Appendix

Frances Lin

June 2021

Simulation of a HPP

Recall that for a HPP, the interarrival times between events, W, are exponentially distributed. In this simulation of a HPP, after initializing the initial time, t, and the time vector, t_{vector}, we generate exponential random variables and use them to index the interarrival times between events.

Figure 2 as shown previously is a realization of a HPP with rates that are roughly constant at $\lambda = 10$.

Algorithm 1: Simulation of a HPP

Input λ, t_{max}

1. Initialize t, t_{vector}
2. **while** $(t \leq t_{max})$
3. Generate $u \sim U(0, 1)$
4. Set $t = t + w$ where $w = -log(u)/\lambda \sim exp(\lambda^* = \lambda)$
5. **if** $(t \leq t_{max})$
6. | Add $t_{vector} = c(t_{vector}, t)$
7. **else**
8. | **return** $\{t_k\}_{k=0,1,...}$

Simulation of a Hawkes Process

In this simulation of a Hawkes process, we use the thinning algorithm (or acceptance-rejection method) to simulate a temporal Hawkes process since it is one of the most popular choices for simulating both temporal and spatio-temporal NPP (Pasupathy, 2010). Broadly put, the thinning algorithm involves randomly deleting points from a point pattern. The process requires first simulating a HPP, creating a $\lambda(t)$ function and applying it to the HPP, and using $min(\lambda^*/\lambda, 1)$ as the accepting probability to randomly keep or 'thin' the points.

Figure 5 as shown previously is a realization of a Hawkes process with parameters such that $\mu = 0.5, \alpha = 0.5$, and $\beta = 0.7$. μ sets the background rate. α and β control the shape and decay rate of the the exponentially decaying triggering function function.

Algorithm 2: Simulation of a Hawkes Process via Thinning Algorithm

---

Imput μ, α, β, λ, t_{max}

1. Simulate a HPP using Algorithm 1
2. Create a $\lambda(t)$ function where the function $= \mu + \sum_{i:T_i<t} \alpha e^{-\beta x}$
3. Set $\lambda^* =$ apply the $\lambda(t)$ function to the HPP
4. Generate $u \sim U(0, 1)$
5. **if** $(u < min(\frac{\lambda^*}{\lambda}, 1))$ where the accepting probability $= min(\lambda^*/\lambda, 1)$
6. | Keep the points
7. **else**
8. | "Thin" or reject the points and **return** $\{t_k\}_{k=0,1,...}$

Simulations of a HPP, NPP, Cox and Matern Cluster Process in 2D

All of the corresponding plots in 2D are created using the **spatstat** package of `R` (Baddeley & Turner, 2005).

HPP

The `rpoispp` function can be used to generate a random point pattern as a realization of a HPP or NPP. `lambda` controls the rate (or intensity) of a HPP and `win` sets the window in which the simulated pattern is observed. In Figure 3, we set `lambda` $= 100$ and `win` $= square(1)$.

NPP

Similar to the above simulation, `lambda` can be set to control the intensity of a NPP. In the second figure of Figure 3, `lambda` $= 400 * x * y$.

Cox Process

Instead of setting `lambda` to be a deterministic function as in the NPP case, here we can set `lambda` to be a random function. In Figure 4, `lambda` is set to be $= rexp(n = 1, rate = 1/100)$.

Matern Cluster Process

Simulations of Matern cluster process are generated using the `rMatClust` function. Specifically, the process involves generating homogeneous Poisson parents and each parent gives rise to Poisson number of offspring uniformly distributed in a disc of radius r centered around the parent. `kappa` controls the intensity of the cluster centers and allows us to specify the number of clusters. `r` specifies how far away cluster is from one another in radius, and `mu` gives the mean number of points per cluster. In the second figure of Figure 4, $kappa = 20, r = 0.05$, and $mu = 5$.