Simulate Power Law Process using R

Miao Cai[*]

2019-05-26

Contents

[*]Department of Epidemiology and Biostatistics, College for Public Health and Social Justice, Saint Louis University. Email: miao.cai@slu.edu

1 A few concepts

Mean function of a point process:

$$\Lambda(t) = E(N(t))$$

$\Lambda(t)$ is the expected number of failures through time t.

Rate of Occurence of Failures (ROCOF): When Λ is differentiable, the ROCOF is:

$$\mu(t) = \frac{d}{dt}\Lambda(t)$$

The ROCOF can be interpreted as the instantaneous rate of change in the expected number of failures.

Intensity function: The intensity function of a point process is

$$\lambda(t) = \lim_{\Delta t \to 0} \frac{P(N(t, t + \Delta t] \geq 1)}{\Delta t}$$

When there is no simultaneous events, ROCOF is the same as intensity function.

2 NHPP and PLP

Nonhomogeneous Poisson Process (NHPP): The NHPP is a Poisson process whose intensity function is non-constant.

Power law process (PLP): When the intensity function of a NHPP is:

$$\lambda(t) = \frac{\beta}{\theta}\left(\frac{t}{\theta}\right)^{\beta-1}$$

Where $\beta > 0$ and $\theta > 0$, the process is called the power law process (PLP).

Therefore, the mean function $\Lambda(t)$ is the integral of the intensity function:

$$\Lambda(t) = \int_0^t \lambda(t)dt = \int_0^t \frac{\beta}{\theta}\left(\frac{t}{\theta}\right)^{\beta-1} = \left(\frac{t}{\theta}\right)^{\beta}$$

3 Simulate PLP using the inverse algorithm

3.1 Theory of the inverse algorithm

The algorithm used in this tutorial is the review provided by Pasupathy (2010). He mentioned that the earliest inversion technique was devised by Cinlar (2013), which was based on a property of NHPP with a continuous expectation function $\Lambda(t)$.

Theorem 3.1 (Cinlar's inversion algorithm)**.** *Let $\Lambda(t)$ be a positive-valued, continuous, nondecreasing function. Then the random variables $T_1, T_2, \cdots$ are event times corresponding to a NHPP with expectation function $\Lambda(t)$ if and only if $\Lambda(T_1), \Lambda(T_2) \cdots$ are the event times corresponding to a HPP with rate one.*

Theorem 3.1 can be used to generate failures from a NHPP: First generate event times from a HPP with rate one, then invert $\Lambda()$ to get the event times. Here are the steps/algorithms provided by Pasupathy (2010) to generate NHPP failure times.

(0) Initialize $s = 0$
(1) Generate u $U(0, 1)$
(2) Set $s \leftarrow s - \log(u)$
(3) Set $t \leftarrow \inf\{\Lambda(v) \geq s\}$
(4) Deliver t
(5) Go to Step (1)

Here the Step (3) is essentially getting the inverse of $\Lambda(v)$ if $\Lambda(v)$ is a continuous function.

$$
\begin{aligned}
s = \Lambda(t) &= \left(\frac{t}{\theta}\right)^{\beta} \\
s^{1/\beta} &= \frac{t}{\theta} \\
\theta \cdot s^{1/\beta} &= t
\end{aligned}
$$

Therefore, the inverse of $s = \Lambda(t)$ is $t = \theta \cdot s^{1/\beta}$

3.2 Failure and time truncated cases

The above algorithm can be tuned into either *failure truncated case* or *time truncated case*.

- **failure truncated case**: Let N denote the number of failures and we can set the number of iterations in the algorithm as N.
- **time truncated case**: Let τ denote the truncation time. We can iterate the algorithm for m times until the next s is greater than τ.

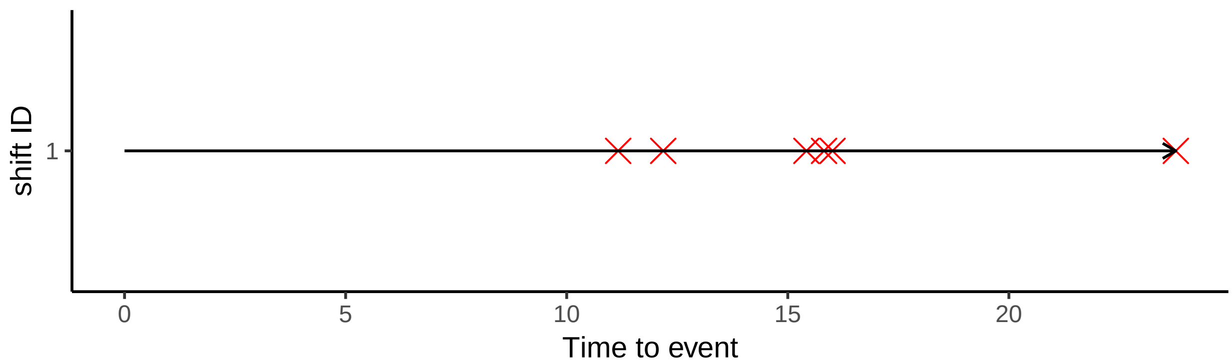

Figure 1: Arrow plot of simulated failures in a failure trucated case

4 Simulation using R

4.1 The most straightforward way of simulation - a loop

In this simulation, we randomly set parameters $\beta = 2, \theta = 10$.

4.1.1 failure truncation

```
set.seed(123)
s = 0; N = 6; t = rep(NA_real_, N) #initialization
beta = 2; theta = 10 # random parameters

for (i in 1:N) {
  u = runif(1)
  s = s - log(u)
  t[i] = theta*s^(1/beta)
}

t
```

```
## [1] 11.16361 12.18250 15.42151 15.81973 16.01255 23.77566
```

An arrow plot of these failures is shown in Figure 1.

Since $\beta = 2 > 1$, the reliability of this system is deteriorating. The failures become more and more intense/frequent at the right side of the plot.

4.1.2 Time truncation

In this time truncation case, I still set $\beta = 2$, $\theta = 10$. The truncation time is set as 30 here.

```
set.seed(123)
sim_plp_tau = function(tau = 30,
                       beta = 2,
                       theta = 10){
  # initialization
   s = 0; t = 0
  while (max(t) <= tau) {
    u <- runif(1)
    s <- s - log(u)
    t_new <- theta*s^(1/beta)
```

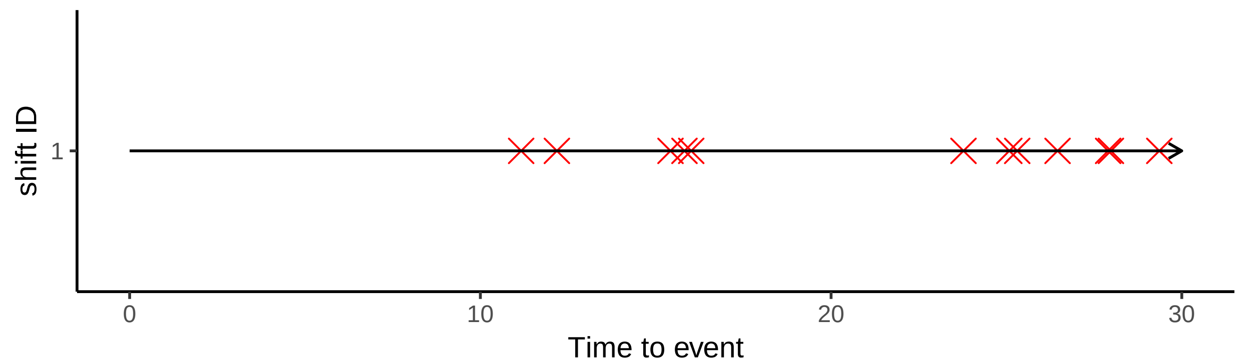

Figure 2: Arrow plot of simulated failures in a time trucated case

```
    t <- c(t, t_new)
  }
  t = t[c(-1, -length(t))]

  return(t)
}

t = sim_plp_tau()
t
```

```
##  [1] 11.16361 12.18250 15.42151 15.81973 16.01255 23.77566 25.08242
##  [8] 25.30830 26.45814 27.90026 27.97923 29.35898
```

An arrow plot of these simulated failures is shown in 2.

4.2 A more efficient way of simulation - vectorized function

We don't have to use a loop to iteratively sample as this algorithm does. Instead, we can use a vectorized form in R to simulate event times using this algorithm. This is only available for the failure truncated case since the number of iterations is known.

```
sim_plp_n = function(mean_n, beta, theta){
  N = rpois(1, mean_n)
  u = runif(N, 0, 1)
  n_logu = -log(u)
  s = cumsum(n_logu)
  Delta_t = theta*s^(1/beta)
  return(Delta_t)
}

set.seed(666)
t1 = sim_plp_n(mean_n = 6, beta = 2, theta = 10)
t1
```

```
## [1] 12.74133 12.82827 18.02351 20.65592 21.36411 21.41437 22.98408 30.98626
```

An arrow plot of these failures is shown in Figure 3.

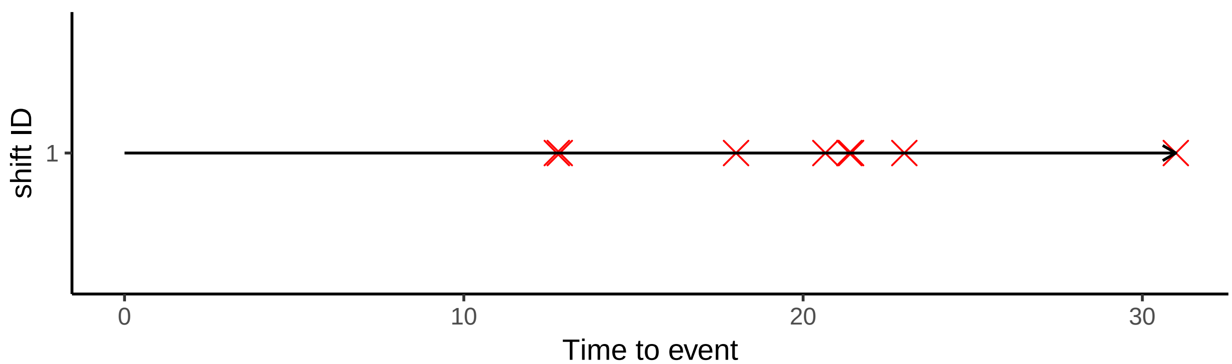

Figure 3: Arrow plot of simulated failures in a failure trucated case (vectorized form)

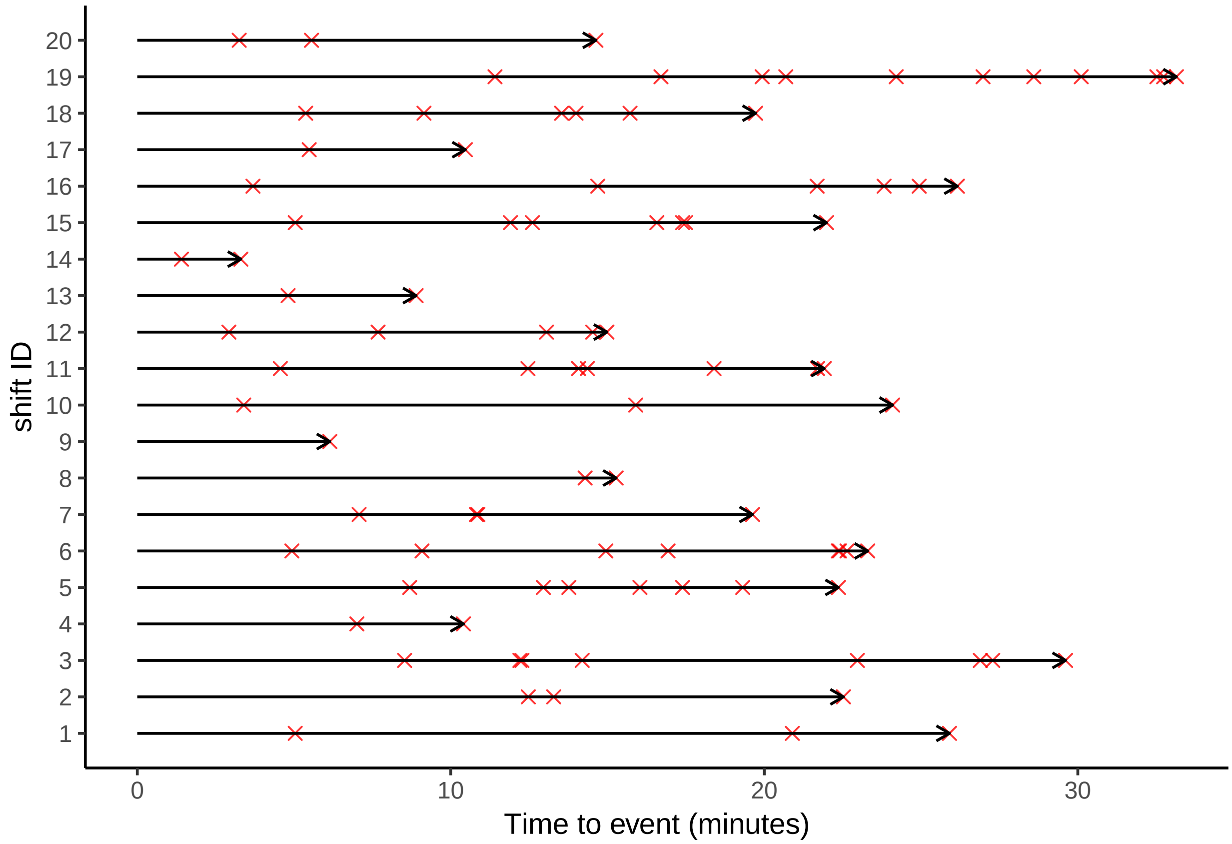

Figure 4: Arrow plot of simulated failures in multiple failure trucated cases

4.3 Simulating multiple simulations

In this multiple shifts simulation, we still keep $\beta = 2, \theta = 10$.

4.3.1 Failure truncated case

```
## List of 3
##  $ event_dat     :'data.frame':    92 obs. of  2 variables:
##   ..$ shift_id  : int [1:92] 1 1 1 2 2 2 3 3 3 3 ...
##   ..$ event_time: num [1:92] 5.04 20.89 25.91 12.47 13.28 ...
##  $ start_end_dat:'data.frame':    20 obs. of  3 variables:
##   ..$ shift_id  : int [1:20] 1 2 3 4 5 6 7 8 9 10 ...
##   ..$ start_time: num [1:20] 0 0 0 0 0 0 0 0 0 0 ...
##   ..$ end_time  : num [1:20] 25.9 22.5 29.6 10.4 22.4 ...
##  $ shift_length : int [1:20] 3 3 8 2 7 8 4 2 1 3 ...
```

A plot for these simulated multiple shifts is shown in Figure 4.

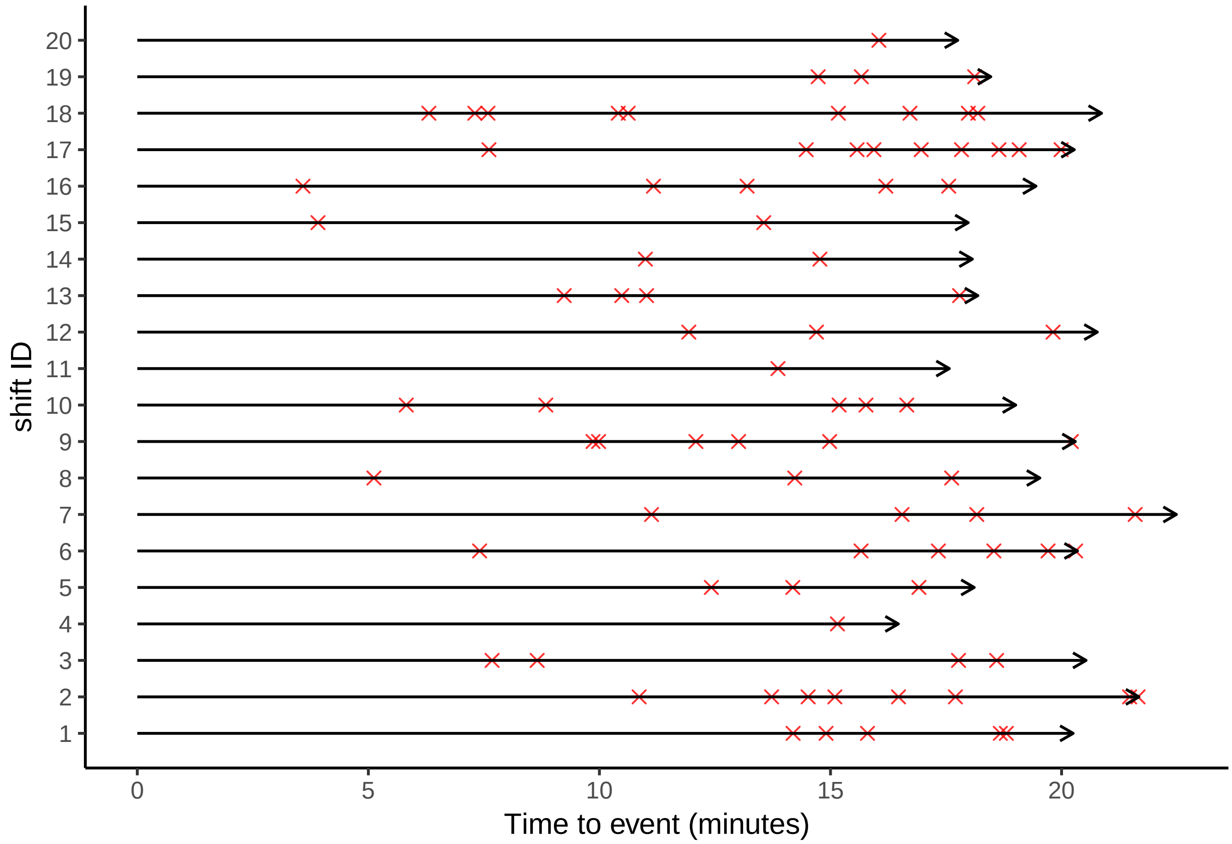

Figure 5: Arrow plot of simulated failures in **multiple** time trucated cases

4.3.2 Time truncated case

```
## List of 3
##  $ event_dat    :'data.frame':   84 obs. of  2 variables:
##   ..$ shift_id  : int [1:84] 1 1 1 1 1 2 2 2 2 2 ...
##   ..$ event_time: num [1:84] 14.2 14.9 15.8 18.7 18.8 ...
##  $ start_end_dat:'data.frame':   20 obs. of  3 variables:
##   ..$ shift_id  : int [1:20] 1 2 3 4 5 6 7 8 9 10 ...
##   ..$ start_time: num [1:20] 0 0 0 0 0 0 0 0 0 0 ...
##   ..$ end_time  : num [1:20] 20.3 21.7 20.5 16.5 18.1 ...
##  $ shift_length : int [1:20] 5 8 4 1 3 6 4 3 6 5 ...
```

An arrow plot of these simulated multiple shifts is shown in Figure 5.

References

Cinlar, Erhan. 2013. *Introduction to Stochastic Processes.* Courier Corporation.

Pasupathy, Raghu. 2010. "Generating Homogeneous Poisson Processes." *Wiley Encyclopedia of Operations Research and Management Science.*