Bayesian hierarchical models for NHPP using `rstan`

Miao Cai miao.cai@slu.edu

2019-08-08

Contents

1 Model setting

Let $T_{d,s,i}$ denote the time to the d-th driver's s-th shift's i-th critical event. The total number critical events of d-th driver's s-th shift is $n_{d,s}$. The ranges of these notations are:

- $i = 1, 2, \cdots, n_{d,S_d}$,
- $s = 1, 2, \cdots, S_d$,
- $d = 1, 2, \cdots, D$.

We assume the times of critical events within the d-th driver's s-th shift were generated from a non-homogeneous Poisson process (NHPP) with a power law process (PLP), with a fix shape parameter β and varying scale parameters $\theta_{d,s}$ across drivers. The data generating process is then:

$$
\begin{aligned}
T_{d,s,1}, T_{d,s,2}, \cdots, T_{d,s,n_{d,s}} &\sim \text{PLP}(\beta, \theta_{d,s}) \\
\beta &\sim \text{Gamma}(1, 1) \\
\log \theta_{d,s} &= \gamma_{0d} + \gamma_1 x_{d,s,1} + \gamma_2 x_{d,s,2} + \cdots + \gamma_k x_{d,s,k} \\
\gamma_{01}, \gamma_{02}, \cdots, \gamma_{0D} &\sim \text{i.i.d. } N(\mu_0, \sigma_0^2) \\
\gamma_1, \gamma_2, \cdots, \gamma_k &\sim \text{i.i.d. } N(0, 10^2) \\
\mu_0 &\sim N(0, 10^2) \\
\sigma_0 &\sim \text{Gamma}(1, 1)
\end{aligned}
$$

2 Simulating data

2.1 Theoretical data generating process (DGP)

1. Random intercepts $\gamma_{01}, \gamma_{02}, \cdots, \gamma_{0D}$. The standard deviation of μ_0 was intentionally set to small number 2 to make $\theta_{d,s}$ fall into a reasonably small range. If I otherwise set it as 10, $\theta_{d,s}$ may be more than 10^5 due to the exponentiation, which may not be realistic in real-life data.

$$\begin{aligned} \mu_0 = 0, \quad \sigma_0 &= 0.5 \\ \sigma_0 &\sim \text{Gamma}(1,1) \\ \gamma_{01}, \gamma_{02}, \cdots, \gamma_{0D} &\sim \text{i.i.d. } N(\mu_0, \sigma_0^2) \end{aligned}$$

2. Fixed parameters: 3 fixed parameters $\gamma_1, \gamma_2, \gamma_3$.

$$\gamma_1, \gamma_2, \gamma_3 \sim \text{i.i.d. } N(0, 0.5^2)$$

3. The number of observations in the d-th driver: N_d.

$$N_d \sim \text{Poisson}(10)$$

4. Data: 3 predictor variables $x_{d,s,1}, x_{d,s,2}, x_{d,s,3}$.

$$\begin{aligned} x_{d,s,1} &\sim \text{N}(0,1) \\ x_{d,s,2} &\sim \text{Gamma}(1,1) \\ x_{d,s,3} &\sim \text{Poisson}(0.2) \end{aligned}$$

5. Scale parameters of a NHPP (random effects): $\theta_{d,s}$.

$$\theta_{d,s} = \text{EXP}(\gamma_{0d} + \gamma_1 x_{d,s,1} + \gamma_2 x_{d,s,2} + \gamma_k x_{d,s,3})$$

6. Shape parameter of a NHPP (fixed effect): $\beta \sim \text{Gamma}(1,1)$. Set

$$\beta = 1.5$$

7. Simulate truncate time τ_s for each shift.

$$\tau_s \sim N(10, 1.3)$$

8. Simulate a NHPP based on β and $\theta_{d,s}$.

$$T_{d,s,1}, T_{d,s,2}, \cdots, T_{d,s,n_{d,s}} \sim \text{PLP}(\beta, \theta_{d,s})$$

2.2 R code to simulate data and parameters according to the DGP

```
pacman::p_load(rstan, tidyverse, data.table)
source("functions/NHPP_functions.R")

set.seed(123)
D = 10 # the number of drivers
K = 3 # the number of predictor variables

# 1. Random-effect intercepts
# hyperparameters
mu0 = 0
sigma0 = 0.5
r_0D = rnorm(D, mean = mu0, sd = sigma0)

# 2. Fixed-effects parameters
R_K = rnorm(K, mean = 0, sd = 0.5)

# 3. The number of observations (shifts) in the $d$-th driver: $N_{d}$
N_K = rpois(D, 10)
N = sum(N_K) # the total number of obs
id = rep(1:D, N_K)

# 4. Generate data: x_1, x_2, .. x_K
sim1 = function(group_sizes = N_K){
  ntot = sum(group_sizes)

  int1 = rep(1, ntot)
  x1 =  rnorm(ntot, 0, 1)
  x2 = rgamma(ntot, 1, 1)
  x3 =  rpois(ntot, 0.2)

  return(data.frame(int1, x1, x2, x3))
}
X = sim1(N_K)

# 5. Scale parameters of a NHPP
# 5a. parameter matrix: P
P = cbind(r0 = rep(r_0D, N_K), t(replicate(N, R_K)))
M_logtheta = P*X

# returned parameter for each observed shift
beta = 1.5
theta = exp(rowSums(M_logtheta))
round(theta, 3)
```

```
##  [1] 0.284 1.721 1.440 0.403 1.672 1.265 0.795 2.269 1.485 1.498 2.208
## [12] 1.502 0.875 0.817 0.713 0.777 1.063 0.626 9.235 4.786 1.793 3.045
## [23] 2.627 3.822 2.729 2.755 2.249 2.171 6.624 2.214 6.020 0.975 3.326
## [34] 1.140 2.079 1.334 1.228 0.852 0.792 0.560 1.887 1.434 1.364 2.426
## [45] 4.139 0.976 0.270 5.343 1.880 2.676 4.558 3.342 1.340 3.739 1.500
## [56] 1.794 1.804 1.680 1.937 1.783 0.728 2.232 0.790 0.474 1.108 1.203
## [67] 0.910 0.646 0.507 1.667 0.597 2.714 1.961 0.772 0.441 0.662 1.144
## [78] 0.740 0.659 0.568 0.916 0.606
```

```r
round(r_OD, 3)
```

```
##  [1] -0.280 -0.115  0.779  0.035  0.065  0.858  0.230 -0.633 -0.343 -0.223
```

2.3 Generate NHPP data to pass to `rstan`

```r
sim_hier_plp_tau = function(){
  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, 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)))
}

df = sim_hier_plp_tau()

hier_dat = list(
    N = nrow(df$event_dat),
    K = nrow(df$start_end_dat),
    D = id, #driver index
    tau = df$start_end_dat$end_time,
    event_time = df$event_dat$event_time,
    s = df$shift_length, #the number of events in each shift
    x1 = X[,2], x2 = X[,3], x3 = X[,4]
)
```

3 Stan code

```
functions{
  real nhpp_log(vector t, real beta, real theta, real tau){
    vector[num_elements(t)] loglik_part;
    real loglikelihood;
    for (i in 1:num_elements(t)){
      loglik_part[i] = log(beta) - beta*log(theta) + (beta - 1)*log(t[i]);
    }
    loglikelihood = sum(loglik_part) - (tau/theta)^beta;
    return loglikelihood;
  }
}
data {
  int<lower=0> N; //total # of obs
  int<lower=0> K; //total # of shifts
  int<lower=0> D[K];//driver index, this must be an array
  vector<lower=0>[K] tau;//truncated time
  vector<lower=0>[N] event_time; //failure time
  int s[K]; //group sizes
  vector[K] x1;
  vector[K] x2;
  vector[K] x3;
}
parameters{
  real<lower=0> beta;
  vector[K] r0; // random intercept
  vector[3] r; // fixed parameters
  real mu0; // hyperparameter
  real<lower=0> sigma0;// hyperparameter
}
transformed parameters{
  vector<lower=0>[K] theta;
  for (k0 in 1:K){
    theta[k0] = exp(r0[ D[k0] ] + x1[k0]*r[1] + x2[k0]*r[2] + x3[k0]*r[3]);
  }
}
model{
  int position;
  position = 1;
  for (k in 1:K){
    if(s[k] == 0) continue;
    segment(event_time, position, s[k]) ~ nhpp(beta, theta[k], tau[k]);
    position = position + s[k];
  }
  beta ~ gamma(1, 1);
  r0 ~ normal(mu0, sigma0);
  r  ~ normal(0, 10);
  mu0 ~ normal(0, 10);
  sigma0 ~ gamma(1, 1);
  theta ~ gamma(1, 0.01);
}
```

4 Estimated results

4.1 A single simulation to demonstrate

```
f = stan("stan/nhpp_plp_tau_ML.stan",
         chains = 1, iter = 1000, data = hier_dat, refresh = 0)
```

```
## DIAGNOSTIC(S) FROM PARSER:
## Info:
## Left-hand side of sampling statement (~) may contain a non-linear transform of a parameter or local v
## If it does, you need to include a target += statement with the log absolute determinant of the Jacobi
## Left-hand-side of sampling statement:
##     theta ~ gamma(...)

## Warning: There were 1 chains where the estimated Bayesian Fraction of Missing Information was low. Se
## http://mc-stan.org/misc/warnings.html#bfmi-low

## Warning: Examine the pairs() plot to diagnose sampling problems

## Warning: Bulk Effective Samples Size (ESS) is too low, indicating posterior means and medians may be
## Running the chains for more iterations may help. See
## http://mc-stan.org/misc/warnings.html#bulk-ess

## Warning: Tail Effective Samples Size (ESS) is too low, indicating posterior variances and tail quanti
## Running the chains for more iterations may help. See
## http://mc-stan.org/misc/warnings.html#tail-ess
```

```
pacman::p_load(magrittr)
est = broom::tidy(f)

pull_est = function(var = "theta", est_obj = f){
  z = est_obj %>%
    broom::tidy() %>%
    filter(grepl(var, term)) %>%
    pull(estimate) %>%
    round(3)
  return(z)
}
```

Estimated values:

- Hyperparameters: $\hat{\mu}_0$: 0.043, $\hat{\sigma}_0$: 0.572
- Individual level parameters: $\gamma_1, \gamma_2, \gamma_3$: 0.626, 0.16, 0.164
- Rate parameter β: 1.499
- θ: 0.292, 1.756, 1.416, 0.417, 1.796, 1.308, 0.816, 2.228, 1.567, 1.572, 2.176, 1.529, 0.899, 0.829, 0.73, 0.768, 1.056, 0.605, 9.656, 4.971, 1.69, 2.911, 2.539, 3.928, 2.675, 2.812, 2.295, 2.221, 6.722, 2.225, 6.253, 0.963, 3.417, 1.15, 1.977, 1.351, 1.167, 0.856, 0.747, 0.555, 1.78, 1.449, 1.323, 2.403, 4.42, 0.991, 0.272, 5.425, 1.862, 2.518, 4.72, 3.214, 1.322, 3.689, 1.433, 1.724, 1.788, 1.555, 1.947, 1.66, 0.734, 2.118, 0.796, 0.475, 1.14, 1.195, 0.911, 0.655, 0.481, 1.632, 0.557, 2.695, 1.911, 0.712, 0.446, 0.657, 1.155, 0.753, 0.674, 0.562, 0.926, 0.596

4.2 Scale up simulation

To be added.

5 Further improvement

In Stan code:

- Need a data matrix X,
- Need matrix multiplication,

In data:

- Need a driver index $d = 1, 2, \cdots, K$ for each shift k
- Need a data matrix X

```
pacman::p_load(rstan, tidyverse, data.table)
source("functions/NHPP_functions.R")


sim_hier_nhpp = function(group_size_lambda = 10, D = 10, K = 3, beta = 1.5)
{
  # 1. Random-effect intercepts
  # hyperparameters
  mu0 = 0.2
  sigma0 = 0.5
  r_0D = rnorm(D, mean = mu0, sd = sigma0)

  # 2. Fixed-effects parameters
  R_K = c(1, 0.3, 0.2)

  # 3. The number of shifts in the $d$-th driver: $N_{d}$
  N_K = rpois(D, group_size_lambda)
  N = sum(N_K) # the total number of obs
  id = rep(1:D, N_K)

  # 4. Generate data: x_1, x_2, .. x_K
  sim1 = function(group_sizes = N_K)
  {
    ntot = sum(group_sizes)
```

```r
  int1 = rep(1, ntot)
  x1 = rnorm(ntot, 1, 1)
  x2 = rgamma(ntot, 1, 1)
  x3 = rpois(ntot, 2)

  return(data.frame(int1, x1, x2, x3))
}
X = sim1(N_K)

# 5. Scale parameters of a NHPP
# 5a. parameter matrix: P
P = cbind(r0 = rep(r_0D, N_K),
          t(replicate(N, R_K)))
M_logtheta = P*X

# returned parameter for each observed shift

theta_vec = exp(rowSums(M_logtheta))

df = sim_hier_plp_tau(N = N, beta = beta, theta = theta_vec)

hier_dat = list(
  N = nrow(df$event_dat),
  K = K,
  S = nrow(df$start_end_dat),
  D = max(id),
  id = id, #driver index
  tau = df$start_end_dat$end_time,
  event_time = df$event_dat$event_time,
  group_size = df$shift_length, #the number of events in each shift
  X_predictors = X[,2:4]
)
```

```r
  true_params = list(
    mu0 = mu0, sigma0 = sigma0,
    r0 = r_0D, r1_rk = R_K,
    beta = beta,
    theta = theta_vec
  )

  return(list(hier_dat = hier_dat, true_params = true_params))
}


# sampling from Stan
df = sim_hier_nhpp(D = 10, beta = 1.5)
f = stan("stan/nhpp_plp_tau_ML1.stan",
         chains = 1, iter = 1000, data = df$hier_dat)
```

```
##
## SAMPLING FOR MODEL 'nhpp_plp_tau_ML1' NOW (CHAIN 1).
## Chain 1:
## Chain 1: Gradient evaluation took 0.000307 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 3.07 seconds.
## Chain 1: Adjust your expectations accordingly!
## Chain 1:
## Chain 1:
## Chain 1: Iteration:   1 / 1000 [  0%]  (Warmup)
## Chain 1: Iteration: 100 / 1000 [ 10%]  (Warmup)
## Chain 1: Iteration: 200 / 1000 [ 20%]  (Warmup)
## Chain 1: Iteration: 300 / 1000 [ 30%]  (Warmup)
## Chain 1: Iteration: 400 / 1000 [ 40%]  (Warmup)
## Chain 1: Iteration: 500 / 1000 [ 50%]  (Warmup)
## Chain 1: Iteration: 501 / 1000 [ 50%]  (Sampling)
## Chain 1: Iteration: 600 / 1000 [ 60%]  (Sampling)
```

```
## Chain 1: Iteration: 700 / 1000 [ 70%]  (Sampling)
## Chain 1: Iteration: 800 / 1000 [ 80%]  (Sampling)
## Chain 1: Iteration: 900 / 1000 [ 90%]  (Sampling)
## Chain 1: Iteration: 1000 / 1000 [100%]  (Sampling)
## Chain 1:
## Chain 1:  Elapsed Time: 2.22766 seconds (Warm-up)
## Chain 1:                1.44747 seconds (Sampling)
## Chain 1:                3.67513 seconds (Total)
## Chain 1:
```

```
# check estimation results
f
```

```
## Inference for Stan model: nhpp_plp_tau_ML1.
## 1 chains, each with iter=1000; warmup=500; thin=1;
## post-warmup draws per chain=500, total post-warmup draws=500.
##
##                mean se_mean   sd  2.5%   25%   50%   75% 97.5% n_eff
## mu0            1.77    0.01 0.13  1.52  1.69  1.78  1.86  2.02   296
## sigma0         0.35    0.01 0.11  0.21  0.28  0.33  0.39  0.58   320
## beta           1.46    0.00 0.05  1.37  1.42  1.46  1.49  1.56   259
## R1_K[1]        0.92    0.00 0.05  0.82  0.88  0.92  0.96  1.03   195
## R1_K[2]        0.25    0.00 0.05  0.15  0.22  0.25  0.28  0.34   554
## R1_K[3]        0.22    0.00 0.04  0.15  0.19  0.22  0.24  0.30   367
## R0[1]          1.52    0.01 0.10  1.31  1.46  1.52  1.58  1.73   205
## R0[2]          1.74    0.01 0.11  1.52  1.66  1.74  1.81  1.95   182
## R0[3]          1.52    0.01 0.09  1.36  1.46  1.52  1.58  1.70   265
## R0[4]          1.81    0.01 0.11  1.59  1.73  1.81  1.88  2.04   419
## R0[5]          1.65    0.01 0.11  1.44  1.59  1.65  1.72  1.87   202
## R0[6]          1.25    0.01 0.11  1.02  1.17  1.25  1.33  1.47   126
## R0[7]          2.03    0.01 0.17  1.68  1.93  2.02  2.14  2.36   292
## R0[8]          1.92    0.01 0.12  1.70  1.85  1.93  1.99  2.17   460
## R0[9]          2.09    0.01 0.11  1.89  2.02  2.08  2.15  2.31   393
## R0[10]         2.20    0.01 0.13  1.94  2.11  2.19  2.28  2.44   229
```

```
## mu0_true       0.19    0.01 0.15  -0.09   0.09   0.19   0.30   0.46   573
## R0_true[1]    -0.06    0.01 0.11  -0.27  -0.14  -0.05   0.02   0.14   473
## R0_true[2]     0.16    0.01 0.12  -0.09   0.08   0.15   0.24   0.37   381
## R0_true[3]    -0.06    0.01 0.12  -0.29  -0.14  -0.06   0.03   0.15   444
## R0_true[4]     0.23    0.01 0.14  -0.04   0.13   0.23   0.32   0.48   433
## R0_true[5]     0.07    0.01 0.12  -0.16   0.00   0.07   0.15   0.31   514
## R0_true[6]    -0.33    0.01 0.12  -0.59  -0.41  -0.33  -0.25  -0.10   348
## R0_true[7]     0.45    0.01 0.20   0.05   0.33   0.45   0.59   0.83   342
## R0_true[8]     0.34    0.01 0.14   0.08   0.25   0.34   0.44   0.61   397
## R0_true[9]     0.51    0.01 0.14   0.21   0.42   0.50   0.60   0.78   598
## R0_true[10]    0.61    0.01 0.14   0.34   0.53   0.62   0.71   0.87   404
## lp__         298.05    0.22 3.04 291.48 296.26 298.25 300.24 303.28   193
##              Rhat
## mu0          1.00
## sigma0       1.00
## beta         1.00
## R1_K[1]      1.00
## R1_K[2]      1.00
## R1_K[3]      1.00
## R0[1]        1.00
## R0[2]        1.00
## R0[3]        1.00
## R0[4]        1.00
## R0[5]        1.00
## R0[6]        1.00
## R0[7]        1.00
## R0[8]        1.00
## R0[9]        1.00
## R0[10]       1.00
## mu0_true     1.00
## R0_true[1]   1.00
## R0_true[2]   1.01
## R0_true[3]   1.00
```

```
## R0_true[4]  1.00
## R0_true[5]  1.00
## R0_true[6]  1.00
## R0_true[7]  1.00
## R0_true[8]  1.00
## R0_true[9]  1.00
## R0_true[10] 1.00
## lp__        1.00
##
## Samples were drawn using NUTS(diag_e) at Thu Aug  8 00:06:58 2019.
## For each parameter, n_eff is a crude measure of effective sample size,
## and Rhat is the potential scale reduction factor on split chains (at
## convergence, Rhat=1).
```

```
df$true_params
```

```
## $mu0
## [1] 0.2
##
## $sigma0
## [1] 0.5
##
## $r0
##  [1]  0.007600664  0.365340280 -0.077310225  0.260786157  0.176201942
##  [6] -0.188125795  0.615720626  0.623153919  0.712069753  0.833998293
##
## $r1_rk
## [1] 1.0 0.3 0.2
##
## $beta
## [1] 1.5
##
## $theta
##   [1]   0.6034702   2.4859378   1.1791556  11.9240479   8.0888078
```

```
##   [6]   8.2844389   5.9921438   4.0092433   4.0793214   2.4560764
##  [11]   0.4900528   9.1472993   7.1127270 385.3064549  46.8419284
##  [16]   9.5229321   1.4952724   1.3077455   7.0098578   1.2306090
##  [21]  11.2896253   4.3779013   1.9697259   1.7257327   4.6076529
##  [26]   2.9979074   5.0998086   5.4272624  28.3335796   6.9349328
##  [31]   4.4659386   4.2114884   2.3906267  27.5547284   6.2070766
##  [36]   6.4071452  15.5636042   5.6499430   3.5256133  70.0814971
##  [41]   5.5929231   2.7913710  24.2494362   0.9141287   5.0234923
##  [46]  23.1163245   7.8887152   8.6089758   2.1110206  23.9357796
##  [51]  49.6033563   0.3241403  14.0838740   8.6657893   8.7143414
##  [56]   8.5086261   6.7382076  15.6917833  87.6953307  15.6815778
##  [61]   2.5723906  13.2830855   9.8674529 235.5315816   3.3269894
##  [66]   7.3713720  10.7808832  17.6446845   4.2852413  24.0938685
##  [71]  50.1672660   3.4503360  17.0192166  12.2051503  14.3430527
##  [76]  51.6852054   2.9420156  11.3943026   2.6243754   4.3414671
##  [81]  14.2609634   1.2562075  35.9411413   3.6466677  16.6637638
##  [86]   2.4246154  10.4860473  10.7440957   4.9308350  28.8213547
##  [91]  32.5376764  10.0808709   1.3616594  10.4626472   1.6026213
##  [96]  78.7852806  27.0651914  31.7142262  16.8507354  15.4378015
## [101]  11.4646960
```