Self-Exciting Point Processes with Isotropic Triggering

Self-Exciting Point Processes

The Hawkes process is a self-exciting point process whose conditional intensity λ^* increases in the aftermath of an event. The conditional intensity can be defined as

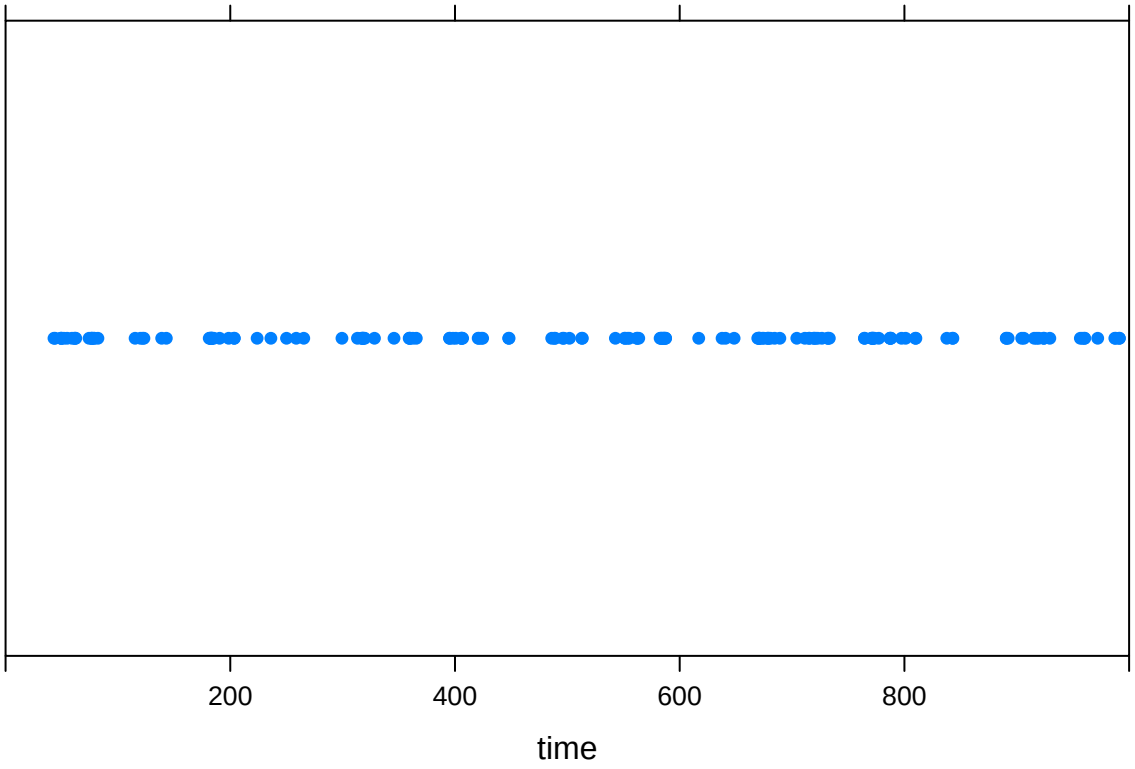
$$\lambda^*(t) = \mu(t) + \sum_{t_i<t} g(t - t_i).$$

A common form for the triggering function g is known as the ETAS model

$$g(t) = \alpha\omega e^{-\omega t}.$$

We can simulate such a process, with constant background rate μ, as follows.

```
library(lattice)
hawkes <- function(param, horizon) {
    ### param is vector, mu the first argument, alpha the second,
    ### omega the third
    t <- 0
    n <- 0
    tn <- NULL
    while (t < horizon) {
        l_t <- horizon - t
        m_t <- param[1] + param[2] * param[3] * (sum(exp(-param[3] *
            (t - tn))))
        s <- rexp(1, rate = m_t)
        U <- runif(1)
        lambda_ts <- param[1] + param[2] * param[3] * (sum(exp(-param[3] *
            (t + s - tn))))  ####
        if (s > l_t) {
            t <- t + l_t
        } else if ((t + s > horizon) | (U > (lambda_ts/m_t))) {
            t <- t + s
        } else {
            n <- n + 1
            tn[n] <- t + s
            t <- t + s
        }
    }
    return(tn)
}
mu <- 0.1
alpha <- 0.5
omega <- 1
horiz <- 1000
sim <- hawkes(param = c(mu, alpha, omega), horizon = horiz)
stripplot(sim, xlim = c(0, horiz), pch = 16, xlab = "time")
```



When faced with a dataset that is believed to have arisen from a point process, our primary concern is typically the inverse problem; inferring estimates for the model parameters. An EM-algorithm which estimates the parameters μ, α, and ω of given point process data $(x_1, x_2, \ldots, x_n)$ on an interval $(0, T]$ is provided as follows, where it is assumed $\omega << T$.

```
EMHawkes <- function(sims, initguess = c(0.2, 0.2, 0.2), horizon,
    itermax = 100, conv = 1e-05) {
    #### inputs are the simulation, an initial guess for mu and
    #### alpha, the horizon, and the max number of iterations
    mu0 <- initguess[1]
    alpha0 <- initguess[2]
    omega0 <- initguess[3]
    iter <- 0
    itermax <- itermax
    p <- matrix(0, length(sims), length(sims))
    mudiff <- conv + 0.001
    alphadiff <- conv + 0.001
    omegadiff <- conv + 0.001
    p2 <- matrix(0, length(sims), length(sims))
    for (k in 1:length(sims)) {
        for (l in 1:length(sims)) {
            if (k > l) {
                p2[k, l] <- sims[k] - sims[l]
            } else {
                p2[k, l] <- 0
            }
        }
    }
    pdivider <- numeric(length(sims))
    while ((iter <= itermax) & ((abs(mudiff) > conv) | (abs(alphadiff) >
```

```
        conv) | (abs(omegadiff) > conv))) {
        #### algorithm stops when mu and alpha have 'converged'
        for (i in 1:length(sims)) {
            pdivider[i] <- mu0 + alpha0 * omega0 * sum(exp(-omega0 *
                (sims[i] - sims[sims < sims[i]])))
        }
        for (i in 1:length(sims)) {
            for (j in 1:length(sims)) {
                if (i == j) {
                  p[i, i] <- mu0/(pdivider[i])
                } else if (i > j) {
                  p[i, j] <- alpha0 * omega0 * (exp(-omega0 *
                    (sims[i] - sims[j])))/(pdivider[i])
                }
            }
        }
        munew <- sum(diag(p))/horizon
        alphanew <- (sum(p) - sum(diag(p)))/length(sims)
        divisor <- p * p2
        if ((sum(p) - sum(diag(p))) == 0) {
            omeganew <- 0
        } else {
            omeganew <- (sum(p) - sum(diag(p)))/sum(divisor)
        }
        mudiff <- munew - mu0
        alphadiff <- alphanew - alpha0
        omegadiff <- omeganew - omega0
        mu0 <- munew
        alpha0 <- alphanew
        omega0 <- omeganew
        iter <- iter + 1
    }
    result <- c(mu0, alpha0, omega0)
    return(result)
}
EMHawkes(sims = sim, initguess = c(0.2, 0.2, 0.2), horizon = horiz,
    itermax = 100, conv = 1e-05)
```

```
## [1] 0.09067888 0.45043100 0.82558805
```

Spatio-Temporal Point Processes

More recently, self-exciting point processes have been explored in the context of crime modelling. In particular, it has been proposed that certain types of crime, including burglary and gang violence, arise in highly clustered sequences, and therefore can be modelled in much the same way as seismic events, where there is increased risk of aftershocks in close proximity to an earthquake.

A spatio-temporal point process can be defined as

$$\lambda(x, y, t) = \mu(x, y, t) + \sum_{t > t_i} g(t - t_i, x - x_i, y - y_i).$$

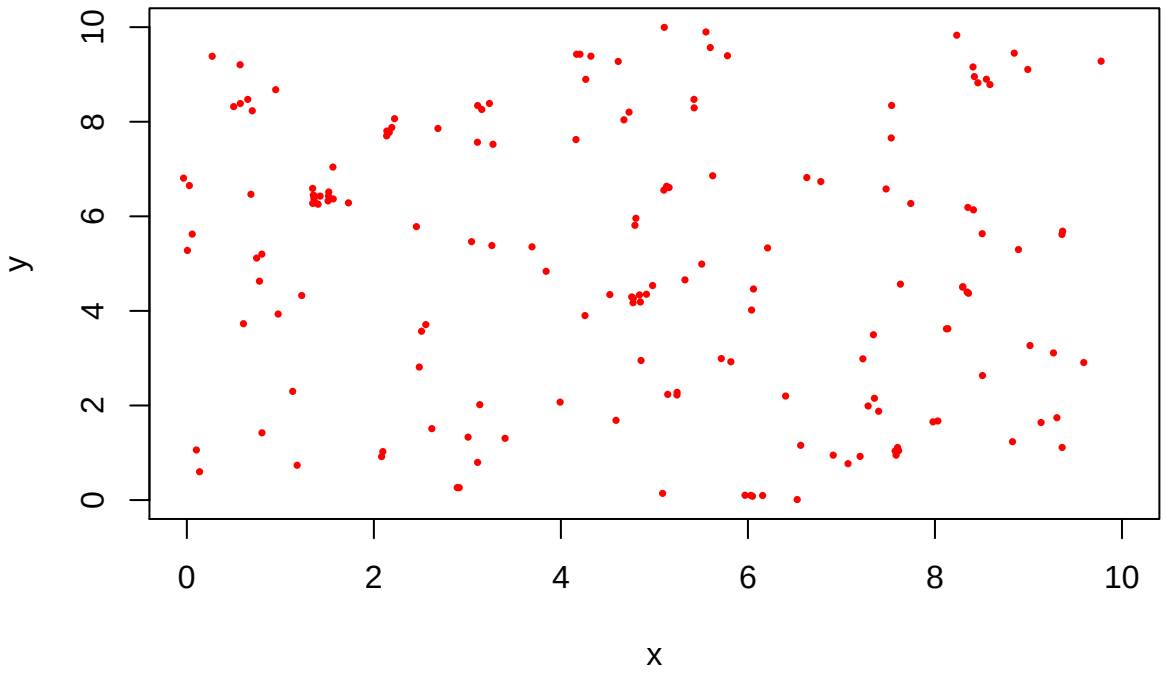
In the case where the triggering function g is exponential in time and Gaussian in space

$$g(t, x, y) = \alpha\omega e^{-\omega t} \cdot \frac{1}{2\pi\sigma^2} e^{-(x^2+y^2)/2\sigma^2},$$

and the background rate μ is constant, we can simulate such a process over time $[0, T]$ in a grid $[0, x] \times [0, y]$ as follows:

```
spacesimulation <- function(xdir = 100, ydir = 100, mu = 0.1,
    alpha = 0.1, omega = 0.1, sigma = 0.1, horizon = 500) {
    ## code generates a spatio-temporal point process over
    ## [0,xdir]x[0,ydir]x[0,horizon] with constant background
    ## rate, and triggering function exponential in time and
    ## gaussian in space
    t <- 0
    n <- 0
    tn <- NULL
    backgroundrate <- mu * xdir * ydir
    while (t < horizon) {
        s <- rexp(1, rate = backgroundrate)
        if ((t + s) > horizon) {
            t <- t + s
        } else {
            n <- n + 1
            tn[n] <- t + s
            t <- t + s
        }
    }
    xvals <- runif(length(tn), min = 0, max = xdir)
    yvals <- runif(length(tn), min = 0, max = ydir)
    triggered <- rpois(length(xvals), lambda = alpha)
    tnreplace <- tn
    xvalsreplace <- xvals
    yvalsreplace <- yvals
    while (sum(triggered) > 0) {
        newt <- numeric(0)
        newx <- numeric(0)
        newy <- numeric(0)
        for (i in 1:length(triggered)) {
            if (triggered[i] > 0) {
                timetrig <- rexp(triggered[i], rate = omega) +
                  tnreplace[i]
                xdisttrig <- rnorm(triggered[i], sd = sigma) +
                  xvalsreplace[i]
                ydisttrig <- rnorm(triggered[i], sd = sigma) +
                  yvalsreplace[i]
                tn <- c(tn, timetrig)
                xvals <- c(xvals, xdisttrig)
                yvals <- c(yvals, ydisttrig)
                newt <- c(newt, timetrig)
                newx <- c(newx, xdisttrig)
                newy <- c(newy, ydisttrig)
            }
        }
        triggered <- rpois(length(newt), lambda = alpha)
        tnreplace <- newt
        xvalsreplace <- newx
        yvalsreplace <- newy
    }
```

```
    ind <- order(tn)
    tn <- tn[ind]
    xvals <- xvals[ind]
    yvals <- yvals[ind]
    newlist <- list(tn, xvals, yvals)
    return(newlist)
}
xdist <- 10
ydist <- 10
mu <- 0.01
alpha <- 0.4
omega <- 0.1
sigma <- 0.1
hor <- 100
spacesim <- spacesimulation(xdir = xdist, ydir = ydist, mu = mu,
    alpha = alpha, omega = omega, sigma = sigma, horizon = hor)
plot(spacesim[[2]], spacesim[[3]], xlab = "x", ylab = "y", pch = 16,
    cex = 0.5, col = "red", xlim = c(0, xdist), ylim = c(0, ydist))
```



Crime Data

We use crime data from the city of Chicago which is publicly available here. We used data where the primary type was listed as 'burglary', and removed entries where there was no information on the location where the crime took place. We also removed entries that were recorded at both the same time and location as another incident.

The following codes were used to model burglary as a self-exciting point process with an isotropic triggering function, using Euclidean distance, Manhattan distance, and Chebyshev distance. The inputs are as follows:

- simstimes - vector containing the times the events took place
- xcoords - vector containing the corresponding x-coordinates where the events took place
- ycoords - vector containing the corresponding y-coordinates where the events took place
- itermax - number of iterations we wish the algorithm to run for

- horizon - the horizon T, where the events have taken place within $[0, T]$
- backgroundbw - the bandwidth of the two dimensional Gaussian kernel used to estimate the background rate
- nearneigh - the number of nearest neighbours used to select D_i in the triggering function
- maxt - maximum t at which the triggering function can have an effect
- maxr - maximum distance at which the triggering function can have an effect.

The output of this function are:

- a vector which gives the estimated probability the input event was a background event
- the times at which an event is said to be triggered
- the corresponding distance at which an event is said to have been triggered. This is the Euclidean, Manhattan or Chebyshev distance depending on what function we are using
- the raw distance in the x-direction at which an event is said to be triggered
- the raw distance in the y-direction at which an event is said to be triggered.

```
EMkdeisotropictrig_fixedbwbg <- function(simstimes, xcoords,
    ycoords, itermax = 100, horizon, backgroundbw = 0.15, nearneigh = 15,
    maxt = 50, maxr = 1) {
    ord <- order(simstimes)
    simstimes <- simstimes[ord]
    xcoords <- xcoords[ord]
    ycoords <- ycoords[ord]
    iter <- 0
    p2 <- matrix(0, length(simstimes), length(simstimes))  ###### matrix with time differences
    for (k in 1:length(simstimes)) {
        for (l in 1:k) {
            if (k > l) {
                p2[k, l] <- simstimes[k] - simstimes[l]
            } else {
                p2[k, l] <- 0
            }
        }
    }
    p3 <- matrix(0, length(simstimes), length(simstimes))  ###### matrix with euclidean distances
    for (k in 1:length(simstimes)) {
        for (l in 1:k) {
            if (k > l) {
                p3[k, l] <- sqrt((xcoords[k] - xcoords[l])^2 +
                  (ycoords[k] - ycoords[l])^2)
            } else {
                p3[k, l] <- 0
            }
        }
    }
    omega = 0.1
    sigmax = 0.3
    initp <- matrix(0, length(simstimes), length(simstimes))
    for (k in 1:length(simstimes)) {
        for (l in 1:k) {
            if (k == l) {
                initp[k, l] <- 1
            } else if ((k > l)) {
                initp[k, l] <- exp(-omega * p2[k, l]) * exp((-p3[k,
                  l]^2/(2 * sigmax^2)))/(2 * pi * sigmax^2)
```

```
        }
    }
}
dividep <- numeric(length(simstimes))
for (i in 1:length(dividep)) {
    dividep[i] <- sum(initp[i, ])
}
p <- matrix(0, length(simstimes), length(simstimes))
for (k in 1:length(simstimes)) {
    for (l in 1:k) {
        p[k, l] <- initp[k, l]/dividep[k]
    }
}
p_x <- matrix(0, length(simstimes), length(simstimes))  ###### matrix with x differences
for (k in 1:length(simstimes)) {
    for (l in 1:k) {
        if (k > l) {
            p_x[k, l] <- xcoords[k] - xcoords[l]
        } else {
            p_x[k, l] <- 0
        }
    }
}
p_y <- matrix(0, length(simstimes), length(simstimes))  ###### matrix with y differences
for (k in 1:length(simstimes)) {
    for (l in 1:k) {
        if (k > l) {
            p_y[k, l] <- ycoords[k] - ycoords[l]
        } else {
            p_y[k, l] <- 0
        }
    }
}
while ((iter < itermax) & (sum(diag(p)) < (0.999 * length(simstimes)))) {
    index <- 1:length(simstimes)
    sampleindex <- numeric(length(simstimes))
    backgroundtimes <- NULL
    backgroundx <- NULL
    backgroundy <- NULL
    trigtimes <- NULL
    trigdist <- NULL
    trigxdir <- NULL
    trigydir <- NULL
    backnum <- 0
    trignum <- 0
    for (i in 1:length(simstimes)) {
        samp <- sample(index, size = 1, prob = p[i, ])
        sampleindex[i] <- samp
        if (samp == i) {
            backnum <- backnum + 1
            backgroundtimes[backnum] <- simstimes[i]
            backgroundx[backnum] <- xcoords[i]
            backgroundy[backnum] <- ycoords[i]
```

```
        } else {
            trignum <- trignum + 1
            trigtimes[trignum] <- p2[i, samp]
            trigdist[trignum] <- p3[i, samp]
            trigxdir[trignum] <- p_x[i, samp]
            trigydir[trignum] <- p_y[i, samp]
        }
    }
    ###### to find nearest neighbours for triggered times, we scale
    ###### them to have unit variance
    trigrate <- trignum/length(simstimes)
    nn2 <- min((nearneigh + 1), trignum)
    sdtrigtimes <- sd(trigtimes)
    sdtrigdist <- sd(trigdist)
    scaledtrigtimes <- trigtimes/sdtrigtimes
    scaledtrigdist <- trigdist/sdtrigdist
    scaleddistance <- matrix(0, length(trigtimes), length(trigtimes))  ###### matrix to find scaled
    for (k in 1:length(trigtimes)) {
        for (l in 1:length(trigtimes)) {
            scaleddistance[k, l] <- sqrt((scaledtrigtimes[k] -
              scaledtrigtimes[l])^2 + (scaledtrigdist[k] -
              scaledtrigdist[l])^2)
        }
    }
    D_trig <- numeric(length(trigtimes))
    for (i in 1:length(D_trig)) {
        D_trig[i] <- scaleddistance[i, ][order(scaleddistance[i,
            ])[nn2]]
    }
    normconst <- numeric(length(trigtimes))
    for (i in 1:length(normconst)) {
        normconst[i] <- 0.5 * (1 + erf(trigdist[i]/(sqrt(2) *
            D_trig[i] * sdtrigdist)))
    }
    g <- matrix(0, length(simstimes), length(simstimes))  ###### matrix with time differences
    registerDoParallel(numCores)
    gparallel <- foreach(k = 1:length(simstimes)) %dopar%
        {
            grow <- numeric(length(simstimes))
            for (l in 1:k) {
              if ((k > l) & (p2[k, l] < maxt) & (abs(p3[k,
                l]) < maxr) & (abs(p3[k, l]) > 0)) {
                grow[l] <- sum(exp(-((p3[k, l] - trigdist)^2)/(2 *
                  (sdtrigdist^2) * (D_trig^2))) * (exp(-((p2[k,
                  l] - trigtimes)^2)/(2 * (sdtrigtimes^2) *
                  (D_trig^2))) + exp(-((-p2[k, l] - trigtimes)^2)/(2 *
                  (sdtrigtimes^2) * (D_trig^2))))/((D_trig^2) *
                  normconst * 2 * pi * p3[k, l]))
              }
            }
            grow
        }
    for (i in 1:length(simstimes)) {
```

```
            g[i, ] <- gparallel[[i]]/(length(simstimes) * sdtrigtimes *
                sdtrigdist * ((2 * pi)))
        }
        rm(gparallel)
        bgpar <- foreach(i = 1:length(simstimes)) %dopar% {
            bgrate <- 0
            for (j in 1:length(backgroundx)) {
                bgrate <- bgrate + exp(-((xcoords[i] - backgroundx[j])^2)/(2 *
                  backgroundbw^2) - ((ycoords[i] - backgroundy[j])^2)/(2 *
                  backgroundbw^2))/(backgroundbw^2)
            }
            bgrate
        }
        bgrate <- numeric(length(simstimes))
        for (i in 1:length(simstimes)) {
            bgrate[i] <- bgpar[[i]]/(horizon * 2 * pi)
        }
        rm(bgpar)
        dividep <- numeric(length(simstimes))
        for (i in 1:length(dividep)) {
            dividep[i] <- sum(g[i, ])
        }
        ppar <- foreach(i = 1:length(simstimes)) %dopar% {
            pnum <- numeric(length(simstimes))
            for (j in 1:length(simstimes)) {
                if (i == j) {
                  pnum[j] <- bgrate[i]/(bgrate[i] + dividep[i])
                } else if (i > j) {
                  pnum[j] <- g[i, j]/(bgrate[i] + dividep[i])
                }
            }
            pnum
        }
        p <- matrix(0, length(simstimes), length(simstimes))
        for (i in 1:length(simstimes)) {
            p[i, ] <- ppar[[i]]
        }
        rm(ppar)
        iter <- iter + 1
        print(iter)
        print(trigrate)
    }
    newlist <- list(diag(p), trigtimes, trigdist, trigxdir, trigydir)
    return(newlist)
}


EMkdemanhattrig_fixedbwbg <- function(simstimes, xcoords, ycoords,
    itermax = 100, horizon, backgroundbw = 0.15, nearneigh = 15,
    maxt = 50, maxr = 1) {
    ord <- order(simstimes)
    simstimes <- simstimes[ord]
    xcoords <- xcoords[ord]
```

```
ycoords <- ycoords[ord]
iter <- 0
p2 <- matrix(0, length(simstimes), length(simstimes))  ###### matrix with time differences
for (k in 1:length(simstimes)) {
    for (l in 1:k) {
        if (k > l) {
            p2[k, l] <- simstimes[k] - simstimes[l]
        } else {
            p2[k, l] <- 0
        }
    }
}
pstarter <- matrix(0, length(simstimes), length(simstimes))  ###### matrix with manhattan distances
for (k in 1:length(simstimes)) {
    for (l in 1:k) {
        if (k > l) {
            pstarter[k, l] <- sqrt((xcoords[k] - xcoords[l])^2 +
              (ycoords[k] - ycoords[l])^2)
        } else {
            pstarter[k, l] <- 0
        }
    }
}
omega = 0.1
sigma = 0.3
initp <- matrix(0, length(simstimes), length(simstimes))
for (k in 1:length(simstimes)) {
    for (l in 1:k) {
        if (k == l) {
            initp[k, l] <- 0.1
        } else if ((k > l)) {
            initp[k, l] <- exp(-omega * p2[k, l]) * exp((-pstarter[k,
              l]^2)/(2 * sigma^2))
        }
    }
}
p3 <- matrix(0, length(simstimes), length(simstimes))  ###### matrix with manhattan distances
for (k in 1:length(simstimes)) {
    for (l in 1:k) {
        if (k > l) {
            p3[k, l] <- (abs(xcoords[k] - xcoords[l]) + abs(ycoords[k] -
              ycoords[l]))
        } else {
            p3[k, l] <- 0
        }
    }
}
dividep <- numeric(length(simstimes))
for (i in 1:length(dividep)) {
    dividep[i] <- sum(initp[i, ])
}
p <- matrix(0, length(simstimes), length(simstimes))
for (k in 1:length(simstimes)) {
```

```
    for (l in 1:k) {
        p[k, l] <- initp[k, l]/dividep[k]
    }
}
rm(initp)
rm(pstarter)
p_x <- matrix(0, length(simstimes), length(simstimes))  ###### matrix with x differences
for (k in 1:length(simstimes)) {
    for (l in 1:k) {
        if (k > l) {
            p_x[k, l] <- xcoords[k] - xcoords[l]
        } else {
            p_x[k, l] <- 0
        }
    }
}
p_y <- matrix(0, length(simstimes), length(simstimes))  ###### matrix with y differences
for (k in 1:length(simstimes)) {
    for (l in 1:k) {
        if (k > l) {
            p_y[k, l] <- ycoords[k] - ycoords[l]
        } else {
            p_y[k, l] <- 0
        }
    }
}
while ((iter < itermax) & (sum(diag(p)) < (0.999 * length(simstimes)))) {
    index <- 1:length(simstimes)
    sampleindex <- numeric(length(simstimes))
    backgroundtimes <- NULL
    backgroundx <- NULL
    backgroundy <- NULL
    trigtimes <- NULL
    trigdist <- NULL
    trigxdir <- NULL
    trigydir <- NULL
    backnum <- 0
    trignum <- 0
    for (i in 1:length(simstimes)) {
        samp <- sample(index, size = 1, prob = p[i, ])
        sampleindex[i] <- samp
        if (samp == i) {
            backnum <- backnum + 1
            backgroundtimes[backnum] <- simstimes[i]
            backgroundx[backnum] <- xcoords[i]
            backgroundy[backnum] <- ycoords[i]
        } else {
            trignum <- trignum + 1
            trigtimes[trignum] <- p2[i, samp]
            trigdist[trignum] <- p3[i, samp]
            trigxdir[trignum] <- p_x[i, samp]
            trigydir[trignum] <- p_y[i, samp]
        }
```

```
}
###### to find nearest neighbours for triggered times, we scale
###### them to have unit variance
trigrate <- trignum/length(simstimes)
nn2 <- min((nearneigh + 1), trignum)
sdtrigtimes <- sd(trigtimes)
sdtrigdist <- sd(trigdist)
scaledtrigtimes <- trigtimes/sdtrigtimes
scaledtrigdist <- trigdist/sdtrigdist
scaleddistance <- matrix(0, length(trigtimes), length(trigtimes))  ###### matrix to find scaled
for (k in 1:length(trigtimes)) {
    for (l in 1:length(trigtimes)) {
        scaleddistance[k, l] <- sqrt((scaledtrigtimes[k] -
          scaledtrigtimes[l])^2 + (scaledtrigdist[k] -
          scaledtrigdist[l])^2)
    }
}
D_trig <- numeric(length(trigtimes))
for (i in 1:length(D_trig)) {
    D_trig[i] <- scaleddistance[i, ][order(scaleddistance[i,
        ])[nn2]]
}
normconst <- numeric(length(trigtimes))
for (i in 1:length(normconst)) {
    normconst[i] <- 0.5 * (1 + erf(trigdist[i]/(sqrt(2) *
        D_trig[i] * sdtrigdist)))
}
g <- matrix(0, length(simstimes), length(simstimes))  ###### matrix with time differences
gparallel <- foreach(k = 1:length(simstimes)) %dopar%
    {
        grow <- numeric(length(simstimes))
        for (l in 1:k) {
          if ((k > l) & (p2[k, l] < maxt) & (p3[k, l] >
            0) & (abs(p3[k, l]) < maxr)) {
            grow[l] <- sum(exp(-((p3[k, l] - trigdist)^2)/(2 *
              (sdtrigdist^2) * (D_trig^2))) * (exp(-((p2[k,
              l] - trigtimes)^2)/(2 * (sdtrigtimes^2) *
              (D_trig^2))) + exp(-((-p2[k, l] - trigtimes)^2)/(2 *
              (sdtrigtimes^2) * (D_trig^2))))/((D_trig^2) *
              normconst * 4 * p3[k, l]))
          }
        }
        grow
    }
for (i in 1:length(simstimes)) {
    g[i, ] <- gparallel[[i]]/(length(simstimes) * sdtrigtimes *
        sdtrigdist * ((2 * pi)))
}
rm(gparallel)
bgpar <- foreach(i = 1:length(simstimes)) %dopar% {
    bgrate <- 0
    for (j in 1:length(backgroundx)) {
        if ((xcoords[i] != backgroundx[j]) | (ycoords[i] !=
```

```
                    backgroundy[j])) {
                    bgrate <- bgrate + exp(-((xcoords[i] - backgroundx[j])^2)/(2 *
                      backgroundbw^2) - ((ycoords[i] - backgroundy[j])^2)/(2 *
                      backgroundbw^2))/(backgroundbw^2)
                }
            }
            bgrate
        }
        bgrate <- numeric(length(simstimes))
        for (i in 1:length(simstimes)) {
            bgrate[i] <- bgpar[[i]]/(horizon * 2 * pi)
        }
        rm(bgpar)
        dividep <- numeric(length(simstimes))
        for (i in 1:length(dividep)) {
            dividep[i] <- sum(g[i, ])
        }
        ppar <- foreach(i = 1:length(simstimes)) %dopar% {
            pnum <- numeric(length(simstimes))
            for (j in 1:length(simstimes)) {
                if (i == j) {
                  pnum[j] <- bgrate[i]/(bgrate[i] + dividep[i])
                } else if (i > j) {
                  pnum[j] <- g[i, j]/(bgrate[i] + dividep[i])
                }
            }
            pnum
        }
        p <- matrix(0, length(simstimes), length(simstimes))
        for (i in 1:length(simstimes)) {
            p[i, ] <- ppar[[i]]
        }
        rm(ppar)
        iter <- iter + 1
        print(iter)
        print(trigrate)
    }
    newlist <- list(diag(p), trigtimes, trigdist, trigxdir, trigydir)
    return(newlist)
}

EMkdechebyshevtrig_fixedbwbg <- function(simstimes, xcoords,
    ycoords, itermax = 100, horizon, backgroundbw = 0.15, nearneigh = 15,
    maxt = 50, maxr = 1) {
    ord <- order(simstimes)
    simstimes <- simstimes[ord]
    xcoords <- xcoords[ord]
    ycoords <- ycoords[ord]
    iter <- 0
    p2 <- matrix(0, length(simstimes), length(simstimes))  ###### matrix with time differences
    for (k in 1:length(simstimes)) {
        for (l in 1:k) {
            if (k > l) {
```

```
            p2[k, l] <- simstimes[k] - simstimes[l]
        } else {
            p2[k, l] <- 0
        }
    }
}
pstarter <- matrix(0, length(simstimes), length(simstimes))  ###### matrix with manhattan distances
for (k in 1:length(simstimes)) {
    for (l in 1:k) {
        if (k > l) {
            pstarter[k, l] <- sqrt((xcoords[k] - xcoords[l])^2 +
              (ycoords[k] - ycoords[l])^2)
        } else {
            pstarter[k, l] <- 0
        }
    }
}
omega = 0.1
sigma = 0.5
initp <- matrix(0, length(simstimes), length(simstimes))
for (k in 1:length(simstimes)) {
    for (l in 1:k) {
        if (k == l) {
            initp[k, l] <- 1
        } else if ((k > l)) {
            initp[k, l] <- exp(-omega * p2[k, l]) * exp((-pstarter[k,
              l]^2)/(2 * sigma^2))
        }
    }
}
p3 <- matrix(0, length(simstimes), length(simstimes))  ###### matrix with manhattan distances
for (k in 1:length(simstimes)) {
    for (l in 1:k) {
        if (k > l) {
            p3[k, l] <- max(abs(xcoords[k] - xcoords[l]),
              abs(ycoords[k] - ycoords[l]))
        } else {
            p3[k, l] <- 0
        }
    }
}
dividep <- numeric(length(simstimes))
for (i in 1:length(dividep)) {
    dividep[i] <- sum(initp[i, ])
}
p <- matrix(0, length(simstimes), length(simstimes))
for (k in 1:length(simstimes)) {
    for (l in 1:k) {
        p[k, l] <- initp[k, l]/dividep[k]
    }
}
rm(initp)
rm(pstarter)
```

```
    p_x <- matrix(0, length(simstimes), length(simstimes))  ###### matrix with x differences
    for (k in 1:length(simstimes)) {
        for (l in 1:k) {
            if (k > l) {
                p_x[k, l] <- xcoords[k] - xcoords[l]
            } else {
                p_x[k, l] <- 0
            }
        }
    }
    p_y <- matrix(0, length(simstimes), length(simstimes))  ###### matrix with y differences
    for (k in 1:length(simstimes)) {
        for (l in 1:k) {
            if (k > l) {
                p_y[k, l] <- ycoords[k] - ycoords[l]
            } else {
                p_y[k, l] <- 0
            }
        }
    }
    while ((iter < itermax) & (sum(diag(p)) < (0.999 * length(simstimes)))) {
        index <- 1:length(simstimes)
        sampleindex <- numeric(length(simstimes))
        backgroundtimes <- NULL
        backgroundx <- NULL
        backgroundy <- NULL
        trigtimes <- NULL
        trigdist <- NULL
        trigxdir <- NULL
        trigydir <- NULL
        backnum <- 0
        trignum <- 0
        for (i in 1:length(simstimes)) {
            samp <- sample(index, size = 1, prob = p[i, ])
            sampleindex[i] <- samp
            if (samp == i) {
                backnum <- backnum + 1
                backgroundtimes[backnum] <- simstimes[i]
                backgroundx[backnum] <- xcoords[i]
                backgroundy[backnum] <- ycoords[i]
            } else {
                trignum <- trignum + 1
                trigtimes[trignum] <- p2[i, samp]
                trigdist[trignum] <- p3[i, samp]
                trigxdir[trignum] <- p_x[i, samp]
                trigydir[trignum] <- p_y[i, samp]
            }
        }
        ###### to find nearest neighbours for triggered times, we scale
        ###### them to have unit variance
        trigrate <- trignum/length(simstimes)
        nn2 <- min((nearneigh + 1), trignum)
        sdtrigtimes <- sd(trigtimes)
```

```
sdtrigdist <- sd(trigdist)
scaledtrigtimes <- trigtimes/sdtrigtimes
scaledtrigdist <- trigdist/sdtrigdist
scaleddistance <- matrix(0, length(trigtimes), length(trigtimes))  ###### matrix to find scaled
for (k in 1:length(trigtimes)) {
    for (l in 1:length(trigtimes)) {
        scaleddistance[k, l] <- sqrt((scaledtrigtimes[k] -
          scaledtrigtimes[l])^2 + (scaledtrigdist[k] -
          scaledtrigdist[l])^2)
    }
}
D_trig <- numeric(length(trigtimes))
for (i in 1:length(D_trig)) {
    D_trig[i] <- scaleddistance[i, ][order(scaleddistance[i,
        ])[nn2]]
}
normconst <- numeric(length(trigtimes))
for (i in 1:length(normconst)) {
    normconst[i] <- 0.5 * (1 + erf(trigdist[i]/(sqrt(2) *
        D_trig[i] * sdtrigdist)))
}
g <- matrix(0, length(simstimes), length(simstimes))  ###### matrix with time differences
gparallel <- foreach(k = 1:length(simstimes)) %dopar%
    {
        grow <- numeric(length(simstimes))
        for (l in 1:k) {
          if ((k > l) & (p2[k, l] < maxt) & (p3[k, l] >
            0) & (abs(p3[k, l]) < maxr)) {
            grow[l] <- sum(exp(-((p3[k, l] - trigdist)^2)/(2 *
              (sdtrigdist^2) * (D_trig^2))) * (exp(-((p2[k,
              l] - trigtimes)^2)/(2 * (sdtrigtimes^2) *
              (D_trig^2))) + exp(-((-p2[k, l] - trigtimes)^2)/(2 *
              (sdtrigtimes^2) * (D_trig^2))))/((D_trig^2) *
              normconst * 8 * p3[k, l]))
          }
        }
        grow
    }
for (i in 1:length(simstimes)) {
    g[i, ] <- gparallel[[i]]/(length(simstimes) * sdtrigtimes *
        sdtrigdist * ((2 * pi)))
}
rm(gparallel)
bgpar <- foreach(i = 1:length(simstimes)) %dopar% {
    bgrate <- 0
    for (j in 1:length(backgroundx)) {
        if ((xcoords[i] != backgroundx[j]) | (ycoords[i] !=
          backgroundy[j])) {
          bgrate <- bgrate + exp(-((xcoords[i] - backgroundx[j])^2)/(2 *
            backgroundbw^2) - ((ycoords[i] - backgroundy[j])^2)/(2 *
            backgroundbw^2))/(backgroundbw^2)
        }
    }
```

```
            bgrate
        }
        bgrate <- numeric(length(simstimes))
        for (i in 1:length(simstimes)) {
            bgrate[i] <- bgpar[[i]]/(horizon * 2 * pi)
        }
        rm(bgpar)
        dividep <- numeric(length(simstimes))
        for (i in 1:length(dividep)) {
            dividep[i] <- sum(g[i, ])
        }
        ppar <- foreach(i = 1:length(simstimes)) %dopar% {
            pnum <- numeric(length(simstimes))
            for (j in 1:length(simstimes)) {
                if (i == j) {
                  pnum[j] <- bgrate[i]/(bgrate[i] + dividep[i])
                } else if (i > j) {
                  pnum[j] <- g[i, j]/(bgrate[i] + dividep[i])
                }
            }
            pnum
        }
        p <- matrix(0, length(simstimes), length(simstimes))
        for (i in 1:length(simstimes)) {
            p[i, ] <- ppar[[i]]
        }
        rm(ppar)
        iter <- iter + 1
        print(iter)
        print(trigrate)
    }
    newlist <- list(diag(p), trigtimes, trigdist, trigxdir, trigydir)
    return(newlist)
}
```

These are the functions used to estimate the self-exciting point processes with isotropic triggering functions using Euclidean, Manhattan and Chebyshev distance, where the background rate is adapted so the background rate is estimated while omitting the contribution of the event which took place at a point.

```
EMkdeisotropictrig_fixedbwadaptedbg <- function(simstimes, xcoords,
    ycoords, itermax = 100, horizon, backgroundbw = 0.15, nearneigh = 15,
    maxt = 50, maxr = 1) {
    ord <- order(simstimes)
    simstimes <- simstimes[ord]
    xcoords <- xcoords[ord]
    ycoords <- ycoords[ord]
    iter <- 0
    p2 <- matrix(0, length(simstimes), length(simstimes))  ###### matrix with time differences
    for (k in 1:length(simstimes)) {
        for (l in 1:k) {
            if (k > l) {
                p2[k, l] <- simstimes[k] - simstimes[l]
            } else {
                p2[k, l] <- 0
```

```
        }
    }
}
p3 <- matrix(0, length(simstimes), length(simstimes))  ###### matrix with euclidean distances
for (k in 1:length(simstimes)) {
    for (l in 1:k) {
        if (k > l) {
            p3[k, l] <- sqrt((xcoords[k] - xcoords[l])^2 +
              (ycoords[k] - ycoords[l])^2)
        } else {
            p3[k, l] <- 0
        }
    }
}
omega = 0.1
sigmax = 0.3
initp <- matrix(0, length(simstimes), length(simstimes))
for (k in 1:length(simstimes)) {
    for (l in 1:k) {
        if (k == l) {
            initp[k, l] <- 1
        } else if ((k > l)) {
            initp[k, l] <- exp(-omega * p2[k, l]) * exp((-p3[k,
              l]^2/(2 * sigmax^2)))/(2 * pi * sigmax^2)
        }
    }
}
dividep <- numeric(length(simstimes))
for (i in 1:length(dividep)) {
    dividep[i] <- sum(initp[i, ])
}
p <- matrix(0, length(simstimes), length(simstimes))
for (k in 1:length(simstimes)) {
    for (l in 1:k) {
        p[k, l] <- initp[k, l]/dividep[k]
    }
}
p_x <- matrix(0, length(simstimes), length(simstimes))  ###### matrix with x differences
for (k in 1:length(simstimes)) {
    for (l in 1:k) {
        if (k > l) {
            p_x[k, l] <- xcoords[k] - xcoords[l]
        } else {
            p_x[k, l] <- 0
        }
    }
}
p_y <- matrix(0, length(simstimes), length(simstimes))  ###### matrix with y differences
for (k in 1:length(simstimes)) {
    for (l in 1:k) {
        if (k > l) {
            p_y[k, l] <- ycoords[k] - ycoords[l]
        } else {
```

```
            p_y[k, l] <- 0
        }
    }
}
while ((iter < itermax) & (sum(diag(p)) < (0.999 * length(simstimes)))) {
    index <- 1:length(simstimes)
    sampleindex <- numeric(length(simstimes))
    backgroundtimes <- NULL
    backgroundx <- NULL
    backgroundy <- NULL
    trigtimes <- NULL
    trigdist <- NULL
    trigxdir <- NULL
    trigydir <- NULL
    backnum <- 0
    trignum <- 0
    for (i in 1:length(simstimes)) {
        samp <- sample(index, size = 1, prob = p[i, ])
        sampleindex[i] <- samp
        if (samp == i) {
            backnum <- backnum + 1
            backgroundtimes[backnum] <- simstimes[i]
            backgroundx[backnum] <- xcoords[i]
            backgroundy[backnum] <- ycoords[i]
        } else {
            trignum <- trignum + 1
            trigtimes[trignum] <- p2[i, samp]
            trigdist[trignum] <- p3[i, samp]
            trigxdir[trignum] <- p_x[i, samp]
            trigydir[trignum] <- p_y[i, samp]
        }
    }
    ###### to find nearest neighbours for triggered times, we scale
    ###### them to have unit variance
    trigrate <- trignum/length(simstimes)
    expecttrig <- (length(simstimes) - sum(diag(p)))/(length(simstimes))
    # nn1 <- min((nearneigh_1 + 1), backnum) #### this ends up
    # finding the nearest neighbour (nn1 - 1)
    nn2 <- min((nearneigh + 1), trignum)
    sdtrigtimes <- sd(trigtimes)
    sdtrigdist <- sd(trigdist)
    scaledtrigtimes <- trigtimes/sdtrigtimes
    scaledtrigdist <- trigdist/sdtrigdist
    scaleddistance <- matrix(0, length(trigtimes), length(trigtimes))  ###### matrix to find scaled
    for (k in 1:length(trigtimes)) {
        for (l in 1:length(trigtimes)) {
            scaleddistance[k, l] <- sqrt((scaledtrigtimes[k] -
              scaledtrigtimes[l])^2 + (scaledtrigdist[k] -
              scaledtrigdist[l])^2)
        }
    }
    D_trig <- numeric(length(trigtimes))
    for (i in 1:length(D_trig)) {
```

```
    D_trig[i] <- scaleddistance[i, ][order(scaleddistance[i,
        ])[nn2]]
}
normconst <- numeric(length(trigtimes))
for (i in 1:length(normconst)) {
    normconst[i] <- 0.5 * (1 + erf(trigdist[i]/(sqrt(2) *
        D_trig[i] * sdtrigdist)))
}
g <- matrix(0, length(simstimes), length(simstimes))  ###### matrix with time differences
registerDoParallel(numCores)
gparallel <- foreach(k = 1:length(simstimes)) %dopar%
    {
        grow <- numeric(length(simstimes))
        for (l in 1:k) {
          if ((k > l) & (p2[k, l] > 0) & (p2[k, l] <
            maxt) & (abs(p3[k, l]) < maxr) & (abs(p3[k,
            l]) > 0)) {
            grow[l] <- expecttrig * sum(exp(-((p3[k,
              l] - trigdist)^2)/(2 * (sdtrigdist^2) *
              (D_trig^2))) * (exp(-((p2[k, l] - trigtimes)^2)/(2 *
              (sdtrigtimes^2) * (D_trig^2))) + exp(-((-p2[k,
              l] - trigtimes)^2)/(2 * (sdtrigtimes^2) *
              (D_trig^2))))/((D_trig^2) * normconst *
              2 * pi * p3[k, l]))
          }
        }
        grow
    }
for (i in 1:length(simstimes)) {
    g[i, ] <- gparallel[[i]]/(trignum * sdtrigtimes *
        sdtrigdist * ((2 * pi)))
}
rm(gparallel)
bgpar <- foreach(i = 1:length(simstimes)) %dopar% {
    bgrate <- 0
    for (j in 1:length(simstimes)) {
        if ((xcoords[i] != xcoords[j]) | (ycoords[i] !=
          ycoords[j])) {
          bgrate <- bgrate + p[j, j] * exp(-((xcoords[i] -
            xcoords[j])^2)/(2 * backgroundbw^2) - ((ycoords[i] -
            ycoords[j])^2)/(2 * backgroundbw^2))/(backgroundbw^2)
        }
    }
    bgrate
}
bgrate <- numeric(length(simstimes))
for (i in 1:length(simstimes)) {
    bgrate[i] <- bgpar[[i]]/(horizon * 2 * pi)
}
rm(bgpar)
dividep <- numeric(length(simstimes))
for (i in 1:length(dividep)) {
    dividep[i] <- sum(g[i, ])
```

```
        }
        ppar <- foreach(i = 1:length(simstimes)) %dopar% {
            pnum <- numeric(length(simstimes))
            for (j in 1:length(simstimes)) {
                if (i == j) {
                  pnum[j] <- bgrate[i]/(bgrate[i] + dividep[i])
                } else if (i > j) {
                  pnum[j] <- g[i, j]/(bgrate[i] + dividep[i])
                }
            }
            pnum
        }
        p <- matrix(0, length(simstimes), length(simstimes))
        for (i in 1:length(simstimes)) {
            p[i, ] <- ppar[[i]]
        }
        expecttrig <- (length(simstimes) - sum(diag(p)))/(length(simstimes))
        rm(ppar)
        iter <- iter + 1
        print(iter)
        print(expecttrig)
    }
    newlist <- list(diag(p), trigtimes, trigdist, trigxdir, trigydir)
    return(newlist)
}

EMkdemanhattrig_fixedbwadaptedbg <- function(simstimes, xcoords,
    ycoords, itermax = 100, horizon, backgroundbw = 0.15, nearneigh = 15,
    maxt = 50, maxr = 1) {
    ord <- order(simstimes)
    simstimes <- simstimes[ord]
    xcoords <- xcoords[ord]
    ycoords <- ycoords[ord]
    iter <- 0
    p2 <- matrix(0, length(simstimes), length(simstimes))  ###### matrix with time differences
    for (k in 1:length(simstimes)) {
        for (l in 1:k) {
            if (k > l) {
                p2[k, l] <- simstimes[k] - simstimes[l]
            } else {
                p2[k, l] <- 0
            }
        }
    }
    pstarter <- matrix(0, length(simstimes), length(simstimes))  ###### matrix with manhattan distances
    for (k in 1:length(simstimes)) {
        for (l in 1:k) {
            if (k > l) {
                pstarter[k, l] <- sqrt((xcoords[k] - xcoords[l])^2 +
                  (ycoords[k] - ycoords[l])^2)
            } else {
                pstarter[k, l] <- 0
            }
```

```
    }
}
omega = 0.1
sigma = 0.3
initp <- matrix(0, length(simstimes), length(simstimes))
for (k in 1:length(simstimes)) {
    for (l in 1:k) {
        if (k == l) {
            initp[k, l] <- 1
        } else if ((k > l)) {
            initp[k, l] <- exp(-omega * p2[k, l]) * exp((-pstarter[k,
              l]^2)/(2 * sigma^2))
        }
    }
}
p3 <- matrix(0, length(simstimes), length(simstimes))  ###### matrix with manhattan distances
for (k in 1:length(simstimes)) {
    for (l in 1:k) {
        if (k > l) {
            p3[k, l] <- (abs(xcoords[k] - xcoords[l]) + abs(ycoords[k] -
              ycoords[l]))
        } else {
            p3[k, l] <- 0
        }
    }
}
dividep <- numeric(length(simstimes))
for (i in 1:length(dividep)) {
    dividep[i] <- sum(initp[i, ])
}
p <- matrix(0, length(simstimes), length(simstimes))
for (k in 1:length(simstimes)) {
    for (l in 1:k) {
        p[k, l] <- initp[k, l]/dividep[k]
    }
}
rm(initp)
rm(pstarter)
p_x <- matrix(0, length(simstimes), length(simstimes))  ###### matrix with x differences
for (k in 1:length(simstimes)) {
    for (l in 1:k) {
        if (k > l) {
            p_x[k, l] <- xcoords[k] - xcoords[l]
        } else {
            p_x[k, l] <- 0
        }
    }
}
p_y <- matrix(0, length(simstimes), length(simstimes))  ###### matrix with y differences
for (k in 1:length(simstimes)) {
    for (l in 1:k) {
        if (k > l) {
            p_y[k, l] <- ycoords[k] - ycoords[l]
```

```
            } else {
                p_y[k, l] <- 0
            }
        }
    }
    while ((iter < itermax) & (sum(diag(p)) < (0.999 * length(simstimes)))) {
        index <- 1:length(simstimes)
        sampleindex <- numeric(length(simstimes))
        backgroundtimes <- NULL
        backgroundx <- NULL
        backgroundy <- NULL
        trigtimes <- NULL
        trigdist <- NULL
        trigxdir <- NULL
        trigydir <- NULL
        backnum <- 0
        trignum <- 0
        for (i in 1:length(simstimes)) {
            samp <- sample(index, size = 1, prob = p[i, ])
            sampleindex[i] <- samp
            if (samp == i) {
                backnum <- backnum + 1
                backgroundtimes[backnum] <- simstimes[i]
                backgroundx[backnum] <- xcoords[i]
                backgroundy[backnum] <- ycoords[i]
            } else {
                trignum <- trignum + 1
                trigtimes[trignum] <- p2[i, samp]
                trigdist[trignum] <- p3[i, samp]
                trigxdir[trignum] <- p_x[i, samp]
                trigydir[trignum] <- p_y[i, samp]
            }
        }
        ###### to find nearest neighbours for triggered times, we scale
        ###### them to have unit variance
        trigrate <- trignum/length(simstimes)
        expecttrig <- (length(simstimes) - sum(diag(p)))/(length(simstimes))
        nn2 <- min((nearneigh + 1), trignum)
        sdtrigtimes <- sd(trigtimes)
        sdtrigdist <- sd(trigdist)
        scaledtrigtimes <- trigtimes/sdtrigtimes
        scaledtrigdist <- trigdist/sdtrigdist
        scaleddistance <- matrix(0, length(trigtimes), length(trigtimes))  ###### matrix to find scaled
        for (k in 1:length(trigtimes)) {
            for (l in 1:length(trigtimes)) {
                scaleddistance[k, l] <- sqrt((scaledtrigtimes[k] -
                  scaledtrigtimes[l])^2 + (scaledtrigdist[k] -
                  scaledtrigdist[l])^2)
            }
        }
        D_trig <- numeric(length(trigtimes))
        for (i in 1:length(D_trig)) {
            D_trig[i] <- scaleddistance[i, ][order(scaleddistance[i,
```

```
        ])[nn2]]
}
normconst <- numeric(length(trigtimes))
for (i in 1:length(normconst)) {
    normconst[i] <- 0.5 * (1 + erf(trigdist[i]/(sqrt(2) *
        D_trig[i] * sdtrigdist)))
}
g <- matrix(0, length(simstimes), length(simstimes))  ###### matrix with time differences
gparallel <- foreach(k = 1:length(simstimes)) %dopar%
    {
        grow <- numeric(length(simstimes))
        for (l in 1:k) {
          if ((k > l) & (p2[k, l] > 0) & (p2[k, l] <
            maxt) & (p3[k, l] > 0) & (abs(p3[k, l]) <
            maxr)) {
            grow[l] <- expecttrig * sum(exp(-((p3[k,
              l] - trigdist)^2)/(2 * (sdtrigdist^2) *
              (D_trig^2))) * (exp(-((p2[k, l] - trigtimes)^2)/(2 *
              (sdtrigtimes^2) * (D_trig^2))) + exp(-((-p2[k,
              l] - trigtimes)^2)/(2 * (sdtrigtimes^2) *
              (D_trig^2))))/((D_trig^2) * normconst *
              4 * p3[k, l]))
          }
        }
        grow
    }
for (i in 1:length(simstimes)) {
    g[i, ] <- gparallel[[i]]/(trignum * sdtrigtimes *
        sdtrigdist * ((2 * pi)))
}
rm(gparallel)
bgpar <- foreach(i = 1:length(simstimes)) %dopar% {
    bgrate <- 0
    for (j in 1:length(simstimes)) {
        if ((xcoords[i] != xcoords[j]) | (ycoords[i] !=
          ycoords[j])) {
          bgrate <- bgrate + p[j, j] * exp(-((xcoords[i] -
            xcoords[j])^2)/(2 * backgroundbw^2) - ((ycoords[i] -
            ycoords[j])^2)/(2 * backgroundbw^2))/(backgroundbw^2)
        }
    }
    bgrate
}
bgrate <- numeric(length(simstimes))
for (i in 1:length(simstimes)) {
    bgrate[i] <- bgpar[[i]]/(horizon * 2 * pi)
}
rm(bgpar)
dividep <- numeric(length(simstimes))
for (i in 1:length(dividep)) {
    dividep[i] <- sum(g[i, ])
}
ppar <- foreach(i = 1:length(simstimes)) %dopar% {
```

```
            pnum <- numeric(length(simstimes))
            for (j in 1:length(simstimes)) {
                if (i == j) {
                  pnum[j] <- bgrate[i]/(bgrate[i] + dividep[i])
                } else if (i > j) {
                  pnum[j] <- g[i, j]/(bgrate[i] + dividep[i])
                }
            }
            pnum
        }
        p <- matrix(0, length(simstimes), length(simstimes))
        for (i in 1:length(simstimes)) {
            p[i, ] <- ppar[[i]]
        }
        rm(ppar)
        iter <- iter + 1
        print(iter)
        print(trigrate)
    }
    newlist <- list(diag(p), trigtimes, trigdist, trigxdir, trigydir)
    return(newlist)
}

EMkdechebyshevtrig_fixedbwadaptedbg <- function(simstimes, xcoords,
    ycoords, itermax = 100, horizon, backgroundbw = 0.15, nearneigh = 15,
    maxt = 50, maxr = 1) {
    ord <- order(simstimes)
    simstimes <- simstimes[ord]
    xcoords <- xcoords[ord]
    ycoords <- ycoords[ord]
    iter <- 0
    p2 <- matrix(0, length(simstimes), length(simstimes))  ###### matrix with time differences
    for (k in 1:length(simstimes)) {
        for (l in 1:k) {
            if (k > l) {
                p2[k, l] <- simstimes[k] - simstimes[l]
            } else {
                p2[k, l] <- 0
            }
        }
    }
    pstarter <- matrix(0, length(simstimes), length(simstimes))  ###### matrix with manhattan distances
    for (k in 1:length(simstimes)) {
        for (l in 1:k) {
            if (k > l) {
                pstarter[k, l] <- sqrt((xcoords[k] - xcoords[l])^2 +
                  (ycoords[k] - ycoords[l])^2)
            } else {
                pstarter[k, l] <- 0
            }
        }
    }
    omega = 0.1
```

```
sigma = 0.3
initp <- matrix(0, length(simstimes), length(simstimes))
for (k in 1:length(simstimes)) {
    for (l in 1:k) {
        if (k == l) {
            initp[k, l] <- 1
        } else if ((k > l)) {
            initp[k, l] <- exp(-omega * p2[k, l]) * exp((-pstarter[k,
              l]^2)/(2 * sigma^2))
        }
    }
}
p3 <- matrix(0, length(simstimes), length(simstimes))  ###### matrix with manhattan distances
for (k in 1:length(simstimes)) {
    for (l in 1:k) {
        if (k > l) {
            p3[k, l] <- max(abs(xcoords[k] - xcoords[l]),
              abs(ycoords[k] - ycoords[l]))
        } else {
            p3[k, l] <- 0
        }
    }
}
dividep <- numeric(length(simstimes))
for (i in 1:length(dividep)) {
    dividep[i] <- sum(initp[i, ])
}
p <- matrix(0, length(simstimes), length(simstimes))
for (k in 1:length(simstimes)) {
    for (l in 1:k) {
        p[k, l] <- initp[k, l]/dividep[k]
    }
}
rm(initp)
rm(pstarter)
p_x <- matrix(0, length(simstimes), length(simstimes))  ###### matrix with x differences
for (k in 1:length(simstimes)) {
    for (l in 1:k) {
        if (k > l) {
            p_x[k, l] <- xcoords[k] - xcoords[l]
        } else {
            p_x[k, l] <- 0
        }
    }
}
p_y <- matrix(0, length(simstimes), length(simstimes))  ###### matrix with y differences
for (k in 1:length(simstimes)) {
    for (l in 1:k) {
        if (k > l) {
            p_y[k, l] <- ycoords[k] - ycoords[l]
        } else {
            p_y[k, l] <- 0
        }
```

```
    }
}
while ((iter < itermax) & (sum(diag(p)) < (0.999 * length(simstimes)))) {
    index <- 1:length(simstimes)
    sampleindex <- numeric(length(simstimes))
    backgroundtimes <- NULL
    backgroundx <- NULL
    backgroundy <- NULL
    trigtimes <- NULL
    trigdist <- NULL
    trigxdir <- NULL
    trigydir <- NULL
    backnum <- 0
    trignum <- 0
    for (i in 1:length(simstimes)) {
        samp <- sample(index, size = 1, prob = p[i, ])
        sampleindex[i] <- samp
        if (samp == i) {
            backnum <- backnum + 1
            backgroundtimes[backnum] <- simstimes[i]
            backgroundx[backnum] <- xcoords[i]
            backgroundy[backnum] <- ycoords[i]
        } else {
            trignum <- trignum + 1
            trigtimes[trignum] <- p2[i, samp]
            trigdist[trignum] <- p3[i, samp]
            trigxdir[trignum] <- p_x[i, samp]
            trigydir[trignum] <- p_y[i, samp]
        }
    }
    ###### to find nearest neighbours for triggered times, we scale
    ###### them to have unit variance
    trigrate <- trignum/length(simstimes)
    expecttrig <- (length(simstimes) - sum(diag(p)))/(length(simstimes))
    # nn1 <- min((nearneigh_1 + 1), backnum) #### this ends up
    # finding the nearest neighbour (nn1 - 1)
    nn2 <- min((nearneigh + 1), trignum)
    sdtrigtimes <- sd(trigtimes)
    sdtrigdist <- sd(trigdist)
    scaledtrigtimes <- trigtimes/sdtrigtimes
    scaledtrigdist <- trigdist/sdtrigdist
    scaleddistance <- matrix(0, length(trigtimes), length(trigtimes))  ###### matrix to find scaled
    for (k in 1:length(trigtimes)) {
        for (l in 1:length(trigtimes)) {
            scaleddistance[k, l] <- sqrt((scaledtrigtimes[k] -
              scaledtrigtimes[l])^2 + (scaledtrigdist[k] -
              scaledtrigdist[l])^2)
        }
    }
    D_trig <- numeric(length(trigtimes))
    for (i in 1:length(D_trig)) {
        D_trig[i] <- scaleddistance[i, ][order(scaleddistance[i,
            ])[nn2]]
```

```
}
normconst <- numeric(length(trigtimes))
for (i in 1:length(normconst)) {
    normconst[i] <- 0.5 * (1 + erf(trigdist[i]/(sqrt(2) *
        D_trig[i] * sdtrigdist)))
}
g <- matrix(0, length(simstimes), length(simstimes))  ###### matrix with time differences
gparallel <- foreach(k = 1:length(simstimes)) %dopar%
    {
        grow <- numeric(length(simstimes))
        for (l in 1:k) {
          if ((k > l) & (p2[k, l] > 0) & (p2[k, l] <
            maxt) & (p3[k, l] > 0) & (abs(p3[k, l]) <
            maxr)) {
            grow[l] <- expecttrig * sum(exp(-((p3[k,
              l] - trigdist)^2)/(2 * (sdtrigdist^2) *
              (D_trig^2))) * (exp(-((p2[k, l] - trigtimes)^2)/(2 *
              (sdtrigtimes^2) * (D_trig^2))) + exp(-((-p2[k,
              l] - trigtimes)^2)/(2 * (sdtrigtimes^2) *
              (D_trig^2))))/((D_trig^2) * normconst *
              8 * p3[k, l]))
          }
        }
        grow
    }
for (i in 1:length(simstimes)) {
    g[i, ] <- gparallel[[i]]/(trignum * sdtrigtimes *
        sdtrigdist * ((2 * pi)))
}
rm(gparallel)
bgpar <- foreach(i = 1:length(simstimes)) %dopar% {
    bgrate <- 0
    for (j in 1:length(simstimes)) {
        if ((xcoords[i] != xcoords[j]) | (ycoords[i] !=
          ycoords[j])) {
          bgrate <- bgrate + p[j, j] * exp(-((xcoords[i] -
            xcoords[j])^2)/(2 * backgroundbw^2) - ((ycoords[i] -
            ycoords[j])^2)/(2 * backgroundbw^2))/(backgroundbw^2)
        }
    }
    bgrate
}
bgrate <- numeric(length(simstimes))
for (i in 1:length(simstimes)) {
    bgrate[i] <- bgpar[[i]]/(horizon * 2 * pi)
}
rm(bgpar)
dividep <- numeric(length(simstimes))
for (i in 1:length(dividep)) {
    dividep[i] <- sum(g[i, ])
}
ppar <- foreach(i = 1:length(simstimes)) %dopar% {
    pnum <- numeric(length(simstimes))
```

```
            for (j in 1:length(simstimes)) {
                if (i == j) {
                  pnum[j] <- bgrate[i]/(bgrate[i] + dividep[i])
                } else if (i > j) {
                  pnum[j] <- g[i, j]/(bgrate[i] + dividep[i])
                }
            }
            pnum
        }
        p <- matrix(0, length(simstimes), length(simstimes))
        for (i in 1:length(simstimes)) {
            p[i, ] <- ppar[[i]]
        }
        rm(ppar)
        iter <- iter + 1
        print(iter)
        print(trigrate)
    }
    newlist <- list(diag(p), trigtimes, trigdist, trigxdir, trigydir)
    return(newlist)
}
```