Simulating Multivariate Poisson

Jacob W Shusko (EID: JWS4654)

December 2021

Abstract

In this paper, I review an adjusted NORTA-based method to generate p-dimensional Poisson vectors with a mean (rate) vector Λ and known correlation R_{pois}. The method first generates p-dimensional Normal vectors and uses these vectors to calculate samples distributed as Poisson with different rates and positive or negative correlations. The correlation matrix R_N for the p-dimensional Normal vector is adjusted using an exponential relationship to better match the desired correlation matrix R_{pois}. Bivariate results indicate the adjusted method improves on the unadjusted NORTA method in terms of mean squared errors over small lambda values, but not large values.

1 Introduction

Simulating systems is a tool used across many industries to assess key metrics and better understand the dynamics of a system. Due to an increase in computational power in the last 20 years, simulating more complex systems is becoming a possibility. Typically, most simulation projects as of date have strayed away from multivariate random variable simulations. This is due to the challenges in simulating general multivariate distributions. The multivariate Normal distribution is the most common multivariate distribution given it is a good approximation for lots of phenomena and it is relatively easy to simulate using Cholesky factors and a simple linear transformation. Unfortunately, the Normal distribution does not model all phenomena.

In particular, the Normal distribution would be an unrepresentative model for the number of cyber-security attacks over a given period. In this case, we would like to model attacks from different sources arriving according to Poisson processes with different rates. Then, the number of attacks from different sources in fixed time period is a multivariate Poisson random variable. The Poisson distribution models the number of rare events that occur during a specific time window, so this is a natural choice for cyber-security attacks. If each component of this Poisson distribution (each source of attack) is completely independent of each other, then simulating this multivariate random variable is relatively easy. However, I would like to include the possibility that particular sources may be working against or with each other. One way to approximate

this dependence is to specify a covariance matrix for this random variable. It is important to note that covariance only captures a first order or linear notion of dependency.

2 Method

There has been much research in developing a method to generate multivariate Poisson random variables. Most of this research has been focused on the bivariate case, which is not sufficient for our use-case. In addition, most of these methods do not allow for negative correlations or even different correlations between different components, which again is not sufficient for modeling cyber-security attacks. In this paper, I will review an adapted NORTA model [1] that allows negative correlations, dimension $p > 2$ and unique correlations between bivariate components. In Yahav and Shumeli's paper [1], they properly summarize the limitations of the previous methods. Considering these methods won't work for our use-case, I will not study their performance, but instead will verify the results of their adapted NORTA technique.

2.1 NORmal TO Anything

The NORTA method is used to generate general multivariate distributions characterized by univariate marginals and a pre-specified covariance matrix. Intuitively, the idea is to generate a p dimensional vector from a multivariate Standard Normal distribution with correlation structure R_N and then transform it to any desires distribution using the inverse cumulative distribution function. The output distribution is know as a normal-copula. Note, since we are generating a p-vector from the Standard Normal distribution (means are 0 and variances are 1), the covariance matrix is equal to the correlation matrix. Hence, from here on we will use the correlation matrix as a measure of linear dependency. The unadjusted NORTA method is as follows:

1. Generate a p-dimensional Normal vector X_n with mean $\mu = 0$ and variance $\sigma = 1$ and correlation matrix R_n.

2. For each value X_{N_i}, $i \in \{1, 2, \ldots, p\}$ calculate the normal CDF:

$$\Phi(X_{N_i})$$

3. For each $\Phi(X_{N_i})$, calculate the Poisson inverse CDF (quantile) with rate λ_i:

$$X_{pois_i} = \Xi^{-1}(\Phi(X_{N_i}))$$

where

$$\Phi(x) = \int_{-\infty}^{x} \frac{1}{\sqrt{2\sigma^2}} e^{\frac{-u^2}{2}} du$$

and

$$\Xi(x) = \sum_{i=0}^{x} \frac{e^{-\lambda}\lambda^i}{i!}$$

.

Then, we have X_{pois} as a p-dimensional Poisson vector with correlation matrix R_{pois} and rates Λ. We know that when Λ is high, the Poisson distribution is asymptotically Normal and thus $R_{pois} \approx R_N$. However, when one or more of the rates is low, the output correlation deviates from the Normal correlation. This occurs because the feasible region is now smaller than $[-1, 1]$ and after this transformation the region is [2]:

$$[\underline{\rho} = corr(\Xi_{\lambda_1}^{-1}(U), \Xi_{\lambda_2}^{-1}(1-U)), \bar{\rho} = corr(\Xi_{\lambda_1}^{-1}(U), \Xi_{\lambda_2}^{-1}(U))]$$

2.2 Adapting the NORTA approach

We can use this fact to adjust the previous NORTA method to get a normal-copula correlation matrix closer to the desired correlation vector. Yahav and Shmueli reviewed previous methods to solve the NORTA correlation matching problem when distributions are discrete, but chose to use a simpler parametric approximation to map R_N to R_{pois}. An existing method from Avarmidis [3] utilizes a bivariate normal integral and the approximation of the derivative of the matching function with respect to the actual correlation, which will clearly be more computationally intensive then using a simple parametric function. Exploring many different parametric models (exponential, double exponential, linear, quadratic, polynomial), Goodness-of-fit tests indicated that the exponential relationship has the best performance (using root mean square error as a metric) [1].

The authors developed this method for a bivariate Poisson random variable. Based on their simulation study, they found the relationship between the desired correlation (ρ_{pois}) and the actual correlation (ρ_N) is best approximated by the exponential function:

$$\rho_{pois} = a \times e^{b\rho_N} + c$$

The coefficients a, b and c can be estimated from the points $(\underline{\rho}, -1), (\bar{\rho}, 1)$ and $(0, 0)$:

$$a = -\frac{\underline{\rho} \times \bar{\rho}}{\underline{\rho} + \bar{\rho}}$$

$$b = log\left(\frac{\bar{\rho} + a}{a}\right)$$

$$c = -a$$

Now, we can state the full adapted NORTA method for the general p-dimensional case. The only change here from their method is that I loop through each pair of components and adjust the correlation.
Input: Rate vector Λ and desired correlation matrix R_{pois}.
Output: p-dimensional vector X_{pois} with rates Λ and correlation matrix approximately equal to R_{pois}.

1. Compute $\underline{\rho}, \bar{\rho}, a, b$ and c.

2. For each pair of components: Compute the initial correlation from equation
$$\rho_{Ni} = \frac{log(\frac{\rho_{pois_i} - c)}{a}}{b}$$

3. Generate bivariate Normal data X_N with parameters $\mu = 0$, $\sigma = 1$ and correlation matrix R_N where
$$R_N(i,j) = 1 : \forall i = j \in \{1, 2, \ldots, p\}$$
and
$$R_N(i,j) = \rho_{Ni} : \forall i \neq j \in \{1, 2, \ldots, p\}$$
.

4. For each value X_{N_i}, $i \in \{1, 2, \ldots, p\}$ calculate the normal CDF:
$$\Phi(X_{N_i})$$

5. For each $\Phi(X_{N_i})$, calculate the Poisson inverse CDF (quantile) with rate λ_i:
$$X_{pois_i} = \Xi^{-1}(\Phi(X_{N_i}))$$

3 Results

To evaluate the performance of this model, I implemented the algorithm in Python code and ran experiments on my MacBook Pro (8-Core Intel Core i9 at 2.3 GHz with 32 GB of memory) with I first compared the results using the adjusted and unadjusted NORTA method under five bivariate experiments that were performed in Yahav and Shmueli simulation. These experiments test the method for lambda values under 1, so I performed a few additional experiments to see the results when lambda is larger. Then, I verified that the univariate marginals were empirically looking Poisson.

3.1 Small Lambda Experiments

In this section, I run the method on five experiments ranging the lambda values (λ_1, λ_2): $(0.1, 0.1), (0.1, 0.5), (0.5, 0.5), (0.5, 0.9)$ and $(0.9, 0.9)$. For each pair of lambda values, I run 50 trials to get 50 2-dimensional Poisson vectors. The results indicate that the adjusted method improves on the unadjusted NORTA method for positive correlations although for both methods there is room for improvement. In Yahav and Shmueli's paper, they demonstrate these results over the correlation range $[0, 0.5]$ and the trend in their figure seems to indicate the method is performing well. I expanded this range to $[-1, 1]$ and it appears the method has unexpected behavior for small and negative correlations.

As we can see in the figure 1, in both the adjusted and unadjusted models, there is strange behavior about 0. For small positive correlations around 0.1, the method outputs a Poisson vector with a slightly negative correlation around -0.1. For small negative correlations around -0.1, the same phenomena occurs with the actual correlation being around 0.1. In general when the desired correlation is negative, the actual correlation appears to be slightly greater in both methods. And, when the desired correlation is positive, the actual correlation is smaller than the desired.

(λ_1, λ_2)	(0.1,0.1)	(0.1,0.5)	(0.5,0.5)	(0.5,0.9)	(0.9,0.9)
Unadjusted MSE	0.159	0.138	0.071	0.059	0.043
Adjusted MSE	0.154	0.124	0.045	0.039	0.028
Delta MSE	0.005	0.014	0.026	0.02	0.015
Unadjusted Sample Time	0.00048	0.00041	0.00046	0.00042	0.0005
Adjusted Sample Time	0.38	0.5	0.62	0.72	0.82
Delta Sample Time	-0.37952	0.49959	-0.61954	-0.71958	-0.8195

Table 1. Performance metrics for small lambda experiments.

Empirically, it appears the adjusted method is more sensitive to the values of lambda. For small values of lambda, the mean square error between the actual and desired correlation is almost four times the MSE for larger values of lambda as seen in the table 1. In general the adjusted model has lower error over all experiments. This is at the expense of time. For most experiments, the time to compute one 2-dimensional vector for the adjusted method was around 0.5 seconds, while the unadjusted method only took about 0.0004 seconds. Hence, if the improvement in error is marginal as compared to the unadjusted method, it may make sense to just use the unadjusted method.

3.2 Large Lambda Experiments

(λ_1, λ_2)	(1,1)	(1,10)	(10,10)	(10,100)	(100,100)
Unadjusted MSE	0.0359	0.0207	0.006	0.00572	0.00494
Adjusted MSE	0.0325	0.0248	0.0336	0.0331	0.0362
Delta MSE	0.0004	-0.0041	-0.0276	-0.02738	-0.03126
Unadjusted Sample Time	0.0003	0.0003	0.0003	0.00031	0.00031
Adjusted Sample Time	0.815	1.095	1.383	3.493	1.60
Delta Sample Time	-0.8147	-1.0947	-1.3827	-3.49269	-1.59969

Table 2. Performance metrics for large lambda experiments.

In addition, I also ran this method on five experiments with higher values of (λ_1, λ_2): $(1,1), (1,10), (10,10), (10,100)$ and $(100,100)$. The adjusted method only outperformed the unadjusted method when lambdas were $(1,1)$. When one of the lambdas were greater than 10, the unadjusted model always performed better. This makes sense considering the Poisson distribution is better

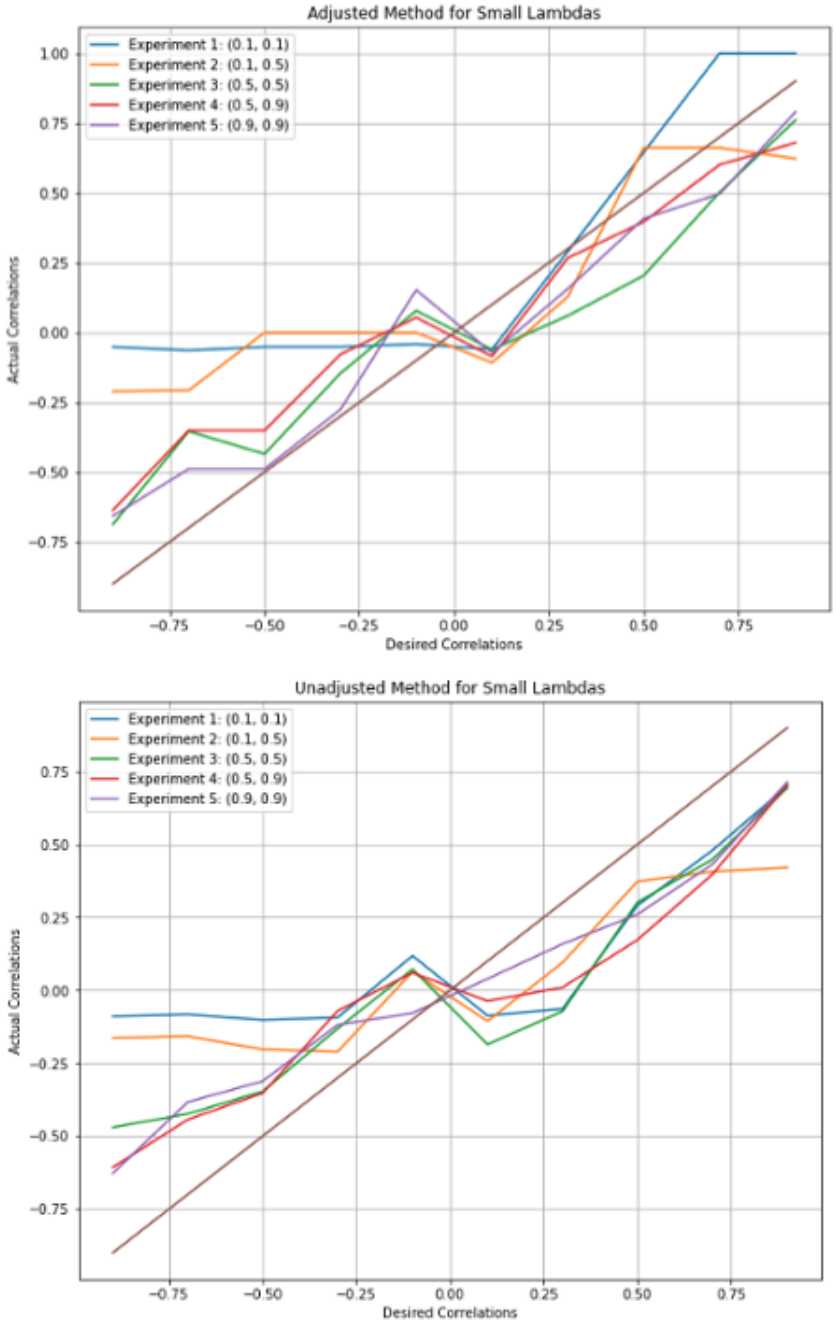
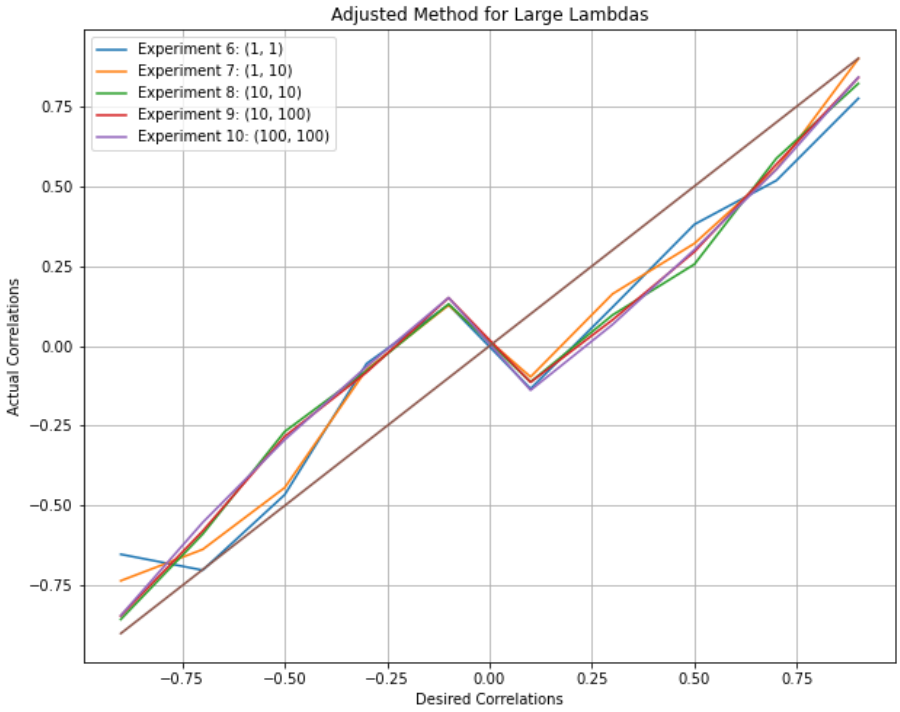


Figure 1: Actual versus desired correlation for small lambda experiments.



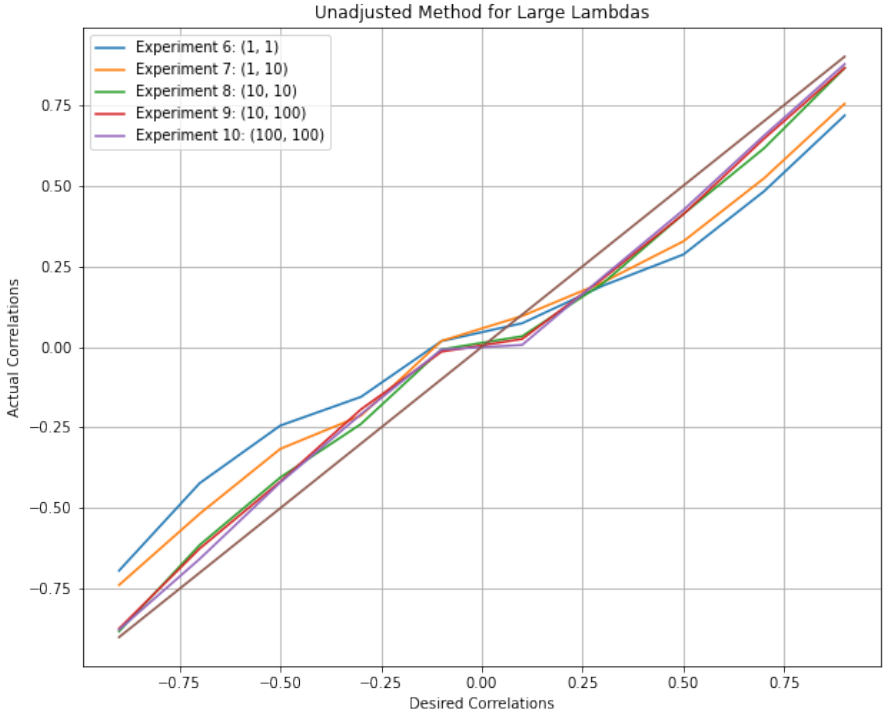


Figure 2. Actual versus expected for large lambda experiments.

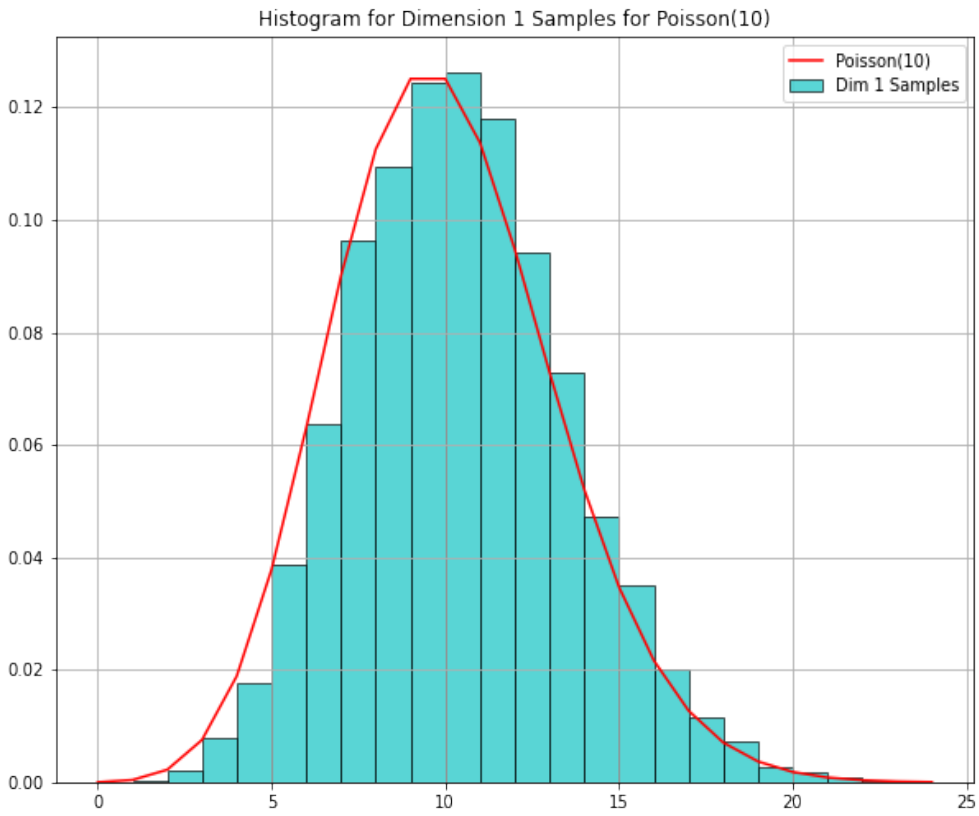


Figure 3. Histogram of dimension 1 univariate marginal Poisson(10) samples.

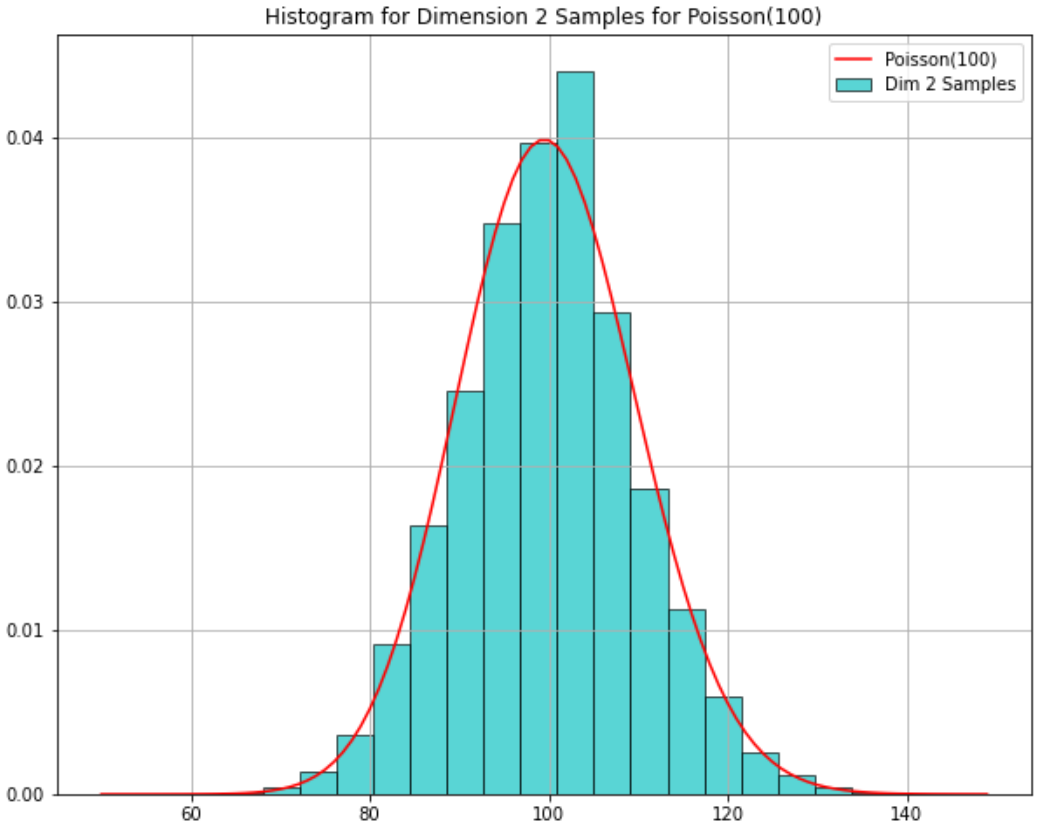


Figure 4. Histogram of dimension 2 univariate marginal Poisson(100) samples.

approximated by the Normal distribution when lambda is larger. Effectively, the modification to the correlation matrix is exactly adjusting for this property. As seen in table 2, as the lambdas get larger the difference between the adjusted and unadjusted mean square error grows larger. Again, this is because the Normal approximates the Poisson for high lambda rates.

Then, I ran 10,000 samples of a 2-dimensional Poisson vector with rates (10,100) and a correlation of 0.4. Looking at figures 3 and 4, it is apparent that the univariate marginals follow the corresponding Poisson quite well.

4 Conclusion

Overall, I think this method is an effective way to generate correlated Poissons with decent accuracy. If all of the lambdas provided are greater than 10, it seems it makes sense to use the unadjusted method. The correlation accuracy is actually better when using the unadjusted method for lambdas higher than 10. Given the adjusted method takes much longer to sample, I would just use the unadjusted method for these high values. When one of the lambdas is less than 1, I would opt to use the adjusted model given we can get a marginal gain in accuracy. If speed is the main priority, however, then we should just use the basic NORTA method for all cases.

With more time, I would like to look into the different ways to solve the correlation matching problems. The NORTA method seems to be the best 'base' method to generate Poissons. If I can design a faster way to solve the correlation mapping problem, or at least one that provides more accuracy over the entire range of lambdas, this would be an interesting improvement.

References

[1] Yahav, I.; Shmueli, G. (2011). On generating multivariate Poisson data in management science applications. Applied Stochastic Models in Business and Industry, 28(1), 91–102. https://doi.org/10.1002/asmb.901

[2] Ward Whitt. "Bivariate Distributions with Given Marginals." Ann. Statist. 4 (6) 1280 - 1289, November, 1976. https://doi.org/10.1214/aos/1176343660

[3] Avramidis, Athanassios Channouf, Nabil L'Ecuyer, Pierre. (2009). Efficient Correlation Matching for Fitting Discrete Multivariate Distributions with Arbitrary Marginals and Normal-Copula Dependence. INFORMS Journal on Computing. 21. 88-106. 10.1287/ijoc.1080.0281.

5 Code

```
import numpy as np
import pandas as pd
```

```
from scipy.stats import multivariate_normal as mvn
from scipy.stats import norm
from scipy.stats import poisson
from scipy.stats import pearsonr

def generateMultivariatePoisson(p,R,lambdas,adj=True,seed=1):
    """
    generateMultivariatePoisson generates a sample of a [p]-dimensional Poisson
    marginal rates [lambdas] and p x p correlation matrix [R] where R(i,j)=1 for
    NORTA approach.
    - calculate feasible correlation range
    - computer parameters for correlation mapping and apply
    - generate multivariate normals with means 0 and variances 1 with an adjuste
    - apply the inverse cdf of N(0,1) to each component of the multivariate samp
    - for each generated U(0,1),
        input this value into the inverse of the poisson parametrized by the cor
    - output p-dim poisson vector with rates lambdas
    """
    np.random.seed(seed)
    # adjust correlations by applying mapping to each pair of bivariates
    corrected = np.copy(R)

    if adj:
        for i in range(p):
            for j in range(p):
                if i != j:
                    np.random.seed(seed)
                    # identify feasible correlation range
                    u = np.random.uniform(size=100000)
                    x = poisson.ppf(u,mu=lambdas[i])
                    y = poisson.ppf(u,mu=lambdas[j])
                    z = poisson.ppf(1-u,mu=lambdas[j])
                    maxcor = pearsonr(x,y)[0] # pearson r outputs (corr coef, p-
                    mincor = pearsonr(x,z)[0]

                    # compute parameters for mapping
                    a = -maxcor * mincor / (maxcor + mincor)
                    b = np.log((maxcor+a)/a)
                    c = -a

                    # apply exponential mapping
                    orgCorr = R[i,j]
                    adjCorr = np.log((orgCorr-c)/a)/b

                    if (adjCorr <= 1) and (adjCorr >= -1):
                        corrected[i,j] = adjCorr
```

```
                corrected[j,i] = adjCorr

    # run the NORTA method using an adjusted covariance matrix
    normal_mu = np.zeros(p)
    normal = np.random.multivariate_normal(mean=normal_mu,cov=corrected)
    unif = norm.cdf(normal)
    pois = poisson.ppf(unif,mu=lambdas)
    return pois
```