

Jump-point PLP (JPLP) simulation

Miao Cai miao.cai@slu.edu

2020-04-08

Contents

1	Power law process (PLP)	1
1.1	PLP intensity function	1
1.2	PLP simulation	1
2	Jump Power Law Process (JPLP)	4
2.1	JPLP intensity function	4
2.2	JPLP simulation	4

1 Power law process (PLP)

1.1 PLP intensity function

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

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

where $\beta > 0$ and $\theta > 0$, the process is called the power law process (PLP). The mean function $\Lambda(t)$ is the integral of the intensity function:

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

1.2 PLP simulation

```
# simulating PLP - time truncated case
sim_plp_tau = function(tau = 30,
                        beta = 1.5,
                        theta = 10){
  # initialization
  s = 0; t = 0
  while (max(t) <= tau) {
    u <- runif(1)
    s <- s - log(u)
    t_new <- theta*s^(1/beta)
    t <- c(t, t_new)
  }
}
```

```

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

return(t)
}

# simulate multiple NHPPs - time truncated case
sim_mul_plp_tau = function(n_shift = 20,
                           shift_len_mean = 20, shift_len_sd = 5,
                           theta = 10, beta = 2, mean_n = 5){
  tau_vector = rnorm(n_shift, shift_len_mean, shift_len_sd) #difference1

  t_list = list()
  len_list = list()
  # end_time1 = list() # not needed for time truncated case

  for (i in 1:n_shift) {
    t_list[[i]] = sim_plp_tau(tau_vector[i], beta, theta)
    len_list[[i]] = length(t_list[[i]])
  }

  event_dat = data.frame(
    shift_id = rep(1:n_shift, unlist(len_list)),
    event_time = Reduce(c, t_list)
  )

  start_end_dat = data.frame(
    shift_id = 1:n_shift,
    start_time = rep(0, n_shift),
    end_time = tau_vector #difference2
  )

  return(list(event_dat = event_dat,
              start_end_dat = start_end_dat,
              shift_length = unlist(len_list)))
}

sim_hier_plp_tau = function(N, beta = 1.5, theta){
  t_list = list()
  len_list = list()
  tau_vector = rnorm(N, 10, 1.3)

  for (i in 1:N) {
    t_list[[i]] = sim_plp_tau(tau_vector[i], beta = beta, theta = theta[i])
    len_list[[i]] = length(t_list[[i]])
  }
}

```

```

}

event_dat = data.frame(
  shift_id = rep(1:N, unlist(len_list)),
  event_time = Reduce(c, t_list)
)

start_end_dat = data.frame(
  shift_id = 1:N,
  start_time = rep(0, N),
  end_time = tau_vector #difference2
)

return(list(event_dat = event_dat,
            start_end_dat = start_end_dat,
            shift_length = unlist(len_list)))
}

plot_est = function(data, var = "beta", hline_var = 1.5){
  p = data %>%
    filter(term == var) %>%
    ggplot(aes(id, est_mean)) +
    geom_point() +
    geom_line(linetype = "dashed", color = "red")+
    geom_errorbar(aes(ymax = est_mean + 1.96*est_sd,
                      ymin = est_mean - 1.96*est_sd),
                  width = 1)+
    geom_segment(aes(x = 10, xend = 100,
                     y = hline_var, yend = hline_var),
                 color = "green")+
    scale_x_continuous(breaks = c(0, 10, 25, 50, 75, 100),
                       labels = c("0", "10", "25", "50", "75", "100")) +
    labs(x = "The number of drivers (random effects)",
         y = var) +
    theme_bw()
  return(p)
}

# plot events
plot_events = function(event_dat, start_end_dat, cross_size = 2){
  p = event_dat %>%
    ggplot(aes(x = event_time, y = shift_id)) +
    geom_point(alpha = 0.8, shape = 4, color = 'red', size = cross_size) +
    scale_y_continuous("shift ID",
                       labels = as.character(start_end_dat$shift_id),
                       breaks = start_end_dat$shift_id)+
    xlab('Time to event (minutes)') +

```

```

geom_segment(data = start_end_dat,
             aes(x = start_time, xend = end_time,
                 y = shift_id, yend = shift_id),
             lineend = 'butt',
             arrow = arrow(length = unit(0.2, "cm")))) +
theme_classic()
return(p)
}

```

2 Jump Power Law Process (JPLP)

2.1 JPLP intensity function

A Bayesian hierarchical JPLP has the following intensity function:

$$\begin{aligned}
\lambda_{\text{JPLP}}(t|d, s, r, \beta, \gamma_{0,d}, \gamma, \mathbf{X}_d, \mathbf{W}) &= \begin{cases} \kappa^0 \lambda(t|\beta, \gamma_{0,d}, \gamma, \mathbf{X}_d, \mathbf{W}) & 0 \leq t \leq a_{d,s,1} \\ \kappa^1 \lambda(t|\beta, \gamma_{0,d}, \gamma, \mathbf{X}_d, \mathbf{W}) & a_{d,s,1} \leq t \leq a_{d,s,2} \\ \dots & \dots \\ \kappa^{R-1} \lambda(t|\beta, \gamma_{0,d}, \gamma, \mathbf{X}_d, \mathbf{W}) & a_{d,s,R-1} \leq t \leq a_{d,s,R} \end{cases} \\
&= \kappa^{r-1} \lambda(t|d, s, r, \kappa, \beta, \gamma_{0,d}, \gamma, \mathbf{X}_d, \mathbf{W}) \quad a_{d,s,r-1} \leq t \leq a_{d,s,r},
\end{aligned} \tag{1}$$

where the introduced parameter κ is the percent of intensity function recovery once the driver takes a break. We assume that this κ is constant across drivers and shifts.

2.2 JPLP simulation

Let T_1, T_2, \dots be random variables representing the event times of a nonhomogeneous Poisson process with continuous expectation function $\Lambda(t)$, and let N_t represent the total number of events occurring before time t in the process. Then, conditional on the number of events $N_{t_0} = n$, the event times T_1, T_2, \dots, T_n are distributed as order statistics from a sample with distribution function $F(t) = \Lambda(t)/\Lambda(t_0)$ for $t \in [0, t_0]$.

This is a generalization of the result for homogeneous Poisson processes. It naturally gives rise to the following algorithm for generating random variates from a nonhomogeneous Poisson process with expectation function $\Lambda(t)$ in a fixed interval $[0, t_0]$.

- (1) Generate $n \sim \text{Poisson}(\Lambda(t_0))$.
- (2) Independently generate n random variates t'_1, t'_2, \dots, t'_n from the cdf $F(t) = \Lambda(t)/\Lambda(t_0)$.
- (3) Order t'_1, t'_2, \dots, t'_n to obtain $t_1 = t'_{(1)}, t_2 = t'_{(2)}, \dots, t_n = t'_{(n)}$.
- (4) Deliver t_1, t_2, \dots, t_n .

2.2.1 Mean function $\Lambda(t)$

```

# Mean function Lambda for JPLP
L_plp = function(t, beta = 1.5, theta = 4) return((t/theta)^beta)
Lambda = function(t,
                  tau = 12,
                  kappa = 0.8,

```

```

        t_trip = c(3.5, 6.2, 9),
        beta = 1.5,
        theta = 4)
{
  t_trip1 = c(0, t_trip)
  n_trip = length(t_trip1)
  comp = L_plp(t_trip, beta, theta)
  kappa_vec0 = rep(kappa, n_trip - 1)^(0:(n_trip - 2))
  kappa_vec1 = rep(kappa, n_trip - 1)^(1:(n_trip - 1))
  cum_comp0 = comp*kappa_vec0
  cum_comp1 = comp*kappa_vec1
  index_trip = max(cumsum(t > t_trip1)) - 1

  if(index_trip == 0){
    return((t/theta)^beta)
  }else{
    return(sum(cum_comp0[1:index_trip]) - sum(cum_comp1[1:index_trip]) +
           kappa^index_trip*(t/theta)^beta)
  }
}

```

```
# test Lambda
```

```
kappa = 0.8
```

```
t_trip = c(3.5, 6.2, 9)
```

```
beta = 1.5
```

```
theta = 4
```

```
Lambda(3.1)
```

```
## [1] 0.6822642
```

```
L_plp(3.1)
```

```
## [1] 0.6822642
```

```
Lambda(4.1)
```

```
## [1] 0.9938842
```

```
kappa^0*L_plp(t_trip[1]) +
  kappa^1*L_plp(4.1) - kappa^1*L_plp(t_trip[1])
```

```
## [1] 0.9938842
```

```
Lambda(8.9)
```

```
## [1] 2.596555
```

```
kappa^0*L_plp(t_trip[1]) +
  kappa^1*L_plp(t_trip[2]) - kappa^1*L_plp(t_trip[1]) +
  kappa^2*L_plp(8.9) - kappa^2*L_plp(t_trip[2])
```

```
## [1] 2.596555
```

```
Lambda(12)
```

```
## [1] 3.564885
```

```
kappa^0*L_plp(t_trip[1]) +  
  kappa^1*L_plp(t_trip[2]) - kappa^1*L_plp(t_trip[1]) +  
  kappa^2*L_plp(t_trip[3]) - kappa^2*L_plp(t_trip[2]) +  
  kappa^3*L_plp(12) - kappa^3*L_plp(t_trip[3])
```

```
## [1] 3.564885
```

```
pacman::p_load(ggplot2, dplyr, tidyr)
```

```
beta = 1.5
```

```
theta = 4
```

```
x = seq(0.01, 12, 0.01)
```

```
y0 = (x/theta)^beta
```

```
y1 = rep(NA_real_, length(x))
```

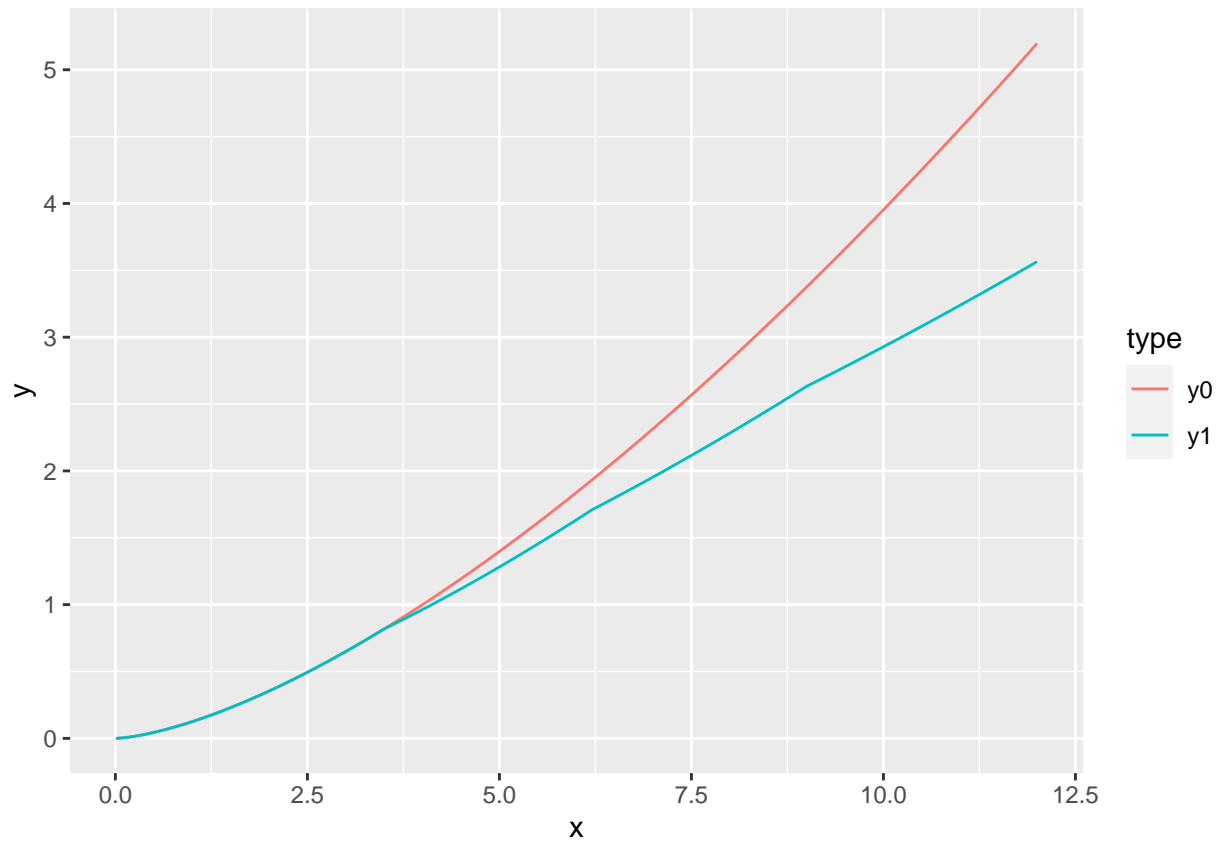
```
for (i in 1:length(x)) {  
  y1[i] = Lambda(t = x[i])  
}
```

```
data.frame(x, y0, y1) %>%
```

```
  tidyr::pivot_longer(cols = c('y0', 'y1'), names_to = "type", values_to = 'y') %>%
```

```
  ggplot(aes(x = x, y = y, group = type, color = type)) +
```

```
  geom_line()
```



2.2.2 Simulation JPLP events

```
sim_jplp = function(tau = 12,  
                    kappa = 0.8,  
                    t_trip = c(3.5, 6.2, 9),  
                    beta = 1.5,  
                    theta = 4)  
{  
  Lambda_t0 = Lambda(tau)  
}
```