point3: Simulating point patterns and more

Philip Dixon

4/6/2020

Simulating point patterns

Poisson processes

A Poisson process is defined by its intensity function (average number per unit area), either constant (HPP) or varying across the region (IPP). The consequence is that a simulated point pattern may have the same expected number, but a different actual number of points. This creates two slightly different implementations of a simulation.

A Poisson process, by definition, has a random number of points. For a HPP, the mean is $\lambda||A||$, where $||A||$ is the area of region A. The actual number for any simulation is $\sim Pois(\lambda)$.

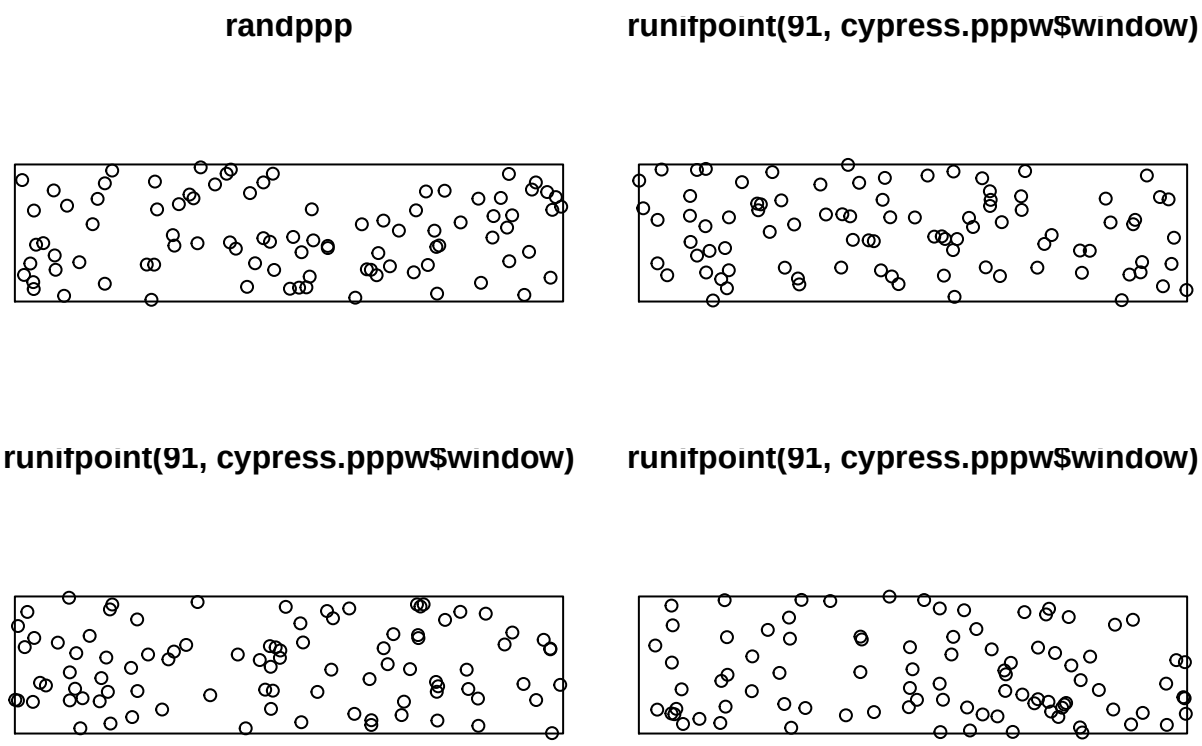
A Binomial process is the name commonly used when the total number of points is fixed. It is common to assume that the data arise from a Poisson process, but fix the number of points in a simulation at the observed number in the data. That's using a Binomial process to generate simulated data.

Simulate a Binomial process within an observation window, 4 realizations with constant number of points.

```
par(mfrow=c(2,2), mar=rep(0.5,4))
randppp <- runifpoint(91, cypress.pppw$window)
plot(randppp)
npoints(randppp)
```

```
## [1] 91
```

```
plot(runifpoint(91, cypress.pppw$window))
plot(runifpoint(91, cypress.pppw$window))
plot(runifpoint(91, cypress.pppw$window))
```



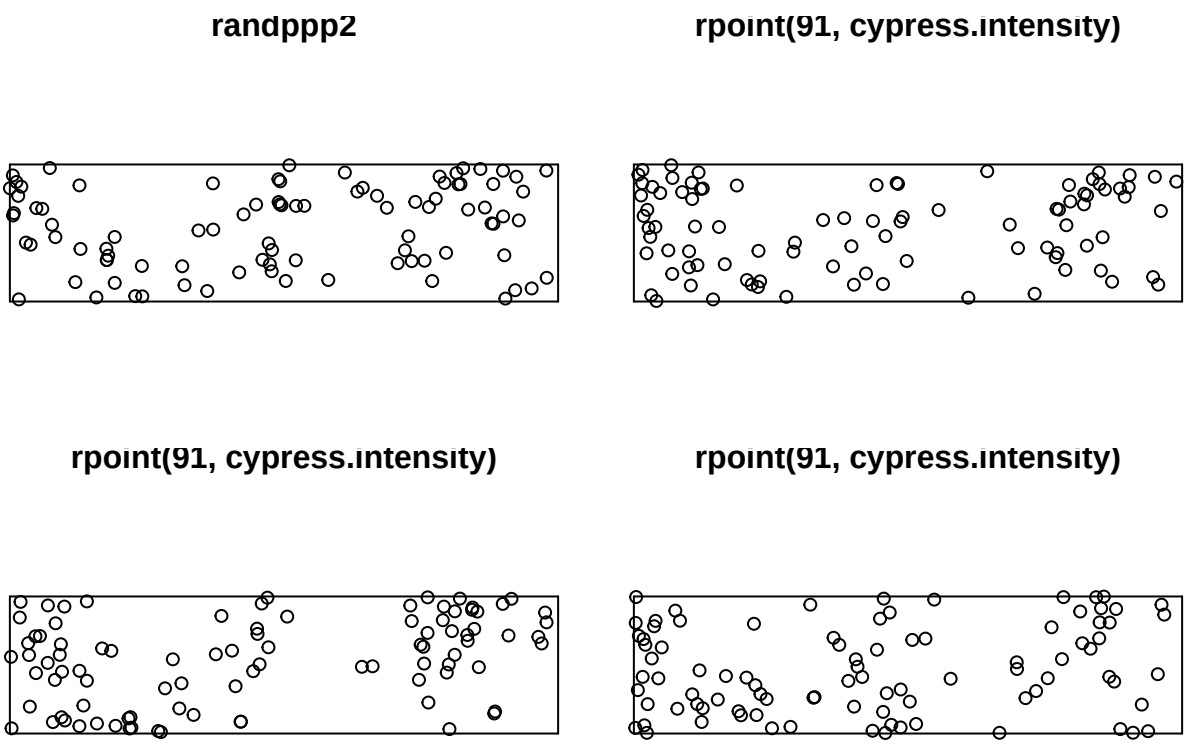
The two arguments are the number of points and the sampling window. The sampling window is specified as an owin() object (just like we used when creating a ppp object). The $window component of a ppp object extracts that sampling window.

Simulate a Binomial process with non-constant intensity, 4 realizations

```
par(mfrow=c(2,2), mar=rep(0.5,4))
cypress.intensity <- density(cypress.pppw, bw.CvL)
randppp2 <- rpoint(91, cypress.intensity)
npoints(randppp2)
```

```
## [1] 91
```

```
plot(randppp2)
plot(rpoint(91, cypress.intensity) )
plot(rpoint(91, cypress.intensity) )
plot(rpoint(91, cypress.intensity) )
```



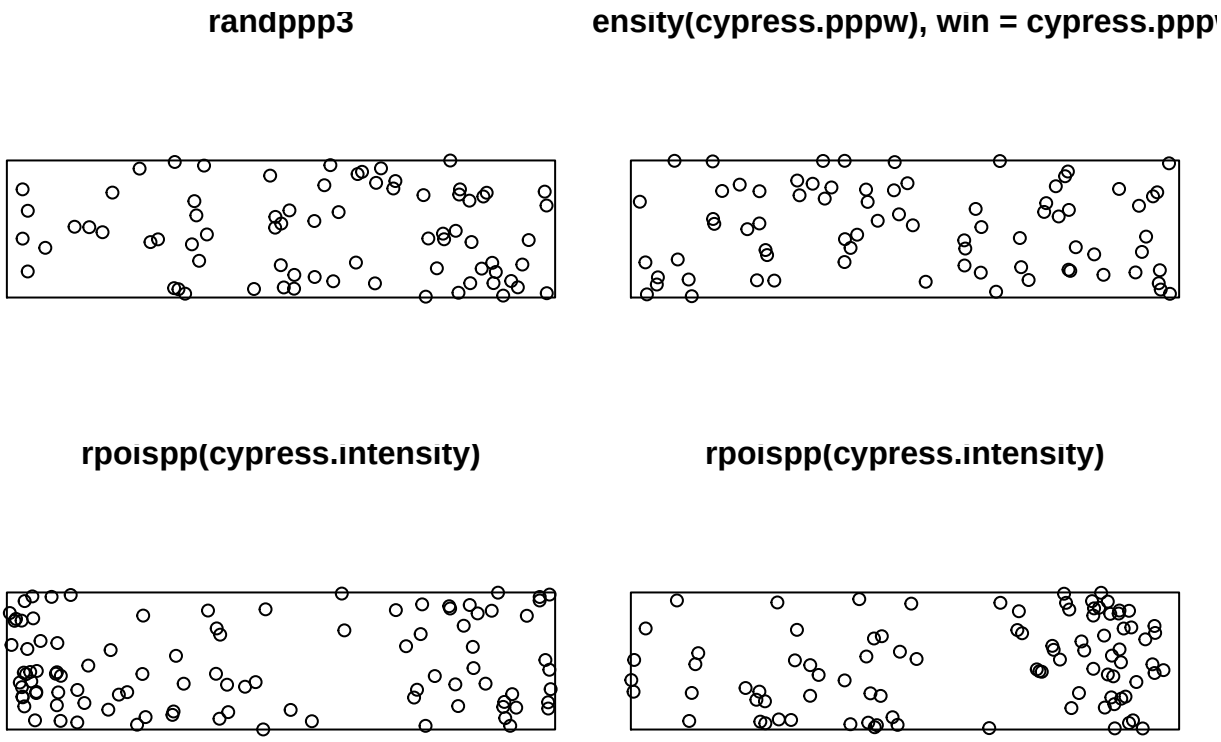
The two arguments to rpoint() are the number of points and the intensity, specified either as a pixel image (used here) or a function of (x,y) coordinates. If you give a pixel image, that contains a sampling window. If you don't, you need to provide a third argument that is the sampling window (as an owin object). I don't know why the plots are shifted.

Sampling Poisson processes with constant intensity or varying intensity. Same ideas above, except that one function does both constant intensity and varying intensity. That is rpoispp(). Here are two simulations of constant intensity and two using the estimated cypress intensity.

```
par(mfrow=c(2,2), mar=rep(0.5,4))
cypress.intensity <- density(cypress.pppw, bw.CvL)
randppp3 <- rpoispp(intensity(cypress.pppw),
  win=cypress.pppw$window)
npoints(randppp3)
```

```
## [1] 74
```

```
plot(randppp3)
plot(rpoispp(intensity(cypress.pppw), win=cypress.pppw$window))
plot(rpoispp(cypress.intensity) )
plot(rpoispp(cypress.intensity) )
```



Remember that the intensity is # points / area. The sampling window is 10000 m^2. If you specify intensity = 91, you get ca 910000 points.

If you specify intensity (first argument as a pixel image), the size of that is taken as the sampling window.

Simulating cluster processes

Matern cluster processes (circular area around daughters)

rMatClust() simulates Matern Cluster processes. It requires 4 arguments. In order, these are:

- intensity (#/unit area) of mothers
- radius of cluster
- mean # pts per cluster (actual # ~ Poisson)
- and the sampling window, as an owin object, created by owin() or extracted from a ppp object

Here are 2 realizations from a Matern process with corresponding L functions and CSR envelopes.

```
par(mfrow=c(2,2), mar=c(3,3,1,0)+0.1, mgp=c(2,0.8,0) )
mc <- rMatClust(2.5, 0.3, 5,win=owin(c(0,2),c(0,2)))
plot(mc, main='Matern Clust process')

mc.env <- envelope(mc, Lest, nsim=199, nrank = 5)
```

```
## Generating 199 simulations of CSR  ...
## 1, 2, 3, 4.6.8.10.12.14.16.18.20.22.24.26.28.30.32.34.36.38.
## 40.42.44.46.48.50.52.54.56.58.60.62.64.66.68.70.72.74.76.78
## .80.82.84.86.88.90.92.94.96.98.100.102.104.106.108.110.112.114.116.
## 118.120.122.124.126.128.130.132.134.136.138.140.142.144.146.148.150.152.154.156
```

```
## .158.160.162.164.166.168.170.172.174.176.178.180.182.184.186.188.190.192.194.
## 196.198 199.
##
## Done.
```

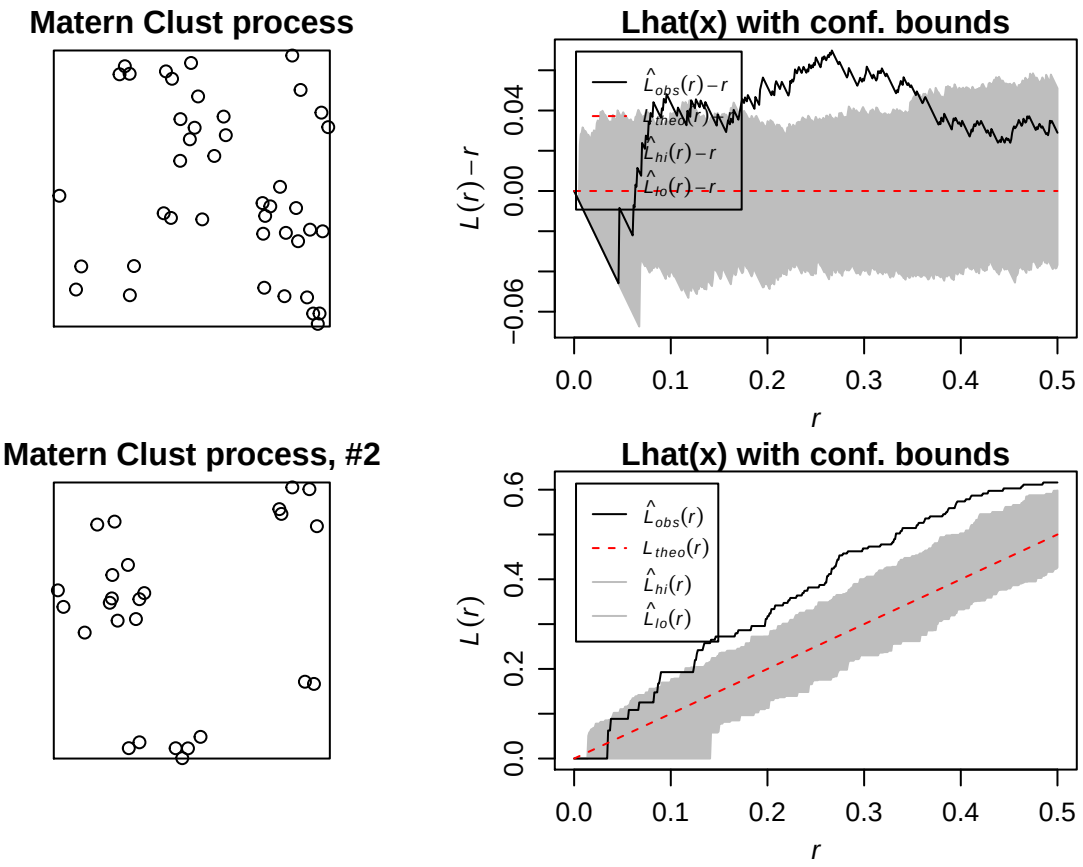
```
plot(mc.env, .-r~r, main='Lhat(x) with conf. bounds', legendargs=list(cex=0.7))

mc <- rMatClust(2.5, 0.3, 5,win=owin(c(0,2),c(0,2)))
plot(mc, .-r~r, main='Matern Clust process, #2')

mc.env <- envelope(mc, Lest, nsim=199, nrank = 5)
```

```
## Generating 199 simulations of CSR  ...
## 1, 2, 3, 4.6.8.10.12.14.16.18.20.22.24.26.28.30.32.34.36.38.
## 40.42.44.46.48.50.52.54.56.58.60.62.64.66.68.70.72.74.76.78
## .80.82.84.86.88.90.92.94.96.98.100.102.104.106.108.110.112.114.116.
## 118.120.122.124.126.128.130.132.134.136.138.140.142.144.146.148.150.152.154.156
## .158.160.162.164.166.168.170.172.174.176.178.180.182.184.186.188.190.192.194.
## 196.198 199.
##
## Done.
```

```
plot(mc.env, main='Lhat(x) with conf. bounds', legendargs=list(cex=0.7))
```



All the "simulate a point pattern" functions accept an optional nsim= argument that specifies the number of simulated patterns. result is then a list of patterns. That is illustrated below for Thomas processes, but the approach applies to all other types of patterns.

Also, all functions for non-Poisson processes simulate the process in a larger area, then trim it to the sampling window. This avoids edge effects. For example, a mom just outside the sampling window can (and is likely to) generate observations inside. The expand= option controls the size of the larger area. My experience is that the default values (different for each type of process) are reasonable.

Thomas processes: daughters ~ bivN around mothers

Same ideas as Matern, except the function is rThomas() and the 2nd argument is the sd around Mom's location.

```
par(mfrow=c(2,2), mar=c(3,3,1,0)+0.1, mgp=c(2,0.8,0) )
tc <- rThomas(2.5, 0.1, 5,win=owin(c(0,2),c(0,2)), nsim=2)
plot(tc[[1]], main='Thomas Clust process')

tc.env <- envelope(tc[[1]], Lest, nsim=199, nrank = 5)
```

```
## Generating 199 simulations of CSR  ...
## 1, 2, 3, 4.6.8.10.12.14.16.18.20.22.24.26.28.30.32.34.36.38.
## 40.42.44.46.48.50.52.54.56.58.60.62.64.66.68.70.72.74.76.78
## .80.82.84.86.88.90.92.94.96.98.100.102.104.106.108.110.112.114.116.
## 118.120.122.124.126.128.130.132.134.136.138.140.142.144.146.148.150.152.154.156
## .158.160.162.164.166.168.170.172.174.176.178.180.182.184.186.188.190.192.194.
## 196.198 199.
##
## Done.
```

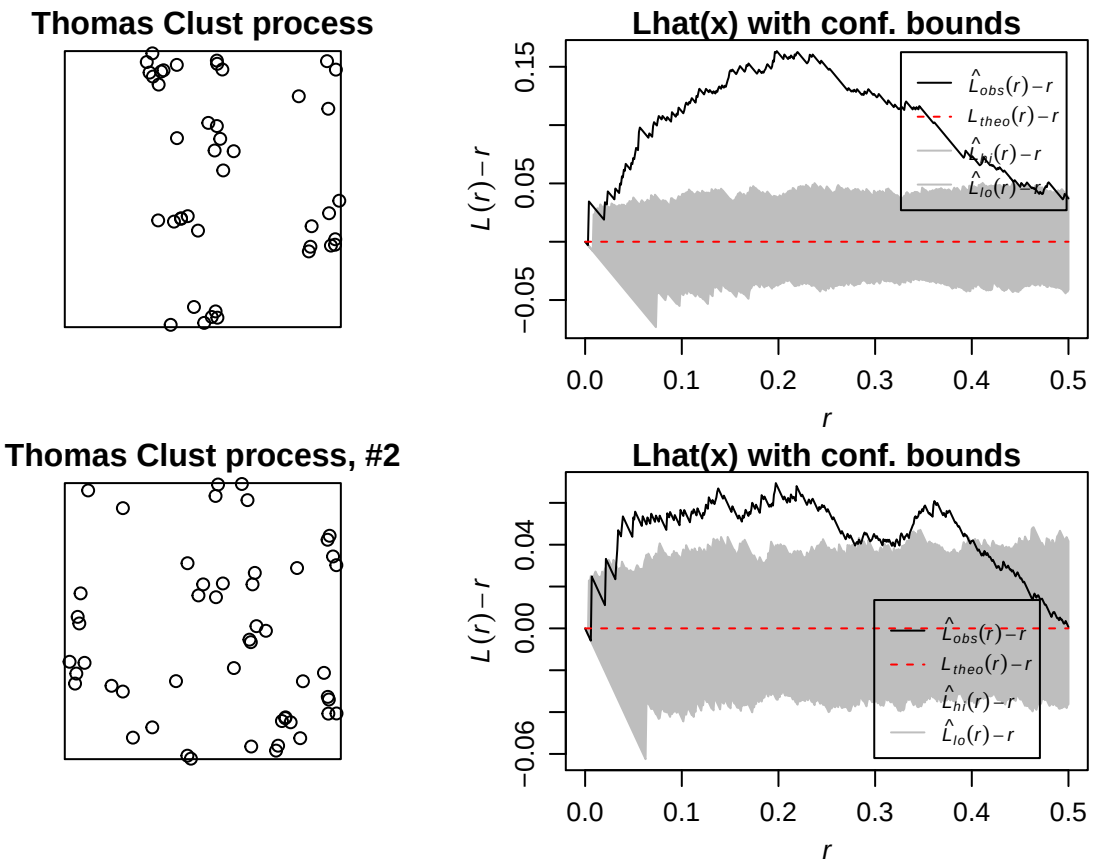
```
plot(tc.env, .-r~r, main='Lhat(x) with conf. bounds', legendargs=list(cex=0.7))

plot(tc[[2]], main='Thomas Clust process, #2')

tc.env <- envelope(tc[[2]], Lest, nsim=199, nrank = 5)
```

```
## Generating 199 simulations of CSR  ...
## 1, 2, 3, 4.6.8.10.12.14.16.18.20.22.24.26.28.30.32.34.36.38.
## 40.42.44.46.48.50.52.54.56.58.60.62.64.66.68.70.72.74.76.78
## .80.82.84.86.88.90.92.94.96.98.100.102.104.106.108.110.112.114.116.
## 118.120.122.124.126.128.130.132.134.136.138.140.142.144.146.148.150.152.154.156
## .158.160.162.164.166.168.170.172.174.176.178.180.182.184.186.188.190.192.194.
## 196.198 199.
##
## Done.
```

```
plot(tc.env, .-r~r, main='Lhat(x) with conf. bounds', legendargs=list(cex=0.7))
```



Note that the sd of a Thomas cluster is smaller than the radius of a Matern cluster, for similar "sized" clusters. 95% of the observations in an isotropic bivariate Normal distribution are within 2.45 sd of the mean.

Simulating processes with inhibition

Spatstat provides methods for many different inhibition processes. I will only illustrate Strauss processes. Others are have similar statistical issues and are specified similarly. Baddeley et al's book describes what's available.

The function is rStrauss(). The arguments are, order:

- intensity of points.
- strength of repulsion
  - 1 = no repulsion, Poisson process
  - 0 = extreme repulsion, Hard core process
- third arg is radius of interaction
- the window for the spatial pattern

If you specify the window as a named argument, note that it is win= for the clustered processes and W= for inihibition processes. No, I don't know why win= for rMatClust changes to W= here.

You need to be careful specifying inhibition processes because it's possible to request something impossible. E.g., 20 points from a hard core process with radius 0.5 in a 2x2 window. Each point requires an area of $\pi 0.5^2 \approx 0.78$. There can be at most 5 of these in a 2x2 area, and that's only when the 5 points are in just the right places.

Here are 2 realizations of 20 points from a hard core process with radius 0.2, in a 2 x 2 window. Next to each is the K function and associated CSR envelope. I illustrate using a loop to examine each pattern one by

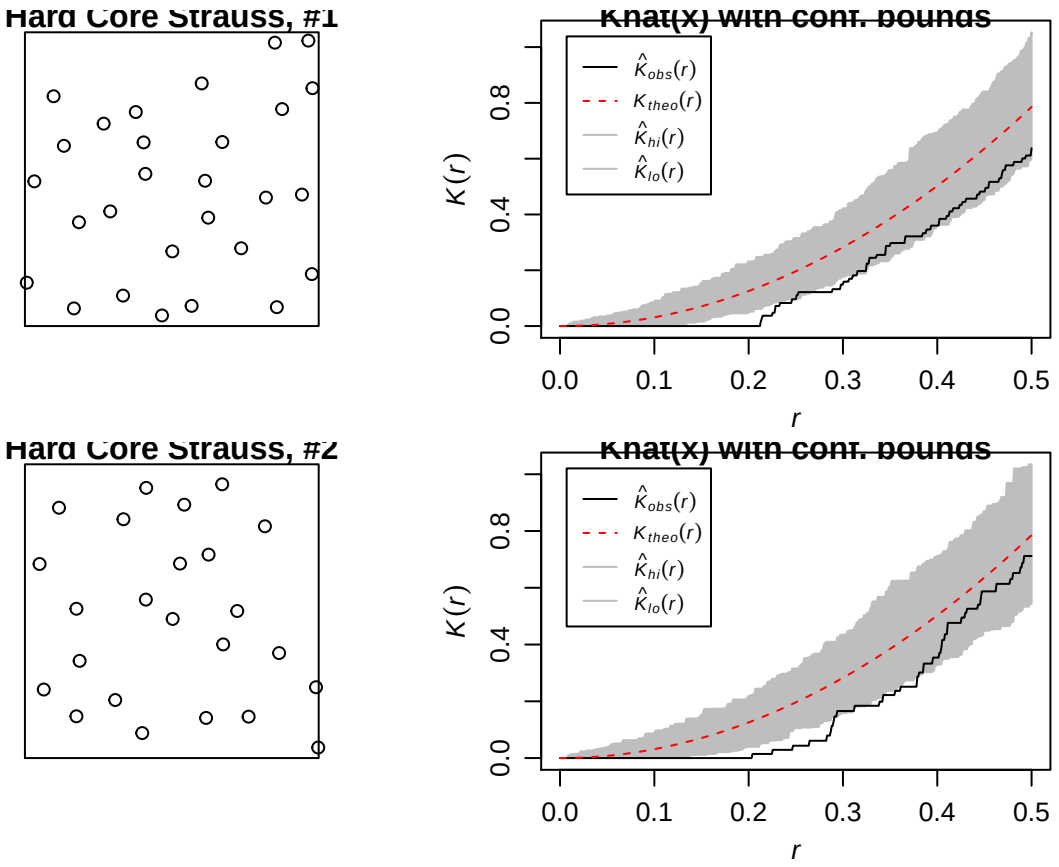
one. length() of a list is the number of components. paste() concatenates strings, so each pattern can be individually numbered in the title.

```
par(mfrow=c(2,2), mar=c(3,3,0,0)+0.3, mgp=c(2,0.8,0))
allstr <- rStrauss(20, 0, 0.2, W=owin(c(0,2),c(0,2)), nsim=2)

npattern <- length(allstr)
for (i in 1:npattern) {
  str <- allstr[[i]]
  plot(str, main=paste('Hard Core Strauss, #', i, sep='') )
  str.env <- envelope(str, Kest, nsim=199, nrank = 5)
  plot(str.env, main='Khat(x) with conf. bounds',
    legendargs=list(cex=0.7))
  }
```

```
## Generating 199 simulations of CSR  ...
## 1, 2, 3, 4.6.8.10.12.14.16.18.20.22.24.26.28.30.32.34.36.38.
## 40.42.44.46.48.50.52.54.56.58.60.62.64.66.68.70.72.74.76.78
## .80.82.84.86.88.90.92.94.96.98.100.102.104.106.108.110.112.114.116.
## 118.120.122.124.126.128.130.132.134.136.138.140.142.144.146.148.150.152.154.156
## .158.160.162.164.166.168.170.172.174.176.178.180.182.184.186.188.190.192.194.
## 196.198 199.
##
## Done.
```

```
## Generating 199 simulations of CSR  ...
## 1, 2, 3, 4.6.8.10.12.14.16.18.20.22.24.26.28.30.32.34.36.38.
## 40.42.44.46.48.50.52.54.56.58.60.62.64.66.68.70.72.74.76.78
## .80.82.84.86.88.90.92.94.96.98.100.102.104.106.108.110.112.114.116.
## 118.120.122.124.126.128.130.132.134.136.138.140.142.144.146.148.150.152.154.156
## .158.160.162.164.166.168.170.172.174.176.178.180.182.184.186.188.190.192.194.
## 196.198 199.
##
## Done.
```



Modeling clustered or inhibition processes

Clustered processes fit by matching K or pcf functions

Spatstat provides 6 functions to estimate parameters of a cluster process. These are:

- matclust.estK(), matclust.estpcf(): Matern cluster processes, by matching K() or g()
- thomas.estK(), thomas.estpcf(): Thomas processes, by matching K() or g()
- lgcp.estK(), lgcp.pcf(): Log Gaussian Cox processes

We haven't discussed log Gaussian Cox processes in lecture. They are a general class of process where the clustering is generated by a random autocorrelated intensity function. The autocorrelation follows a geostatistical variogram model.

All 6 functions have similar arguments. In order:

- either a point pattern or fitted K() or pcf() object
  - better to provide the point pattern so can estimate lambda
- labelled vector of starting values
  - for Matern: kappa = intensity of moms, R = radius
  - for Thomas: kappa = intensity of moms, scale=sd
- optionally:
  - rmin= and rmax=: to specify range over which to match the curve. Default is the entire curve.

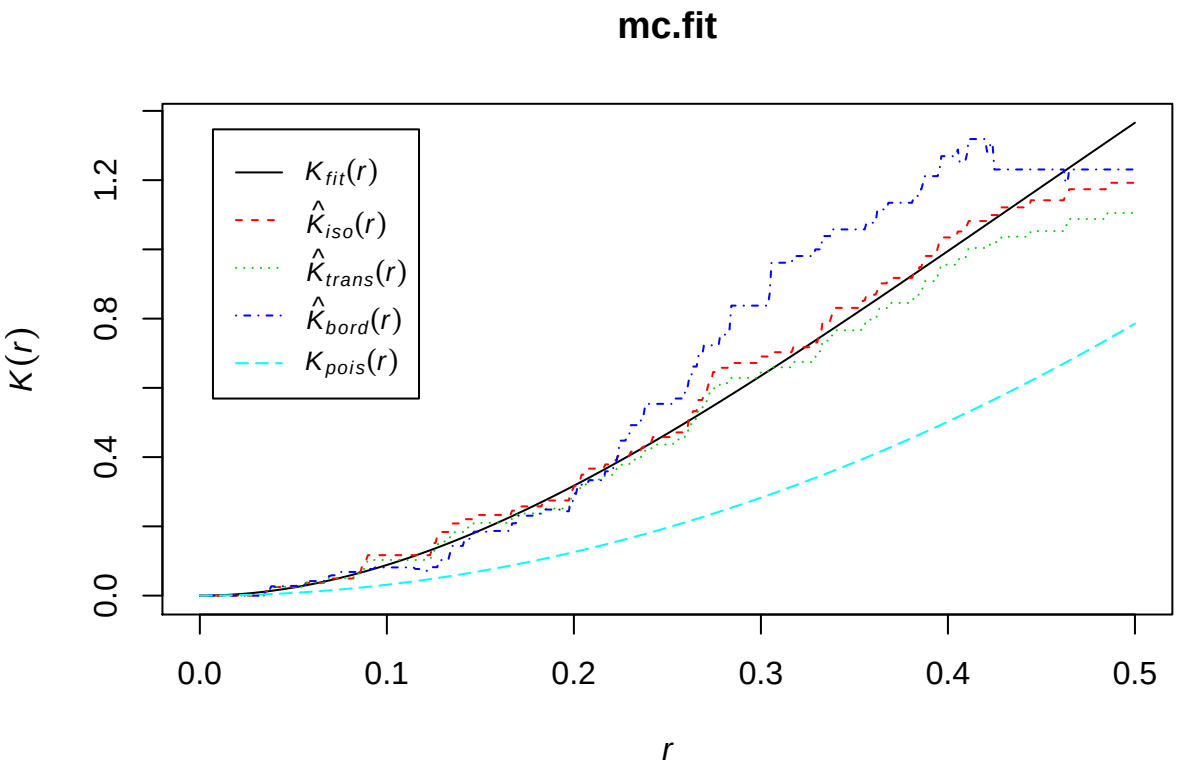
To fit the second simulated Matern, generated with kappa=2.5, R=0.3:

```
mc.fit <- matclust.estK(mc, c(kappa=1.5, R=0.2))
mc.fit
```

```
## Minimum contrast fit (object of class "minconfit")
```

```
## Model: Matern Cluster process
## Fitted by matching theoretical K function to Kest(mc)
##
## Internal parameters fitted by minimum contrast ($par):
##     kappa         R
## 1.6533142 0.3017772
##
## Fitted cluster parameters:
##     kappa     scale
## 1.6533142 0.3017772
##
## Converged successfully after 51 function evaluations
##
## Starting values of parameters:
## kappa     R
##   1.5   0.2
## Domain of integration: [ 0 , 0.5 ]
## Exponents: p= 2, q= 0.25
```

```
plot(mc.fit, legendargs=list(cex=0.8))
```



Printing the result gives you the estimated parameters and information about the fit. For Matern fits, the internal and fitted parameters are the same. For Thomas fits, the internal parameters are kappa and σ^2 (variance) while the fitted parameters are kappar and σ (sd=scale).

Plotting the object shows the empirical K function from various edge corrections and the modeled K function.

To see the uncertainty in the fit, use a parametric bootstrap: repeatedly simulate data from the fitted model,

estimate the multiple K functions, and draw an envelope.

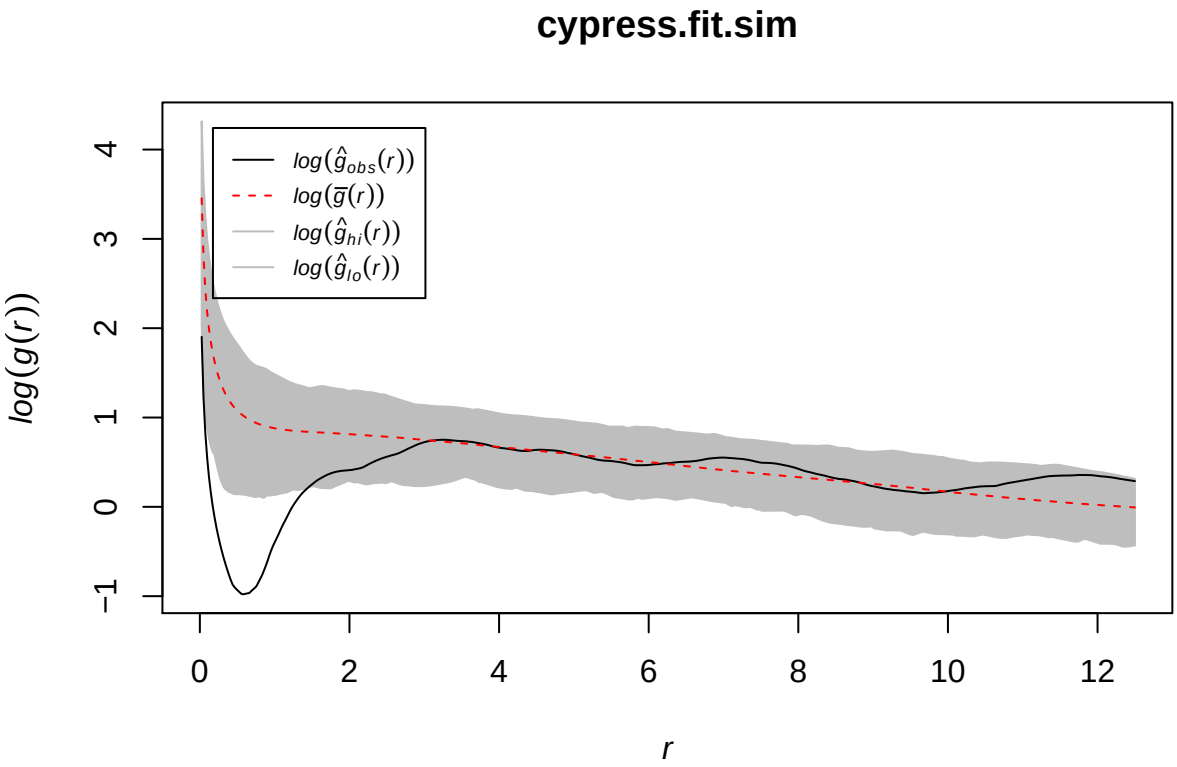
Details: you need specify how to simulate data. This is an expression() including the function name, the appropriate parameter values, and the sampling window. The fitted model parameters are stored in $modelpar. This vector includes mu, the mean number of daughters. One difficult is that the parameters must be given as scalars.

Using the cypress data, fit using a pair correlation function fit to distances between 3 and 10. Minimum distance chosen to avoid the inhibition at short distances.

```
cypress.fit<- matclust.estpcf(cypress.pppw,
  c(kappa=10/10000, scale=5), rmin=3, rmax=10)
kappa <- cypress.fit$modelpar[1]
R <- cypress.fit$modelpar[2]
mu <- cypress.fit$modelpar[3]
cypress.fit.sim <- envelope(cypress.pppw, pcf,
  nsim=199,nrank=5,
  simulate=expression(
    rMatClust(kappa, R, mu,
    win=window(cypress.pppw) )) )
```

```
## Generating 199 simulations by evaluating expression  ...
## 1, 2, 3, 4.6.8.10.12.14.16.18.20.22.24.26.28.30.32.34.36.38.
## 40.42.44.46.48.50.52.54.56.58.60.62.64.66.68.70.72.74.76.78
## .80.82.84.86.88.90.92.94.96.98.100.102.104.106.108.110.112.114.116.
## 118.120.122.124.126.128.130.132.134.136.138.140.142.144.146.148.150.152.154.156
## .158.160.162.164.166.168.170.172.174.176.178.180.182.184.186.188.190.192.194.
## 196.198 199.
##
## Done.
```

```
plot(cypress.fit.sim, log(.)~r, legendargs=list(cex=0.7))
```



The good fit between 3 and 10 m is apparent. The lack of fit between 0 and 3 m is because of the inhibition that was ignored when fitting the curve.

Since the model was fit using pcf(), evaluating lack of fit using pcf() is inappropriate. Same reason why leave-one-out or some other cross validation provides a better evaluation of model assumptions. For a point pattern fit using K() or g(), the obvious (and recommended) choices are G(x) and F(x), the nearest neighbor and nearest event distance distributions. Just replace pcf in the envelope() function with Gest() or Fest(). Not done for the cypresses, because we know there will be lack of fit at short distances (the inhibition that we've ignored).

There is also a kppm() function with very general functionality, including different criteria to measure fit. You can specify a maximum distance but not a minimum distance. Not illustrated for the cypress data because of the inhibition at short distances.

Inhibition processes fit by pseudo-likelihood

The K() function for inhibition processes is often hard to write down. Hence, inhibitation models are usually fit using pseudo-likelihood. The radius of inhibition is an irregular parameter, so it is estimated first using profile likelihood. The other parameters are then estimated by numerical maximization of the likelihood conditional on the radius.

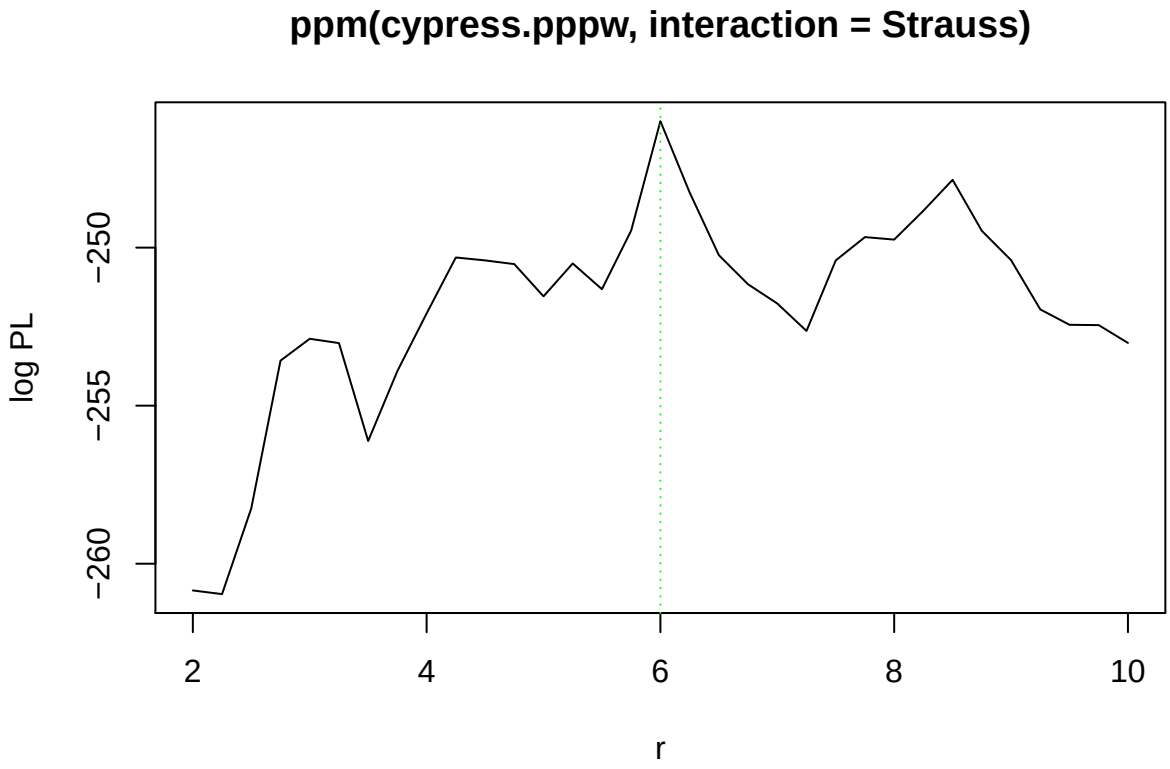
profilepl() calculates the log pseudolikelihood for a range of values of r, the radius of inhibition. plot() draws the profile trace and identifies the location of the maximum.

```
temp <- profilepl(data.frame(r=seq(2, 10, 0.25)),
  Strauss, cypress.pppw)
```

```
## (computing rbord)
```

```
## comparing 33 models...
## 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28
## fitting optimal model...
## done.
```

```
plot(temp)
```



This plot illustrates why standard optimization methods don't work for r. There are multiple local optima (e.g. ca 3, 4.5, 6 and 8.5). 6 is the mle for r. If the exact value mattered, rerun profilepl() with a sequence of values between 5.5 and 6.5.

ppm() then fits a point pattern model using maximum likelihood or pseudolikelihood. The three arguments to fit an inhibition model are, in order: * the point pattern * the trend model, ~1 for constant intensity + could be something else, just as with IPPs * interact= specifies which interaction model + the interaction radius needs to be specified inside the model name

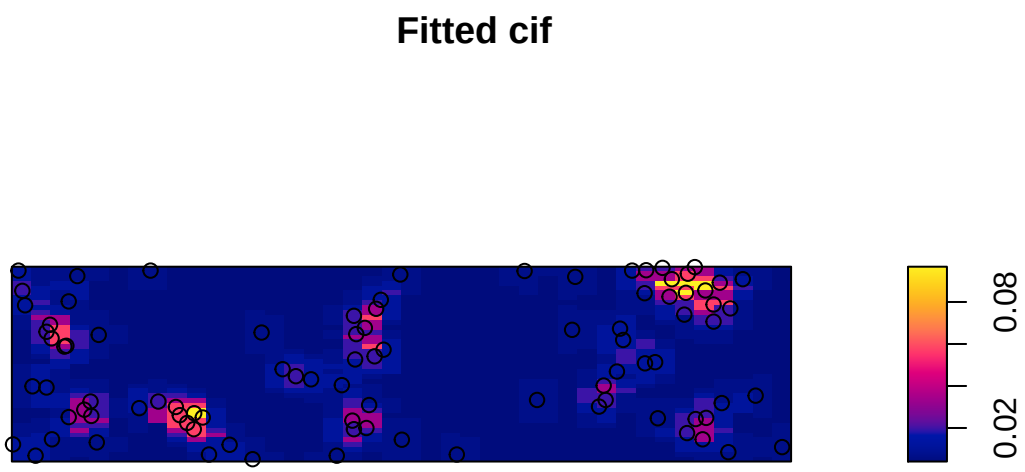
The help file for ppm describes 28 interaction models.

```
cypress.strauss <- ppm(cypress.pppw, ~1,
  interact=Strauss(6))
cypress.strauss
```

```
## Stationary Strauss process
##
## First order term:  beta = 0.004022312
##
## Interaction distance:    6
```

```
## Fitted interaction parameter gamma:   1.6991652
##
## Relevant coefficients:
## Interaction
##    0.530137
##
## For standard errors, type coef(summary(x))
##
## *** Model is not valid ***
## *** Interaction parameters are outside valid range ***
```

```
plot(cypress.strauss)
```



Printing the result gives you parameter estimates and information about the fit. Here, we see the note that the model doesn't fit. The interaction parameter should be between 0 and 1 and the mle is 1.7, outside the valid range. That's because the data have both inhibitation and clustering and the likelihood function can't separate the two processes.

Plotting the result gives one or more image plots of the model fit. The conditional intensity plot is the predicted intensity given locations of points in the data.