1 with a variety of rate functions is plotted in Figure 2. We see similar sample paths as the inversion method. Again, the mean and variances of each sample path differs from the base process as expected. For instance, when the rate function is $r(t) = 2$ and $r(t) = 0.5$, the sample averages for 100 adjusted inter-event times are 0.509 and 2.036 with sample variances 0.231 and 3.695 respectively.

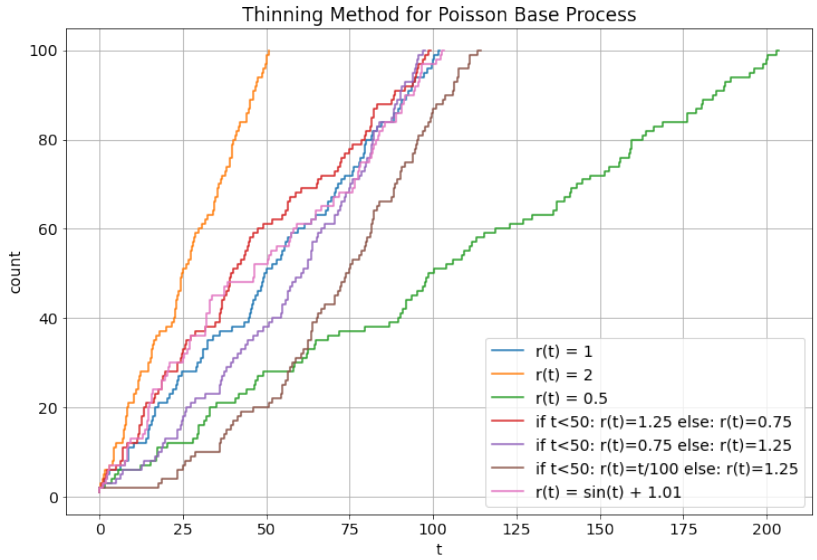

Figure 2: 100 samples from Thinning Method when the base process is Poisson under varying rate functions.

2.4 Equivalence of Inversion and Thinning

The two methods are expected to achieve the same arrival rate, but they are not equivalent in general. In particular, Gerhardt and Nelson argue that the expectations differ under certain rate functions, but when the process is Poisson with rate 1 they are equivalent.

In Figure 3, I generate 100 samples for 10 different replications for each method and then plot these sample paths. The plot demonstrates that at least for this particular trigonometric rate function, the thinning method produces samples paths that are less dispersed from a Poisson Process with rate 1. Across both methods, we can see that this particular rate function creates a sample path that is a bit more variable than a standard Poisson Process but follows the same trend. Hence, this rate function may be a good choice to include seasonal effects for count data by varying the amplitude.

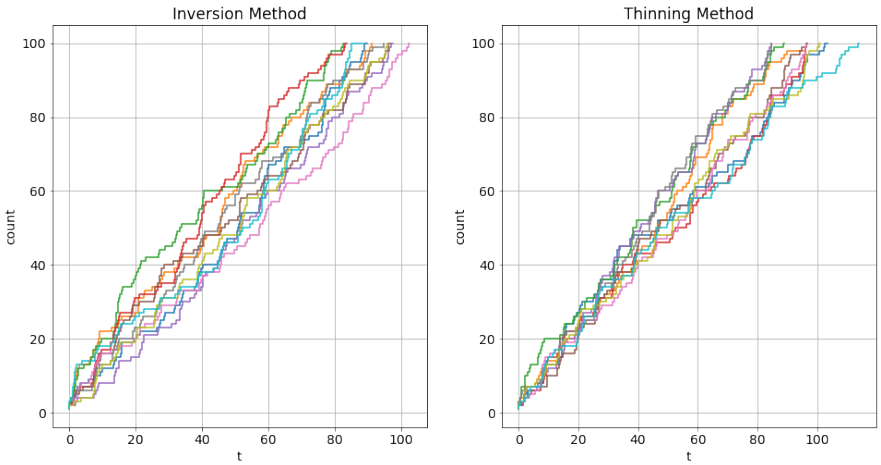

Figure 3: 10 sample paths when the base process is Poisson and with rate function $r(t) = \sin(t) + 1.01$ using both methods.

3 Weibull as the Inter-event Time Distribution

3.1 Weibull Distribution

For many applications, we would like to consider failure rates that can vary with time. The exponential distribution can model "time-to-failure" only when the failure rate is constant. The Weibull distribution generalizes this notion by allowing failure rates to decrease or increase over time [4]. The probability density function of a Weibull random variable is

$$f(x;\lambda,k) = \frac{k}{\lambda}\left(\frac{x}{\lambda}\right)^{k-1} e^{-(x/\lambda)^k}$$

for $x \geq 0$ and 0 otherwise. We characterize this distribution by shape parameter $k > 0$ and scale parameter $\lambda > 0$ [7]. When $k < 1$, the failure rate decreases over time and when $k > 1$ the failure rate increases. When $k = 1$, the failure rate is constant and the Weibull distribution reduces to an exponential with rate $1/\lambda$ [4]. This distribution has mean $\lambda\Gamma(1 + 1/k)$ and variance $\lambda^2\left(\Gamma(1 + 2/k) - (\Gamma(1 + 1/k))^2\right)$ [7].

To estimate these parameters, we can use the maximum likelihood estimators for each quantity [9]. The estimator for λ given k is

$$\widehat{\lambda}_{MLE} = \left(\frac{1}{n}\sum_{i=1}^{n} x_i^k\right)^{1/k}$$

and the estimator for k is the solution to the following equation which must be solved numerically:

$$0 = \frac{\sum_{i=1}^{n} x_i^k \ln x_i}{\sum_{i=1}^{n} x_i^k} - \frac{1}{k} - \frac{1}{n}\sum_{i=1}^{n} \ln x_i$$

3.2 Inversion Method

For the inversion method, we need to specify G such that $\tau = 1$ and $\text{cv}^2 = \sigma^2$. For Weibull Inter-event times, this means we need the following equation to hold:

$$1 = \lambda\Gamma(1 + 1/k)$$

Hence, we first fit the shape parameter k using maximum likelihood estimation. Then, given the shape parameter, we can solve for λ as such:

$$\widehat{\lambda} = \frac{1}{\Gamma(1 + 1/\widehat{k}_{MLE})}$$

Then, we have the following expression for cv^2:

$$\text{cv}^2 = \left(\frac{1}{\Gamma(1 + 1/\widehat{k}_{MLE})}\right)^2 \left(\Gamma(1 + 2/\widehat{k}_{MLE}) - \left(\Gamma(1 + 1/\widehat{k}_{MLE})\right)^2\right)$$

3.3 Thinning Method

For the thinning method, we need to specify G such that $\tau = (r^*)^{-1}$ and $\text{cv}^2 = \frac{\sigma^2}{\tau^2} = \sigma^2(r^*)^2$. For this case, τ will be calculated using the supremum of whatever rate function we choose (assumed finite). We can then use the MLE estimators for the parameters to calculate cv^2 as such:

$$\text{cv}^2 = \widehat{\lambda}^2_{MLE}\left(\Gamma(1 + 2/\widehat{k}_{MLE}) - \left(\Gamma(1 + 1/\widehat{k}_{MLE})\right)^2\right)(r^*)^2$$

4 Ransomware Case Study

4.1 Data Pre-processing

In this section, I apply the methods to the 'BitcoinHeistRansomwareAddressDataset' data set [1]. This data set can be used to explore graph based relationships between Bitcoin ransomware heists and normal Bitcoin transactions, but for my use I only need the date of the malicious transactions. This is a large data set with over $2,916,697$ data points; only $41,413$ of these transactions were known to be ransomware heists. A histogram of the malicious transactions and their inter-transaction times is presented in Figure 4.

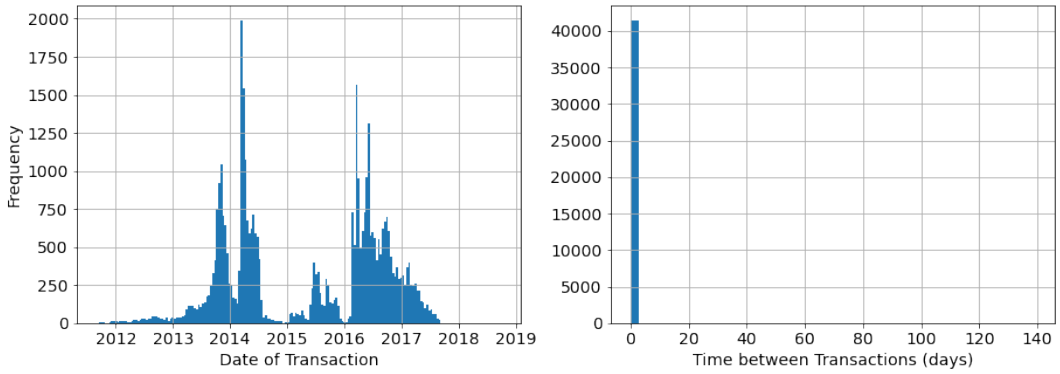

Figure 4: Histograms of Bitcoin transaction dates known to be ransomware and their inter-transaction times.

Given the large number of transactions and the fact that the data is indexed by only the date of transaction, most of the inter-transaction times are exactly zero. Semiconductor fabs are threatened with ransomware attacks less frequently, but with higher financial risks given the costs of operating such complex manufacturing systems. With this in mind, I filter the data set looking at Bitcoin transactions with Satoshi values higher than 3×10^{10} (1 bitcoin = 100 million satoshis [1]). In this filtered data set, ransomware families CryptoLocker [6], CryptoWall and Razy were responsible for 77, 12 and 3 ransomware transactions respectively.

Let $x_1, \ldots, x_{92}$ be the transaction dates for data set and let $y_1, \ldots, y_{92}$ be the corresponding inter-transaction times. Looking at the histogram for the transaction dates, I decided to partition the data into three bins named b_i for $i \in \{1, 2, 3\}$ with the first bin including transactions up to August 15th, 2013, the second bin including transactions from August 15th, 2013 up to December 15th, 2013 and the last bin including transactions on and after December 15th, 2013 as seen in Figure 5. Also, in this figure, we can see this subset of Bitcoin inter-transaction times fit to a Weibull using the MLE method on the right.

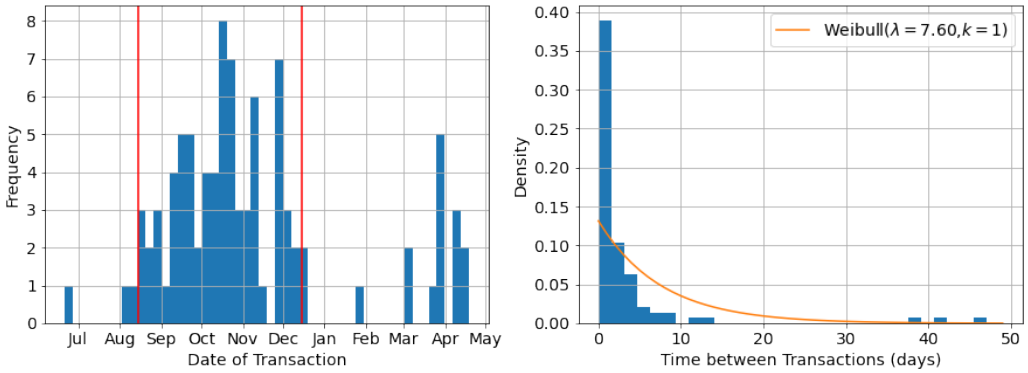

Figure 5: Histograms of 92 Bitcoin transaction dates known to be ransomware with a Satoshi amount greater than 3×10^{10} and their inter-transaction times. The dates of transactions ranges from July 2013 to May 2014.

Then, using these partitions I calculated the average rate r_i for each bin $i \in \{1, 2, 3\}$ as the sample mean:

$$r_i = \frac{1}{\mid b_i \mid} \sum_{y \in b_i} \frac{1}{y}$$

Based on this sample $y_1, \ldots, y_{92}$, we have the following rate function where t is the number of days since the data point x_1:

$$R(t) = \begin{cases} 0.056, & \text{for } t < 55 \\ 0.613, & \text{for } 55 \leq t < 178 \\ 0.124, & \text{for } t \geq 178 \end{cases}$$

Given the rate function $r(t)$ is easily integrable, I decided to use the inversion method with $G \sim$ Weibull as the inter-transaction time distribution to get 100 samples from this renewal process fit to the data set. The MLE estimate for the shape parameter was $\widehat{k}_{MLE} = 1$ meaning this Weibull reduces to an exponential distribution. Consequently, we have $\widehat{\lambda} = 1/\Gamma(1 + 1/1) = 1$. Ten sample paths of size 100 are plotted in Figure 6.

The samples follow the right behavior of the frequency of ransomware heists from mid 2013 until late 2014. Unfortunately, because the Weibull reduced to an exponential, we can not directly see the increase or decrease in variability one may expect with shape parameter $k > 1$ and $k < 1$ respectively. We can see a rapid increase in the number of heists during the middle of the simulation run which is consistent with the behavior we say during 2014. Historically, this was a period of rampant bitcoin ransomware heists.

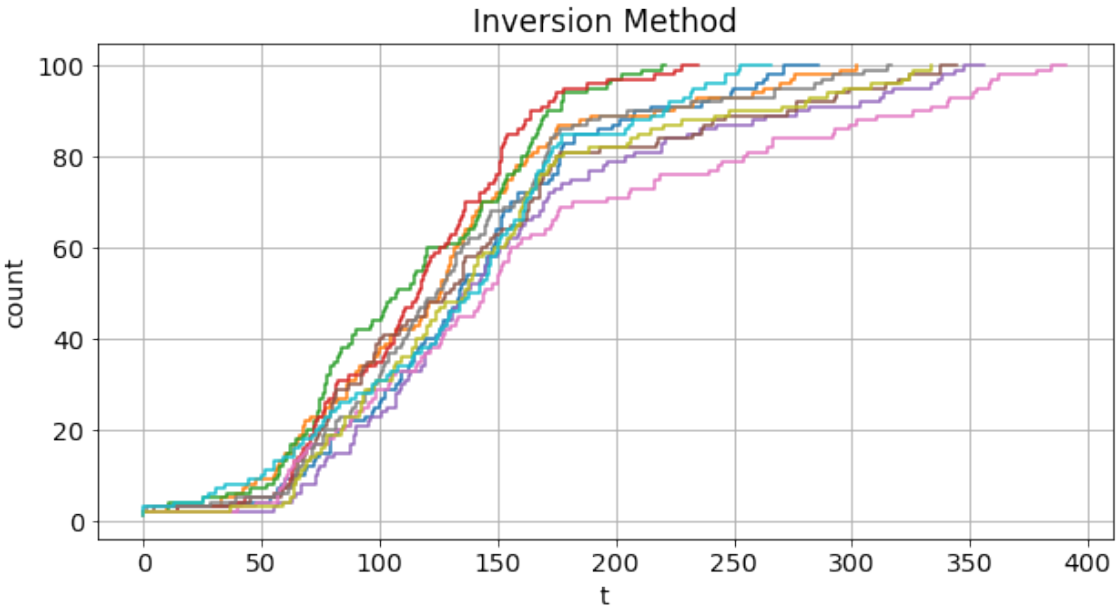

Figure 6: Ten samples paths of size 100 for the Bitcoin process modeled by Weibull inter-transactions times generated using the Inversion Method.

5 Conclusion

Modeling non-stationary renewal processes will continue to be an important task for a variety of applications. The results of this paper indicate that it is rather simple to implement algorithms that can modify non-stationary Poisson Process sampling in two ways: allowing for heterogeneous rates and inter-event distributions that are more or less variable than the exponential distribution.

Exploring my own use case and many other related data sets, I learned that many inter-event times for different processes are actually modeled by exponential distribution. Adding a heterogeneous rate to the process may improve simulation performance. In particular, for my use, it may be interesting to explore predicting different future ransomware scenarios over a particular horizon by fitting a rate function to historical data. It may be interesting to incorporate some time series forecasting for the rate function as well.

Acknowledgements

Thank you Professor Sarkar for reinvigorating my interest in statistics. I will definitely be interested in using some of the techniques I learned in this course for my graduate research.

References

[1] Cuneyt Gurcan Akcora et al. “BitcoinHeist: Topological Data Analysis for Ransomware Detection on the Bitcoin Blockchain”. In: *arXiv preprint [Web Link]* (2019).

[2] Andrew Gelman et al. “Bayesian data analysis, third edition”. In: 2013.

[3] Ira Gerhardt and Barry L. Nelson. “Transforming Renewal Processes for Simulation of Nonstationary Arrival Processes”. In: *INFORMS Journal on Computing* 21.4 (Nov. 2009), pp. 630–640. DOI: 10.1287/ijoc.1080.0316. URL: https://ideas.repec.org/a/inm/orijoc/v21y2009i4p630-640.html.

[4] R Jiang and D N P Murthy. “A study of Weibull shape parameter: Properties and significance”. en. In: *Reliab. Eng. Syst. Saf.* 96.12 (Dec. 2011), pp. 1619–1626.

[5] Vidyadhar G. Kulkarni. *Modeling and analysis of Stochastic Systems*. CRC Press, Taylor amp; Francis Group, 2017.

[6] Kevin Liao et al. “Behind closed doors: measurement and analysis of CryptoLocker ransoms in Bitcoin”. In: *2016 APWG Symposium on Electronic Crime Research (eCrime)*. 2016, pp. 1–13. DOI: 10.1109/ECRIME.2016.7487938.

[7] Athanasios Papoulis and S Unnikrishna Pillai. *Probability, random variables and stochastic processes*. en. 4th ed. McGraw-Hill series in electrical and computer engineering. Maidenhead, England: McGraw Hill Higher Education, Dec. 2001.

[8] Robert Piessens et al. “Quadpack: A Subroutine Package for Automatic Integration”. In: 2011.

[9] Abhra Sarkar. “UT Austin SDS 383C Statistical Modeling I Slides”. 2022.

Appendix A

```python
def inversion(n_iters,dist,rate,seed=0):
    """
    inversion simulates [n_iters] events from
    inter-event distribution [dist] with rate function [rate]
    using the inversion method.

    inputs:
    - dist: scipy.stats frozen statistical function
    - n_iters: number of iterations; n_iters>1
    - rate: rate function; can be heterogenous or homogenous
    - seed: np.random.seed(seed)

    outputs: (X,S,W,V)
    - X: interarrivals
    - S: arrivals
    - W: adjusted interarrivals
    - V: adjusted arrivals
    """
    start = time.time()
    np.random.seed(seed)
    # intialize parameters
    X = np.zeros(n_iters) # interarrivals
    S = np.zeros(n_iters) # arrivals
    W = np.zeros(n_iters) # adjusted interarrivals
    V = np.zeros(n_iters) # adjusted arrivals
    # step 1
    n = 1
    S[n] = dist.rvs(1)[0]
    # compute R^-1(s_1)
    V[n] = R_inv(rate,S[n],0)
    # step 2
    W[n] = V[n] - V[n-1]
    # step 3
    n += 1
    while(n < n_iters):
        X[n] = dist.rvs(1)[0]
        S[n] = S[n-1] + X[n]
        V[n] = R_inv(rate,S[n],V[n-1])
        W[n] = V[n] - V[n-1]
        n += 1
    end = time.time()
    print(f'Completed sampling in {end-start} seconds.')
    return X,S,W,V

def R_inv(rate, s, t_0, precision=0.01):
    """
    R_inv returns the lowest input within error [precision]
    such that the integral of [rate] evaluated at this input is
    greater than event time [s]. [t_0] is the starting point to
    search for such an input.

    inputs:
    - rate: rate function; can be heterogenous or homogenous
    - s: event time
    - t_0: starting value to search from
    - precision: step size for searching for lowest input

    outputs:
    - inf{t:R(t)>=s) where t in [t_0,t_0+precision,t_0+2*precision,...]
    """
    R = 0
    t = t_0
    while(s > R):
        R = integrate.quad(lambda x: rate(x), 0, t)[0]
        t += precision
    return t
```

Figure 7: Inversion method algorithm implemented in Python.

Appendix B

```python
def thinning(n_iters,dist,rate,r_max,seed=0):
    """
    thinning simulates [n_iters] events from inter-event distribution [dist]
    with rate function [rate] using the inversion method.

    inputs:
    - n_iters: number of iterations; n_iters>1
    - dist: scipy.stats frozen statistical function
    - rate: rate function; can be heterogenous or homogenous
    - r_max: maximum of rate function
    - seed: np.random.seed(seed)

    outputs: (X,S,W,V)
    - X: interarrivals
    - S: arrivals
    - Y: adjusted interarrivals
    - T: adjusted arrivals
    """
    start = time.time()
    np.random.seed(seed)
    # intialize parameters
    X = np.zeros(100*n_iters) # interarrivals
    S = np.zeros(100*n_iters) # arrivals
    Y = np.zeros(100*n_iters) # adjusted interarrivals
    T = np.zeros(100*n_iters) # adjusted arrivals
    # step 1
    n = 1
    k = 1
    S[n] = dist.rvs(1)[0]
    # step 2
    if stats.uniform(0,1).rvs(1)[0] <= (rate(S[n])/r_max):
        T[n] = S[n]
        Y[n] = T[n]-T[n-1]
        k = 2
    # step 3
    n += 1
    while(k < n_iters):
        X[n] = dist.rvs(1)[0]
        S[n] = S[n-1] + X[n]
        # step 4
        if stats.uniform(0,1).rvs(1)[0] <= (rate(S[n])/r_max):
            T[k] = S[n]
            Y[k] = T[k] - T[k-1]
            k += 1
        n += 1
    end = time.time()
    print(f'Completed sampling in {end-start} seconds.')
    Y,T = np.trim_zeros(Y,'b'), np.trim_zeros(T,'b')
    return X,S,Y,T
```

Figure 8: Thinning method algorithm implemented in Python.