Jump-point PLP (JPLP) simulation

Miao Cai miao.cai@slu.edu

2020-05-14

Contents

1 Power law process (PLP) **1**
1.1 PLP intensity function . . . 1
1.2 PLP simulation . . . 1

2 Jump Power Law Process (JPLP) **3**
2.1 JPLP intensity function . . . 3
2.2 JPLP mean function . . . 3
2.3 Simulation JPLP events in one shift . . . 6
2.4 Simulation JPLP events in multiple shifts . . . 12

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)
```

```r
    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)))
}
```

```r
# 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:

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

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 mean function

> Let $T_1, T_2, \ldots$ 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 $T_1, T_2, \ldots, T_n$ 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 $t'_1, t'_2, \ldots, t'_n$ from the cdf $F(t) = \Lambda(t)/\Lambda(t_0)$.
(3) Order $t'_1, t'_2, \ldots, t'_n$ to obtain $t_1 = t'_{(1)}, t_2 = t'_{(2)}, \ldots, t_n = t'_{(n)}$.
(4) Deliver $t_1, t_2, \ldots, t_n$.

2.2.1 Mean function $\Lambda(t)$

```
# Mean function Lambda for JPLP
Lambda_PLP = function(t, beta = 1.5, theta = 4) return((t/theta)^beta)
Lambda_JPLP = function(t,
```

```r
                  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 = Lambda_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)
  }
}
```

2.2.2 Test the mean function $\Lambda(t)$

```r
# test Lambda_JPLP
kappa = 0.8
t_trip = c(3.5, 6.2, 9)
beta = 1.5
theta = 4

Lambda_JPLP(3.1)
```

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

```r
Lambda_PLP(3.1)
```

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

```r
Lambda_JPLP(4.1)
```

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

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

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

```r
Lambda_JPLP(8.9)
```

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

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

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

```r
Lambda_JPLP(12)
```

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

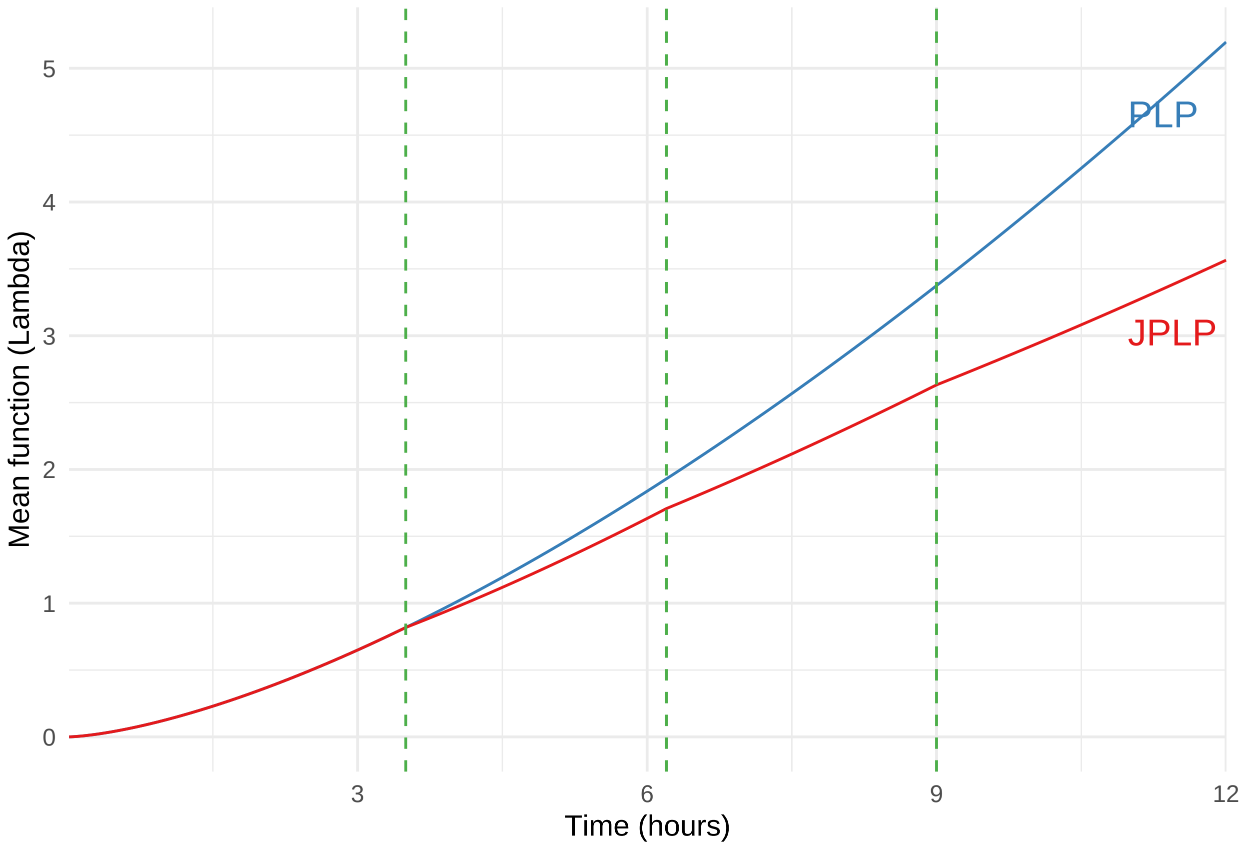
```r
kappa^0*Lambda_PLP(t_trip[1]) +
  kappa^1*Lambda_PLP(t_trip[2]) - kappa^1*Lambda_PLP(t_trip[1]) +
  kappa^2*Lambda_PLP(t_trip[3]) - kappa^2*Lambda_PLP(t_trip[2]) +
  kappa^3*Lambda_PLP(12) - kappa^3*Lambda_PLP(t_trip[3])
```

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

2.2.3 Plot the mean function of PLP and JPLP

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

t_trip = c(3.5, 6.2, 9)
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_JPLP(t = x[i], kappa = 0.8)
}

data.frame(x, y0, y1) %>%
  tidyr::pivot_longer(cols = c('y0', 'y1'),
                      names_to = "Type",
                      values_to = 'y') %>%
  mutate(Type = case_when(Type == 'y0' ~ 'PLP',
                          Type == 'y1' ~ 'JPLP')) %>%
  mutate(Type = factor(Type, levels = c('PLP', 'JPLP'))) %>%
  ggplot(aes(x = x, y = y, group = Type, color = Type)) +
  geom_line() +
  geom_dl(aes(label = Type),
          method = list(dl.trans(x = x - 1.3, y = y - 0.95),
                        "last.points", cex = 1.2)) +
  scale_color_manual(values = c(PLP = "#377eb8", JPLP = "#e41a1c")) +
  geom_vline(xintercept = t_trip, linetype = "dashed", color = '#4daf4a') +
  guides(color = FALSE) +
  labs(x = 'Time (hours)',
```

```
       y = 'Mean function (Lambda)') +
  theme_minimal() +
  scale_x_continuous(expand = c(0, 0), breaks = seq(0, 12, 3))
```



2.3 Simulation JPLP events in one shift

2.3.1 Simulate JPLP events in one shift

```
# Define a inverse function for mean function Lambda
inverse = function (f, lower = 0.0001, upper = 1000) {
   function (y) uniroot((function (x) f(x) - y), lower = lower, upper = upper)[1]
}

# Test the inverse function
Lambda1 = function(t, tau = 12, kappa = 0.8, t_trip = c(3.5, 6.2, 9),
                beta = 1.5, theta = 4){
  return(Lambda_JPLP(t, tau = tau, kappa = kappa, t_trip = t_trip,
                beta = beta, theta = theta))
}
inv_Lambda = inverse(Lambda1, 0.0001, 100)
inv_Lambda(3.564885)$root
```

```
## [1] 12
```

```r
# simulating JPLP - time truncated case
sim_jplp = function(tau0 = 12,
                    kappa0 = 0.8,
                    t_trip0 = c(3.5, 6.2, 9),
                    beta0 = 1.5,
                    theta0 = 0.5)
{ s = 0; t = 0
  Lambda1 = function(t, tau1 = tau0, kappa1 = kappa0, t_trip1 = t_trip0,
                     beta1 = beta0, theta1 = theta0){
    return(Lambda_JPLP(t, tau = tau1, kappa = kappa1, t_trip = t_trip1,
                  beta = beta1, theta = theta1))
  }
  inv_Lambda = inverse(Lambda1, 0.0001, 100)

  while (max(t) <= tau0) {
    u <- runif(1)
    s <- s - log(u)
    t_new <- inv_Lambda(s)$root
    t <- c(t, t_new)
  }
  t = t[c(-1, -length(t))]

  return(t)
}

sim_jplp(theta0 = 1)
```

```
##  [1]  0.3434214  0.6982757  1.9668339  2.1537226  3.2637677  3.7043029
##  [7]  5.1077061  5.6114654  5.7380023  6.0577339  6.5334161  6.8835452
## [13]  6.9050260  7.3024808  7.5522308  8.9954932  9.0553777  9.1984943
## [19] 10.6006232 10.6090574 11.1637054 11.5107823 11.7518029 11.9019663
```

```r
sim_jplp(theta0 = 2)
```

```
##  [1]  0.03732516  2.20522105  4.03099811  4.24079406  5.93202127
##  [6]  7.99753254  8.76639825  9.58785287 11.11548674 11.96509858
```

2.3.2 Plot JPLP events in one shift

```r
set.seed(123)

tauX = 12
kappaX = 0.8
t_tripX = c(3.5, 6.2, 9)
betaX = 1.5
thetaX = 2

t_events = sim_jplp(tau0 = tauX,
```

```r
                   kappa0 = kappaX,
                   t_trip0 = t_tripX,
                   beta0 = betaX,
                   theta0 = thetaX)

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_JPLP(t = x[i], kappa = 0.8)
}

dLambda = data.frame(x, y0, y1) %>%
  tidyr::pivot_longer(cols = c('y0', 'y1'),
                      names_to = "Type",
                      values_to = 'y') %>%
  mutate(Type = case_when(Type == 'y0' ~ 'PLP',
                          Type == 'y1' ~ 'JPLP')) %>%
  mutate(Type = factor(Type, levels = c('PLP', 'JPLP')))

d_event = data.frame(t_events = t_events, y = 0)
d_shift = data.frame(start_x = 0,
                     end_x = tauX,
                     start_y = 0,
                     end_y = 0)
```

```r
ggplot() +
  geom_line(data = dLambda, aes(x = x, y = y, group = Type, color = Type)) +
  geom_dl(data = dLambda, aes(x = x, y = y, color = Type, label = Type),
          method = list(dl.trans(x = x - 1.3, y = y - 0.9),
                        "last.points", cex = 1.2)) +
  scale_color_manual(values = c(PLP = "#377eb8", JPLP = "#e41a1c")) +
  guides(color = FALSE) +
  geom_vline(xintercept = t_trip, linetype = "dashed", color = '#4daf4a') +
  labs(x = 'Time (hours)',
       y = 'Mean function (Lambda)') +
  theme_minimal() +
  geom_point(data = d_event, aes(x = t_events, y = y),
             alpha = 0.8, shape = 4, color = 'red', size = 2) +
  geom_segment(data = d_shift,
               aes(x = start_x, xend = end_x,
                   y = start_y, yend = end_y),
               lineend = 'butt',
               arrow = arrow(length = unit(0.2, "cm"))) +
  scale_x_continuous(expand = c(0, 0), breaks = seq(0, 12, 3))
```

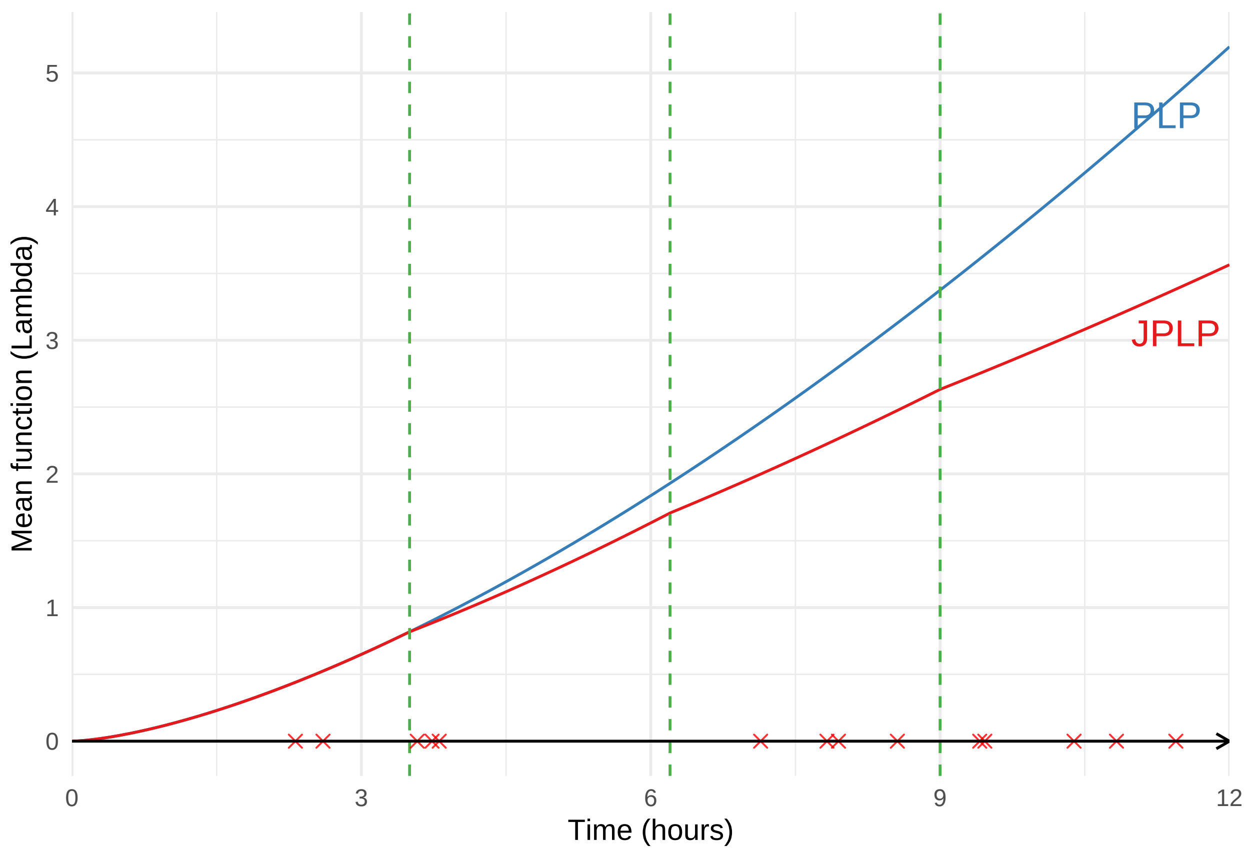


Figure 1: Event times (red cross) and mean function Lambda (curves)

```r
set.seed(666)

tauX = 12
kappaX = 0.8
t_tripX = c(3.5, 6.2, 9)
betaX = 1.5
thetaX = 1

t_events = sim_jplp(tau0 = tauX,
                    kappa0 = kappaX,
                    t_trip0 = t_tripX,
                    beta0 = betaX,
                    theta0 = thetaX)

lambda_plp = function(t, beta, theta) return(beta*theta^(-beta)*t^(beta - 1))
lambda_jplp = function(t, beta, theta, kappa = 0.8, t_trip){
  index_trip = max(cumsum(t > t_trip))
  return((kappa^index_trip)*beta*theta^(-beta)*t^(beta - 1))
}


x = seq(0.01, 12, 0.01)
y_plp = lambda_plp(x, beta = betaX, theta = thetaX)
y_jplp = rep(NA_real_, length(lambda_plp))

for (i in 1:length(x)) {
  y_jplp[i] = lambda_jplp(x[i],
                          beta = betaX, theta = thetaX,
                          kappa = 0.8, t_trip = t_tripX)
}


dlambda = tibble::tibble(x, y_plp, y_jplp) %>%
  tidyr::pivot_longer(cols = c('y_plp', 'y_jplp'),
                      names_to = "Type",
                      values_to = 'y') %>%
  mutate(Type = case_when(Type == 'y_plp' ~ 'PLP',
                          Type == 'y_jplp' ~ 'JPLP')) %>%
  mutate(Type = factor(Type, levels = c('PLP', 'JPLP')))

d_event = data.frame(t_events = t_events, y = 0)
d_shift = data.frame(start_x = 0,
                     end_x = tauX,
                     start_y = 0,
                     end_y = 0)
```

```r
ggplot() +
  geom_line(data = dlambda, aes(x = x, y = y, group = Type, color = Type)) +
  scale_color_manual(values = c(PLP = "#0B775E", JPLP = "#F2300F")) +
  geom_dl(data = dlambda, aes(x = x, y = y, label = Type, color = Type),
          method = list(dl.trans(x = x - 1.3, y = y - 0.5),
                        "last.points", cex = 1.2)) +
  geom_vline(xintercept = t_trip, linetype = "dashed") +
  labs(x = 'Time (hours)',
       y = 'Mean function (Lambda)') +
  theme_minimal() +
  guides(color = FALSE) +
  geom_point(data = d_event, aes(x = t_events, y = y),
             alpha = 0.8, shape = 4, color = 'red', size = 2) +
  geom_segment(data = d_shift,
               aes(x = start_x, xend = end_x,
                   y = start_y, yend = end_y),
               lineend = 'butt',
               arrow = arrow(length = unit(0.2, "cm"))) +
  scale_x_continuous(expand = c(0, 0), breaks = seq(0, 12, 3))
```

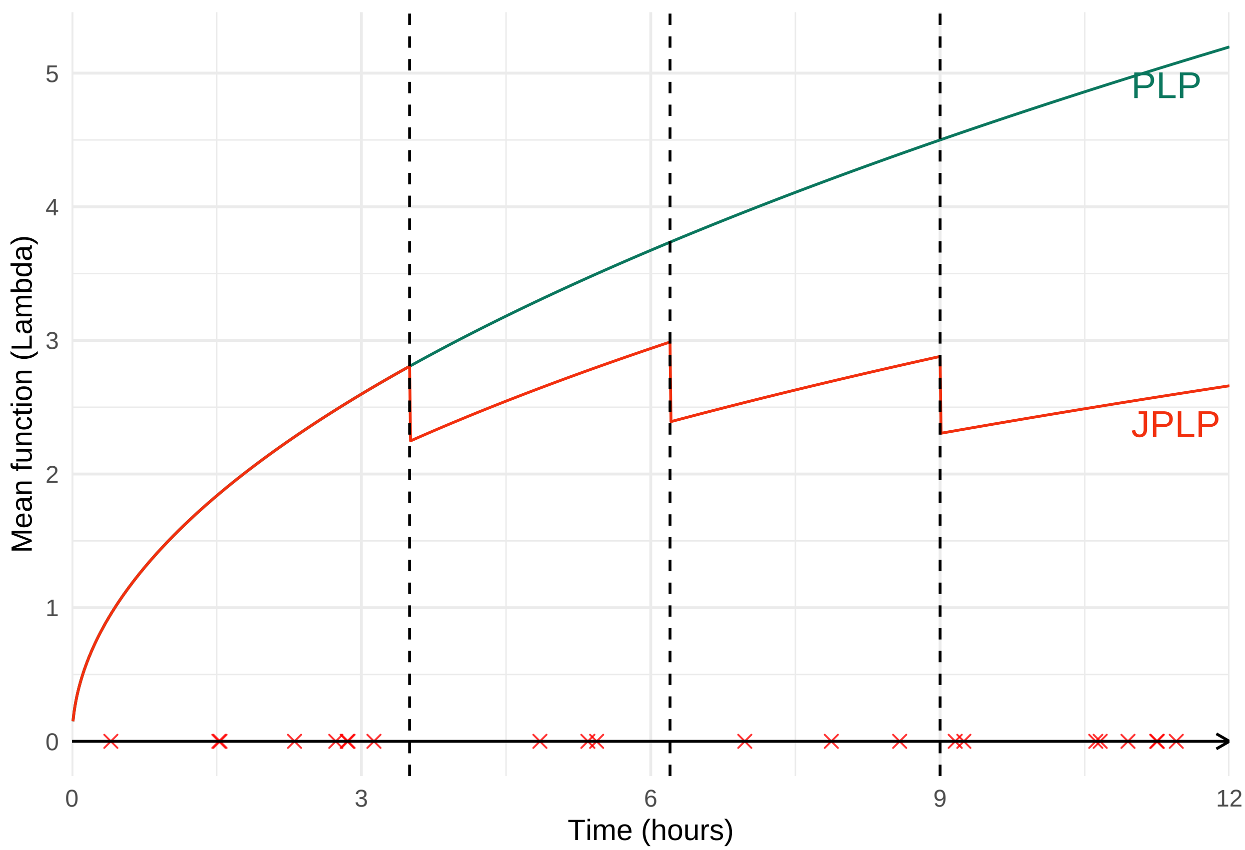


Figure 2: Event times (red cross) and intensity function lambda (curves)

2.4 Simulation JPLP events in multiple shifts

```r
# Function to get the number of stops
get_n_stop = function() sample(1:4, 1, TRUE)

# Function to simulate event time for multiple shifts
sim_mul_jplp = function(kappa = 0.8, beta = 1.5, theta = 2, n_shift = 10)
{
  t_shift_vec = list()
  n_stop_vec = list()
  t_stop_vec = list()
  n_event_vec = list()
  t_event_vec = list()

  for (i in 1:n_shift) {
    sim_tau = rnorm(1, 10, 1.3)
    n_stop = get_n_stop()
    sim_t_trip = round((1:n_stop)*sim_tau/(n_stop + 1) +
                        rnorm(n_stop, 0, sim_tau*0.15/n_stop), 2)
    t_events = sim_jplp(tau0 = sim_tau,
                       kappa0 = kappa,
                       t_trip0 = sim_t_trip,
                       beta0 = beta,
                       theta0 = theta)
    t_shift_vec[[i]] = sim_tau
    n_stop_vec[[i]] = n_stop
    t_stop_vec[[i]] = sim_t_trip
    n_event_vec[[i]] = length(t_events)
    t_event_vec[[i]] = t_events
  }

  event_dt = data.frame(
    shift_id = rep(1:n_shift, unlist(n_event_vec)),
    event_time = Reduce(c, t_event_vec)
  )

  trip_dt = data.frame(
    shift_id = rep(1:n_shift, unlist(n_stop_vec)),
    trip_time = Reduce(c, t_stop_vec)
  )

  shift_dt = data.frame(
    shift_id = 1:n_shift,
    start_time = rep(0, n_shift),
    end_time = Reduce(c, t_shift_vec)
  )
```

```r
  return(list(event_time = event_dt,
              trip_time = trip_dt,
              shift_time = shift_dt))

}

set.seed(666)
z = sim_mul_jplp(theta = 3)
str(z)
```

```
## List of 3
##  $ event_time:'data.frame':  49 obs. of  2 variables:
##   ..$ shift_id  : int [1:49] 1 2 3 3 3 3 3 3 3 4 ...
##   ..$ event_time: num [1:49] 10.8 8.48 2.31 3.33 4.84 ...
##  $ trip_time :'data.frame':  30 obs. of  2 variables:
##   ..$ shift_id : int [1:30] 1 1 1 1 2 2 2 3 3 3 ...
##   ..$ trip_time: num [1:30] 1.85 4.66 6.59 8.52 1.38 4.29 7.38 2.94 6.15 8.2 ...
##  $ shift_time:'data.frame':  10 obs. of  3 variables:
##   ..$ shift_id  : int [1:10] 1 2 3 4 5 6 7 8 9 10
##   ..$ start_time: num [1:10] 0 0 0 0 0 0 0 0 0 0
##   ..$ end_time  : num [1:10] 10.98 8.61 11.12 10.1 10.63 ...
```

```r
# Plot events in multiple shifts
plot_jplp = function(dt){
  p = ggplot() +
    geom_point(data = dt$event_time, aes(x = event_time, y = shift_id),
               alpha = 0.8, shape = 4, color = '#F2300F', size = 3, stroke = 1) +
    geom_point(data = dt$trip_time, aes(x = trip_time, y = shift_id),
               shape = 3, color = '#0B775E', fill = '#0B775E', size = 3.5, stroke = 1.05) +
    geom_segment(data = dt$shift_time,
                 aes(x = start_time, xend = end_time,
                     y = shift_id, yend = shift_id),
                 lineend = 'butt',
                 arrow = arrow(length = unit(0.2, "cm"))) +
    scale_y_continuous("shift ID",
                       labels = as.character(dt$shift_time$shift_id),
                       breaks = dt$shift_time$shift_id) +
    labs(x = 'Time (hours)') +
    theme_classic()
    return(p)
}

plot_jplp(z)
```

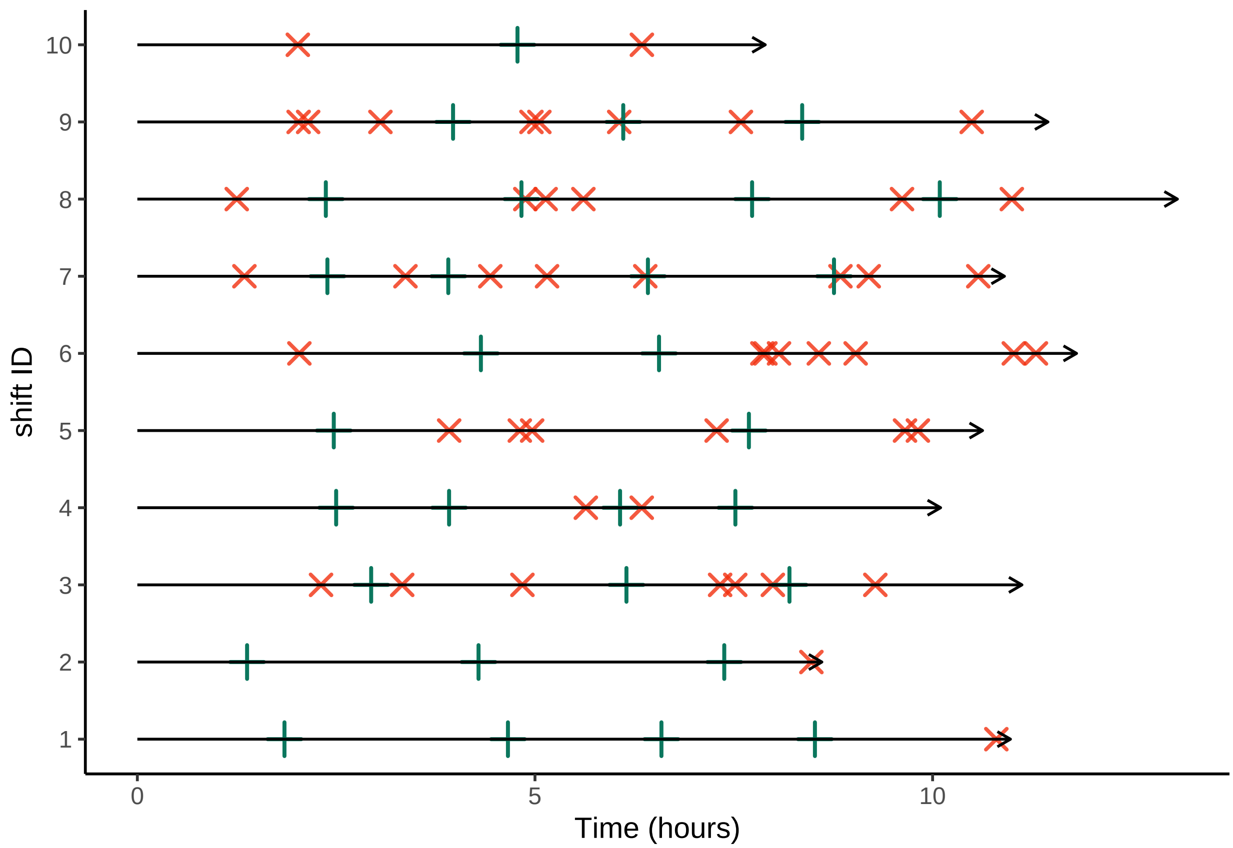


Figure 3: Simulated events (red cross) in different trips (green cross) and shifts