point2: estimating and modeling intensity

Philip Dixon

Revised 4/5/2020

Estimating intensity

All this code uses the live cypress subset of the swamp data set

```
# in case data not already loaded
library(spatstat)

swamp <- read.csv('swamp.csv', as.is=T)
cypress <- subset(swamp, live == 1 & sp=='TD')
# keep live cypress (Taxodium distichum) locations

cypress.ppp <- as.ppp(cypress[,c('x','y')], W=c(0,50, 0, 200))
cypress.pppw <- as.ppp(cypress[,c('y','x')], W=c(0,200, 0, 50))
```

global: = # pts / area

```
npoints(cypress.ppp) / area(cypress.ppp)
```

```
## [1] 0.0091
```

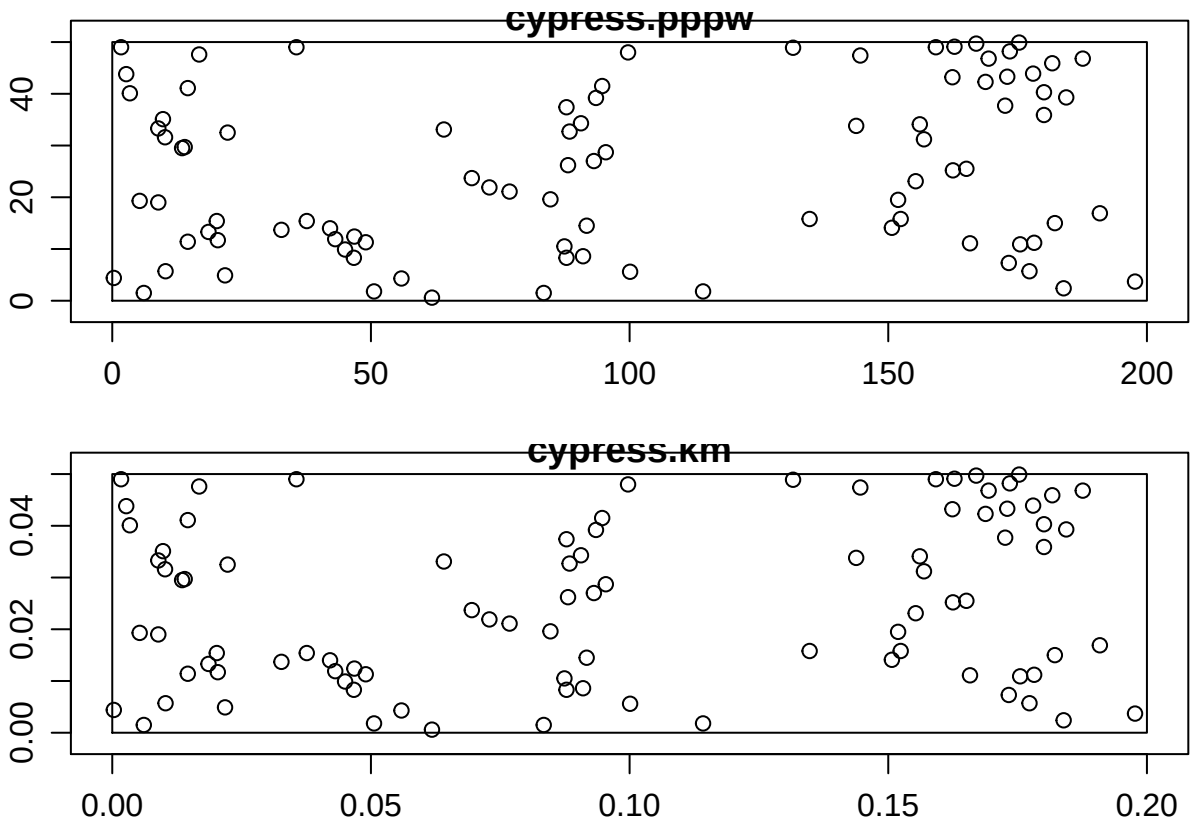
```
intensity(cypress.ppp)
```

```
## [1] 0.0091
```

npoints() returns the number of points in the point pattern. area() returns the area of the sampling window. intensity() returns the intensity

Sometimes the unit of distance, and hence the unit of area, is inconveniently small or large. These can be rescaled using the rescale() function, which changes both the location coordinates and the window dimensions.

```
cypress.km <- rescale(cypress.pppw, 1000, 'km')
par(mfrow=c(2,1), mar=c(3,3,0,0)+0.2, mgp=c(2,0.8,0))
plot(cypress.pppw, axes=T)
plot(cypress.km, axes=T)
```




```
c(m2=intensity(cypress.pppw), km2=intensity(cypress.km))
```

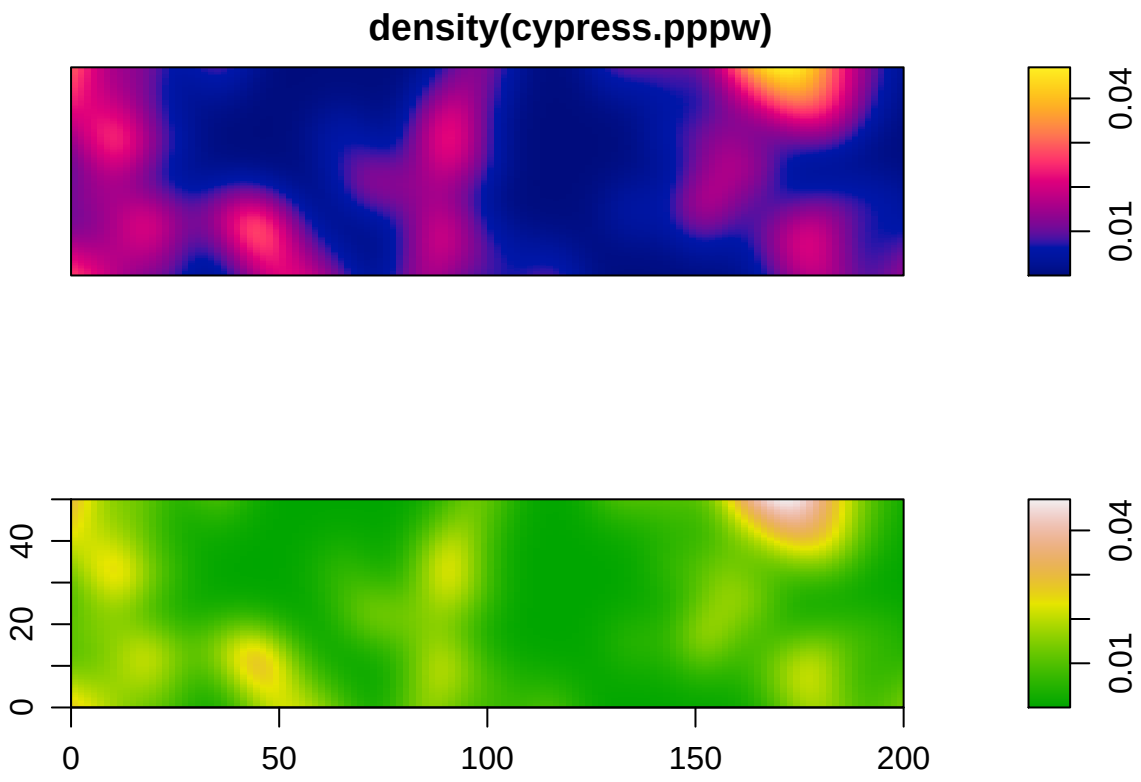
```
##      m2     km2
## 9.1e-03 9.1e+03
```

The first argument to rescale() is the point pattern (ppp) object, the second is the conversion factor (as number of old units per 1 new unit), the third is an optional name for the new unit. So the above code converts distance from m to km and area from m^2 to km^2.

local: allows intensity to vary across the sampling window

density() computes the kernel estimate of local intensity. Actually, this is density.ppp() for a point pattern, but R figures that out for you. There is also density() in base R that is the kernal smoothed alternative to a histogram. The output of density.ppp() is a spatstat pixel image. These are the im class. This can be plotted; the result looks just like a gridded sp image.

```
par(mfrow=c(2,1), mar=c(3,3,0,0)+0.7, mgp=c(2,0.8,0))
plot(density(cypress.pppw))
plot(density(cypress.pppw), col=terrain.colors(256), axes=T,
  main='')
```



The plot function is plot.im(), so help is available by ?plot.im. If you want to change the colour palette, you can do this by adding col=palette(256), because plot.in() uses 256 colors. You can also use any of the other base graphics options.

The most difficult part of kernel smoothing is choosing the kernel bandwidth. density.ppp() calls this parameter sigma. The default value is probably not very reasonable. The help file describes it as "a default value of sigma calculated by a simple rule of thumb that depends only on the size of the window."

spatstat provides four different methods to compute a more appropriate bandwidth. The most commonly used method is Diggle's, but my experience is that usually undersmooths. The estimated sigma is too small. I usually find the point process likelihood (ppl) sigma produces pictures that I like better. The Cronie and van Lieshout method is very recent (2018 paper). I have almost no experience with it. For the cypress data, the estimated bandwidth is identical to that from the ppl method.

All bandwidth methods have plot and print functions. print prints the optimal choice. plot plots the criterion across a range of values so you can see whether the choice is clear or vague. I don't illustrate bw.scott(), a criterion based on sample size

```
bw.diggle(cypress.pppw, fig.height=6)
```

```
##    sigma
## 2.996575
```

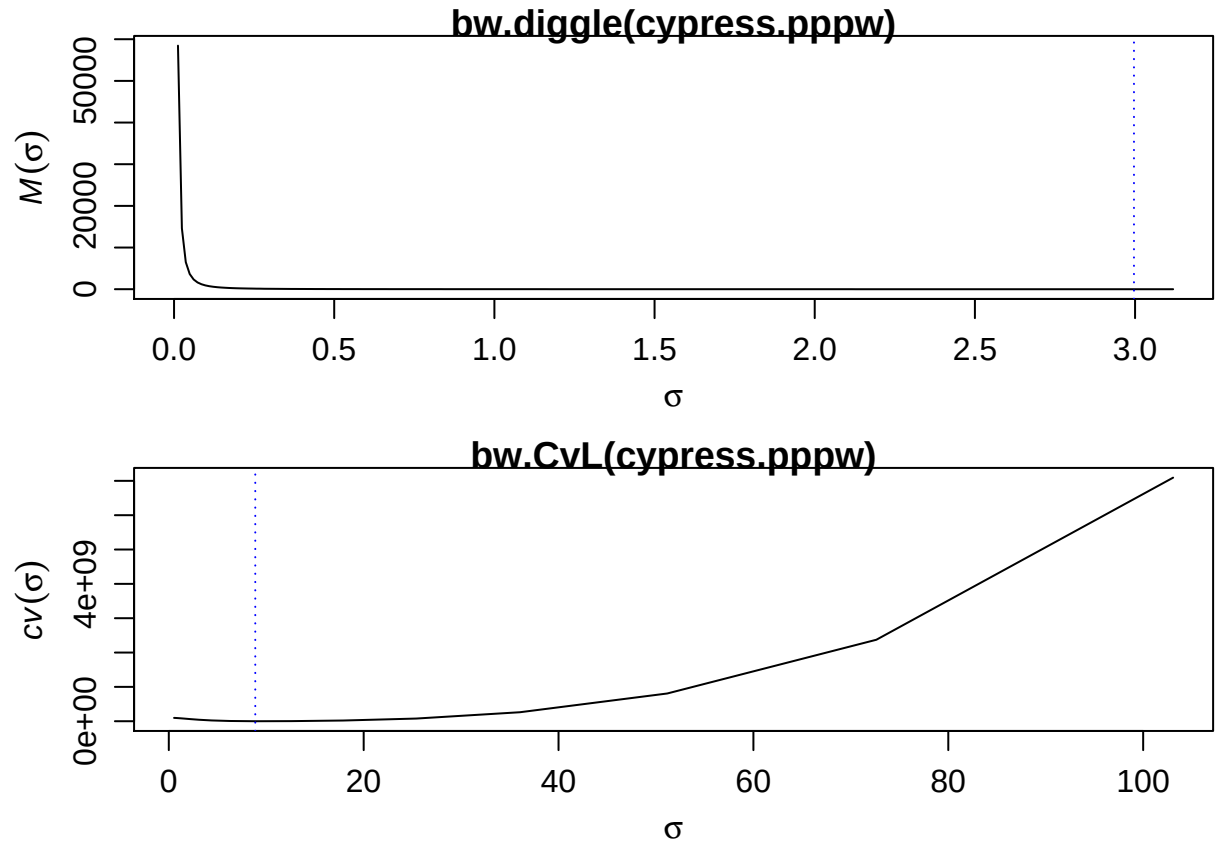
```
bw.ppl(cypress.pppw)
```

```
##    sigma
## 8.876612
```

```
bw.CvL(cypress.pppw)
```

```
##    sigma
## 8.876612
```

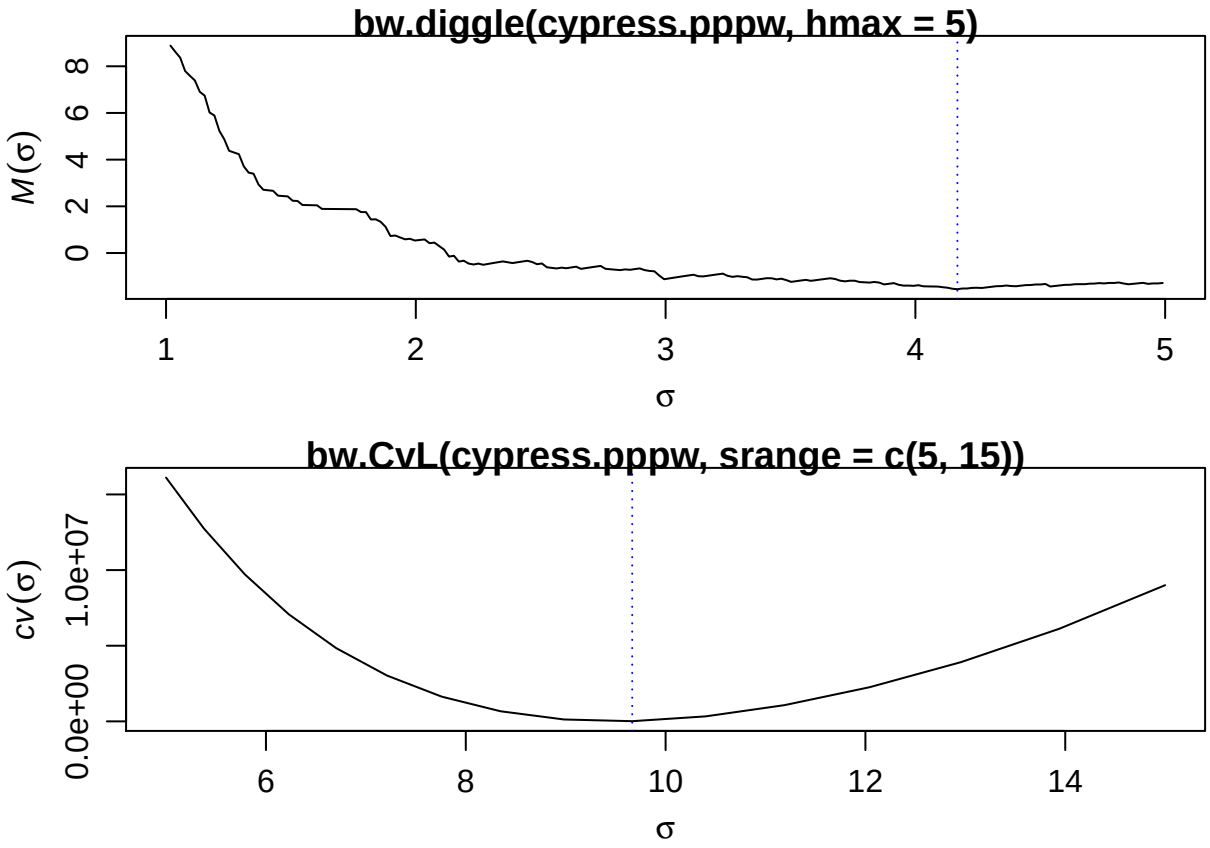
```
par(mfrow=c(2,1), mar=c(3,3,0,0)+0.7, mgp=c(2,0.8,0))
plot(bw.diggle(cypress.pppw))
plot(bw.CvL(cypress.pppw))
```




```
par(mfrow=c(1,1))
```

The smoothing plots usually require some touch-up to be useful. For example, the Diggle plot is dominated by the spike at very small backwidths. You really care about the curve around the optimal value. That appears to be 3.0 for the cypress data, although that is suspect because it's close to the upper bound under consideration. Here are two ways to focus on interesting parts of the curve.

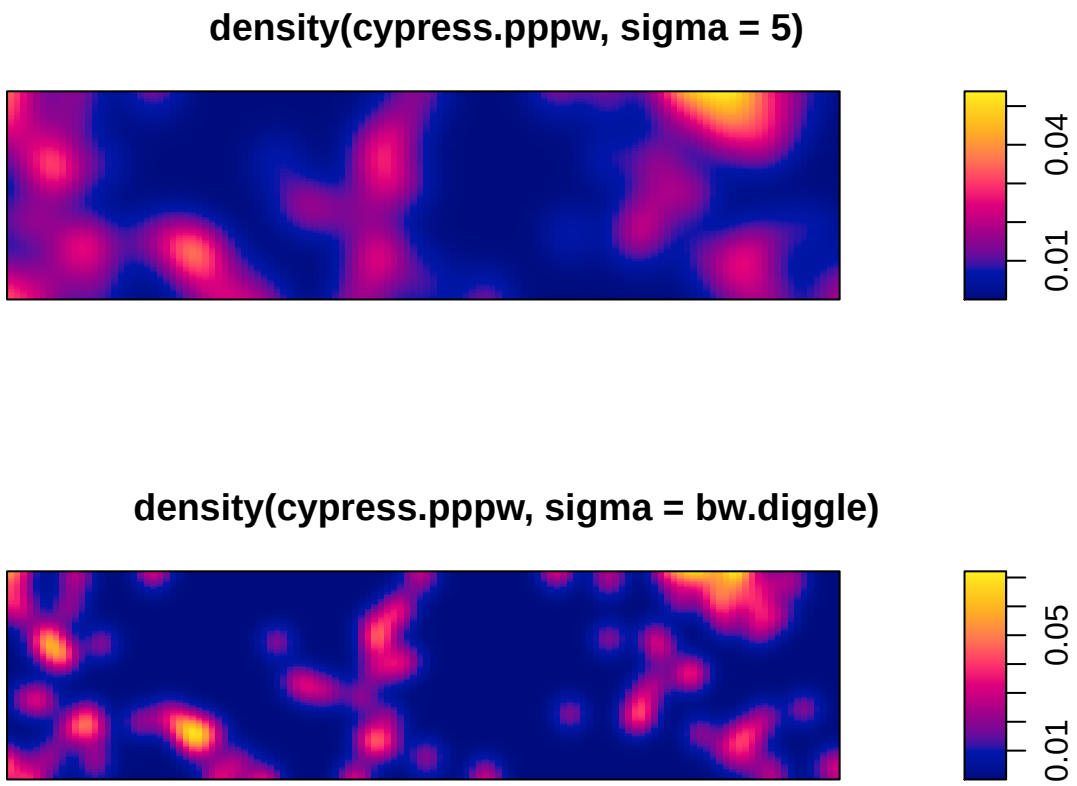
```
# change the upper limit (only used in bw.diggle)
#   and only plot from 1 to 5
par(mfrow=c(2,1), mar=c(3,3,0,0)+0.7, mgp=c(2,0.8,0))
plot(bw.diggle(cypress.pppw, hmax=5), xlim=c(1,5)  )

# change the range of sigma (used in bw.CvL and bw.ppl)
plot(bw.CvL(cypress.pppw, srange=c(5,15)) )
```



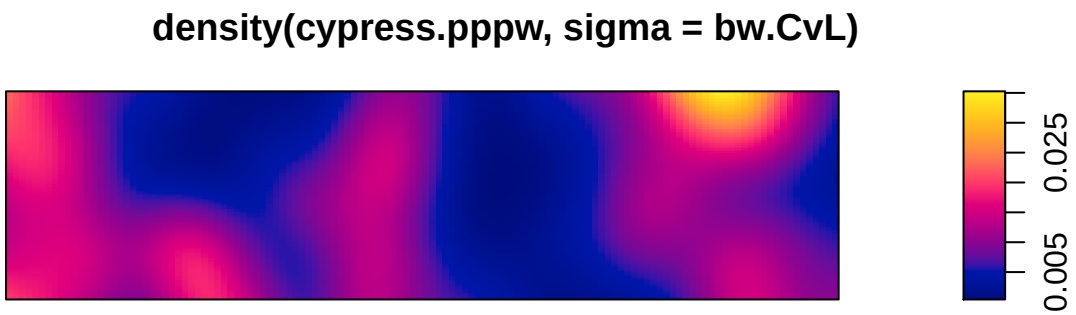

```
par(mfrow=c(1,1))
```

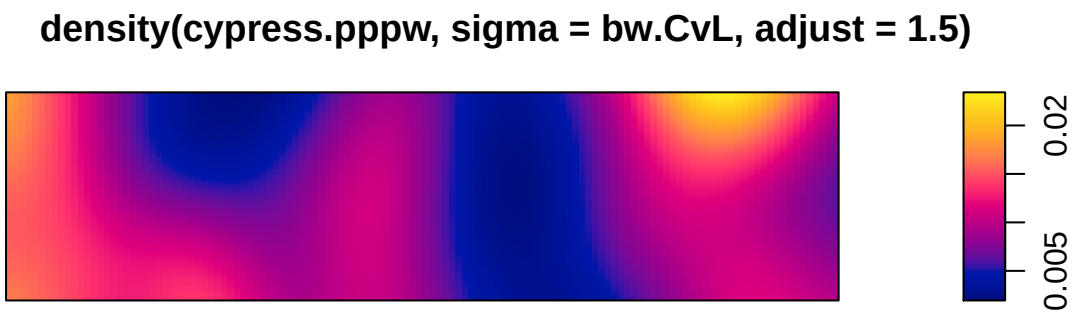
You can tell density what sigma to use, or tell it to use the result of one of bandwidth functions. A third option, which I use quite a bit, is adjust=. This is a multiplicative adjustment to the smoothing parameter. When you specify sigma=bw.diggle, adjust=1.5, the smoothing parameter estimated by bw.diggle() is multiplied by 1.5.

```
par(mfrow=c(2,1), mar=c(3,3,0,0)+0.7, mgp=c(2,0.8,0))
plot(density(cypress.pppw, sigma=5))
plot(density(cypress.pppw, sigma=bw.diggle))
```




```
plot(density(cypress.pppw, sigma=bw.CvL))
plot(density(cypress.pppw, sigma=bw.CvL, adjust=1.5))
```






```
par(mfrow=c(1,1))
```

My experience is that Diggle's method, bw.diggle, undersmooths the data. My personal preference is bw.ppl, although bw.CvL looks like a faster way to get to what seems to be the same place.

By default, smoothing is isotropic (same amount of smoothing in all directions). You can do anisotropic smoothing by specifying a bivariate variance-covariance matrix to smooth different amounts in different directions.

Modeling intensity (inhomogeneous point processes)

The ppm() function will fit Poisson or inhibition processes. These are fit by likelihood. If you want to model a cluster process, use kppm() instead. That fits models using K functions.

Spatstat recently changed the syntax of the ppm() function. This file describes the new syntax. Bivand and other references describe the old syntax. For now, both are accepted. The new syntax is much closer to the standard R modeling syntax.

The model is specified by a formula: pattern ~ trend. Pattern is the name of the ppp object with the locations. Trend specifies the variables in the intensity model. E.g. cypress.ppp ~ x + y will fit a spatial linear trend to the log intensity of live cypress trees.

Covariates can be specified in many ways: as coordinates (x,y), as a pixel image giving covariate values on a fine grid of points, as a function of x and y coordinates. Others are described in the ppm helpfile. Note that covariates need to be available across the study window. It isn't sufficient to only have them at observed locations (remember the integral over the entire area in the point process likelihood).

interaction= specifies the point process model. If omitted, ppm() fits a Poisson process. You can also specify one of many different interaction models that describe inhibition processes.

correction= specifies the edge correction. The default is border, which omits locations close to the edge. I prefer Ripley (also called isotropic), but that's based on experience with K functions. I don't know of a comparative study of edge corrections.

The output is a table of estimates, standard errors, a confidence interval, and Z test. Plotting a fitted model shows the locations with fitted trend, then a plot of locations and estimated se. The pause=F option turns off pausing between plots. I suggest pausing when working interactively.

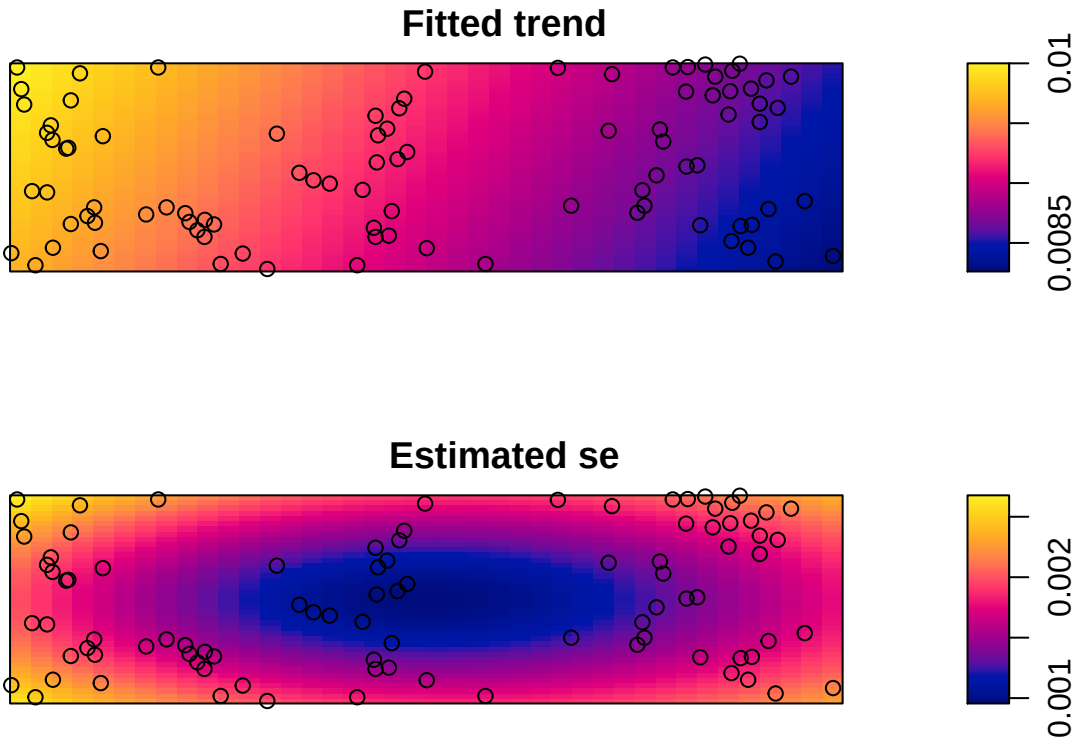
```
cypress.ipp <- ppm(cypress.pppw ~ x + y,
  correction='Ripley')
cypress.ipp
```

```
## Nonstationary Poisson process
##
## Log intensity:  ~x + y
##
## Fitted trend coefficients:
##   (Intercept)             x             y
## -4.6358905680 -0.0008133415  0.0006634978
##
##                  Estimate        S.E.      CI95.lo      CI95.hi Ztest
## (Intercept) -4.6358905680 0.274971311 -5.174824434 -4.096956702   ***
## x           -0.0008133415 0.001817664 -0.004375898  0.002749215
## y            0.0006634978 0.007265246 -0.013576123  0.014903119
##                     Zval
## (Intercept) -16.85954273
## x            -0.44746526
## y             0.09132489
```

```
c(lnl=logLik(cypress.ipp), AIC=AIC(cypress.ipp) )
```

```
##       lnl       AIC
## -518.5484 1043.0968
```

```
par(mfrow=c(2,1), mar=c(3,3,0,0)+0.7, mgp=c(2,0.8,0))
plot(cypress.ipp, pause=F)
```




```
par(mfrow=c(1,1))
```

Z tests are one parameter at a time. Can also do model comparison using a likelihood ratio test (the equivalent of anova). To test whether any spatial trend, i.e. both slope for x = 0 and slope for y = 0, compare the IPP (x+y) to a homogeneous Poisson process (intercept only, i.e. 1).

The anova() function does model comparison tests. By default (for models other than linear normal ones), it does not calculate a p-value. That's because there is no one-size-fits-all reference distribution. You can request a p-value by adding test= 'something'. test='Chi', requests a Chi-square test. This is theoretically well justified (at least asymptotically) for Poisson processes. That theory doesn't exist for pseudolikelihood (what is used to fit interaction processes). I'll discuss fitting clustered and interaction processes in a week or so.

```
cypress.hpp <- ppm(cypress.pppw ~ 1, correction='Ripley')
anova(cypress.hpp, cypress.ipp, test='Chi')
```

```
## Analysis of Deviance Table
##
## Model 1: ~1   Poisson
## Model 2: ~x + y   Poisson
##   Npar Df Deviance Pr(>Chi)
## 1    1
## 2    3  2  0.20869   0.9009
```

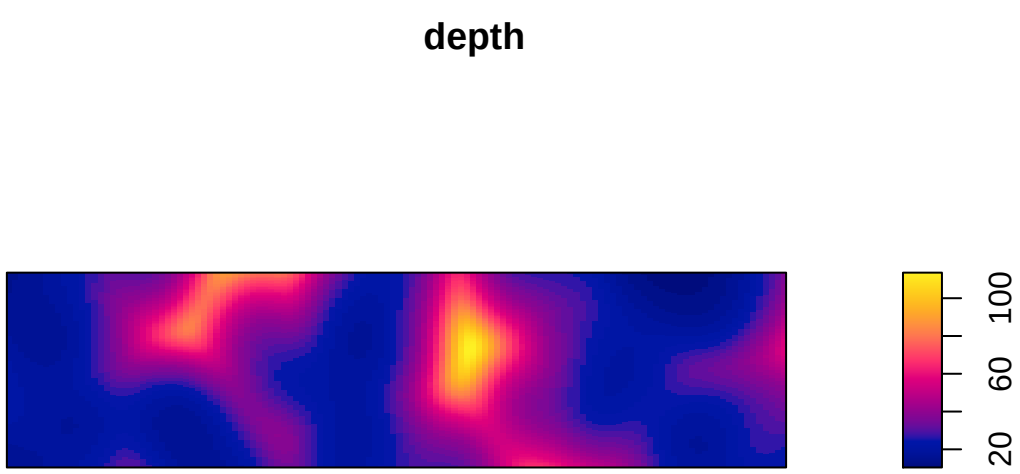
This gives you the likelihood ratio test that compares the HPP to the IPP using x and y as covariates.

using covariates other than coordinates

x and y are especially convenient covariates because they are easily computed anywhere in the sampling window. Other covariates can be specified in various ways. The goal is to enable computing the covariate value for an arbitrary (x,y) location. The two most useful ways are to write a function and to provide a pixel image. The function inputs (x,y) and returns the covariate value. This is useful if the covariate is distance from some pre-specified location, e.g. a hazardous waste site. The pixel image provides the covariate value for each point on a fine grid.

Here's how to use an image (spatstat im object) to specify the covariate. depth.Rdata contains depth, a pixel image, with an approximation of the water depth throughout the plot. This is a "horizontal" image, so the coordinates match the cypress.pppw point pattern.

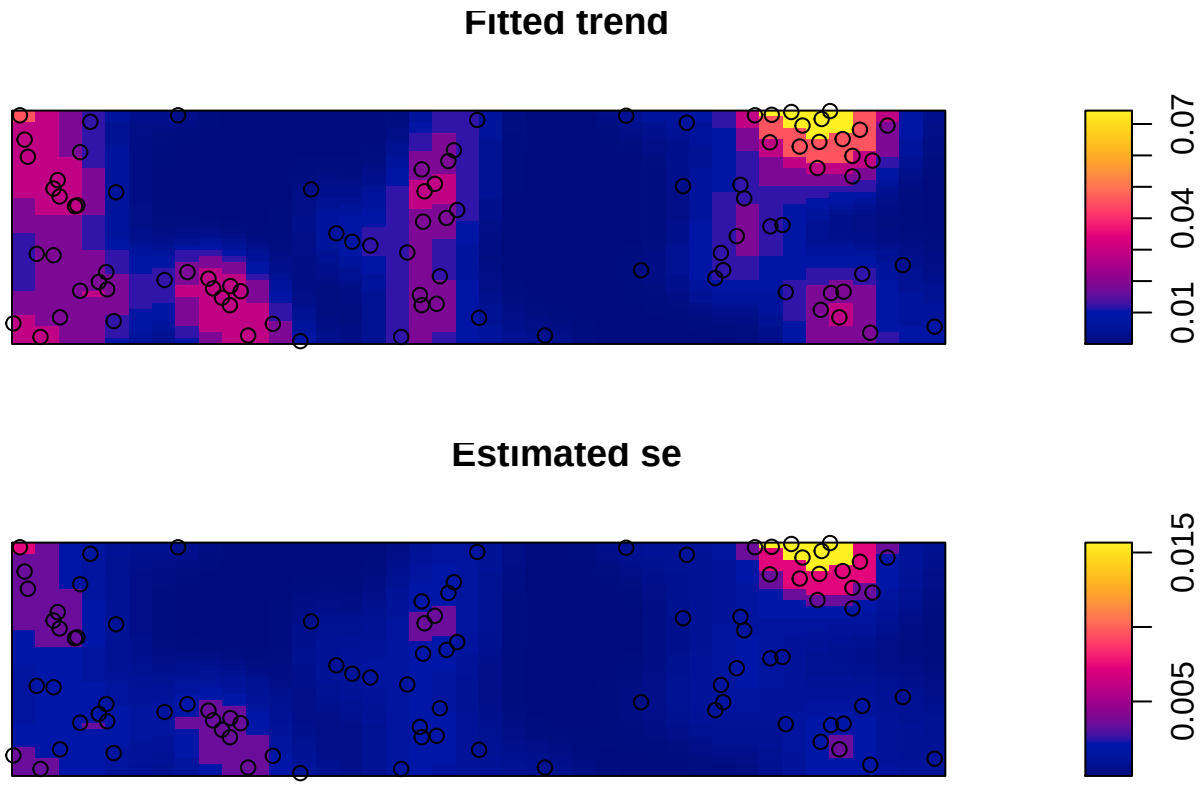
```
load('depth.Rdata')
plot(depth)
```



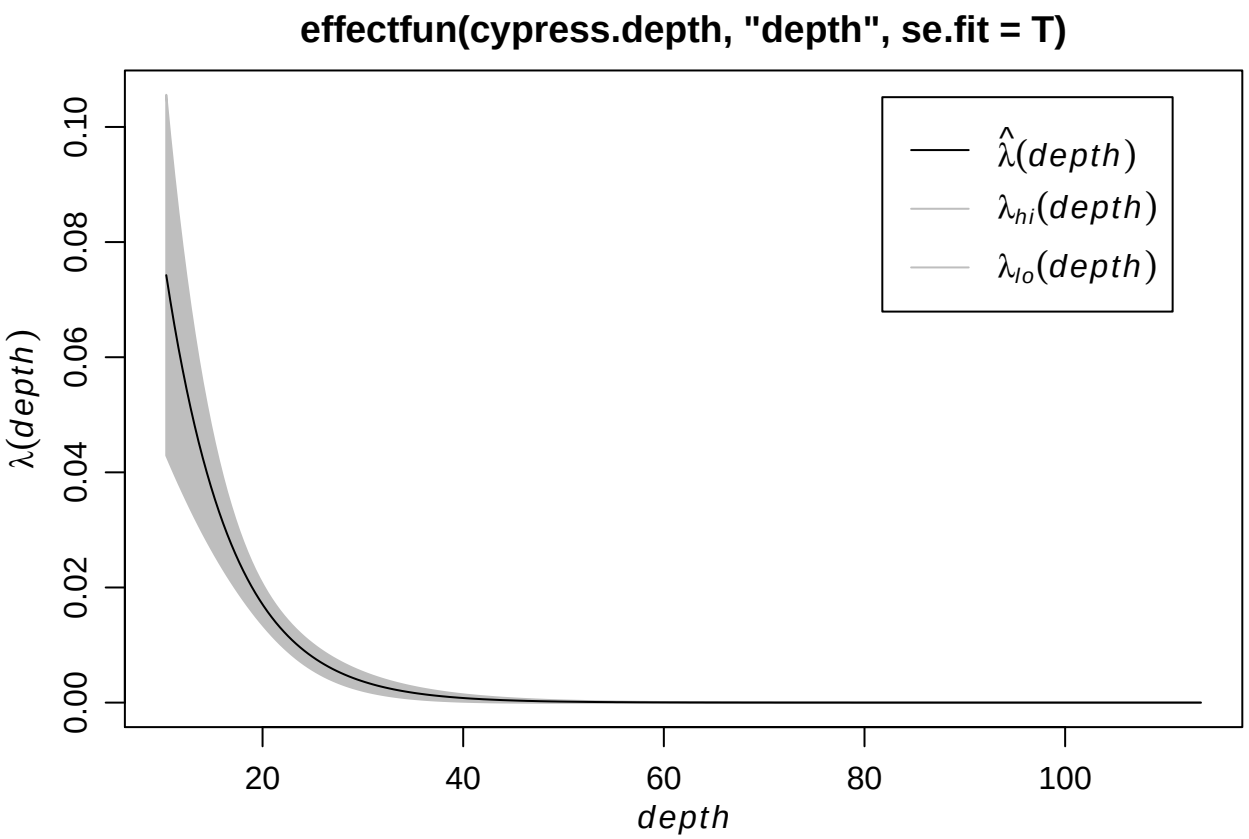

```
cypress.depth <- ppm(cypress.pppw ~ depth)
cypress.depth
```

```
## Nonstationary Poisson process
##
## Log intensity:  ~depth
##
## Fitted trend coefficients:
## (Intercept)       depth
##  -1.0057500  -0.1532764
##
##               Estimate       S.E.    CI95.lo    CI95.hi Ztest      Zval
## (Intercept) -1.0057500 0.39791267 -1.7856445 -0.2258555     * -2.527565
## depth       -0.1532764 0.01930366 -0.1911108 -0.1154419   *** -7.940273
```

```
par(mfrow=c(2,1), mar=rep(0.5,4),
  mgp=c(2,0.8,0))
plot(cypress.depth, pause=F)
```




```
par(mfrow=c(1,1), mar=c(3,3,2,0)+0.2)
plot(effectfun(cypress.depth, 'depth',
  se.fit=T))
```



The covariate image can be used just like a coordinate or function. Just specify the name of the image. By default, it will be looked for in your working directory.

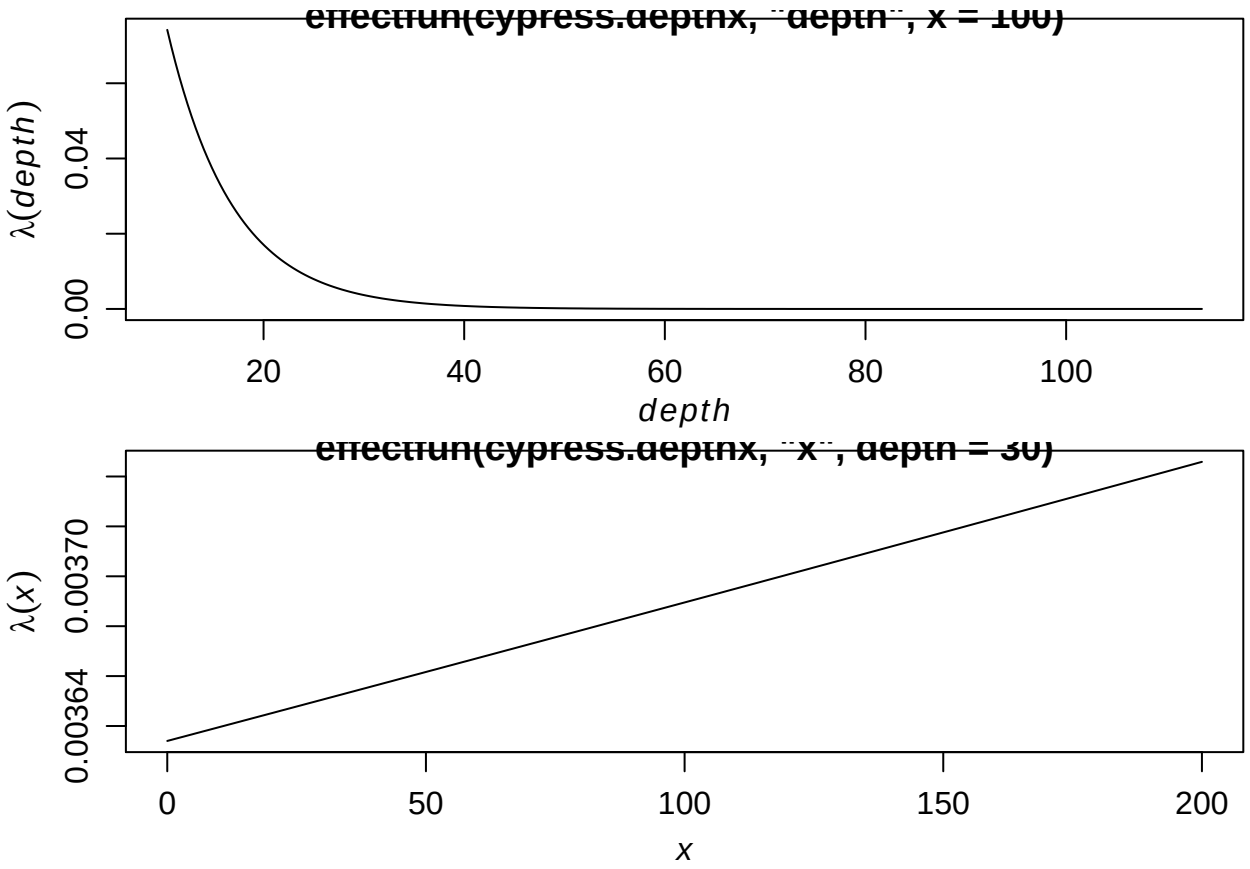
Printing the result gives you the estimated coefficients and associated information.

Plotting it gives you two plots: the predicted intensity and the estimated se of that intensity. The preceding par() puts the two plots one above the other and reduces the margins. The pause=F suppresses waiting between plots.

effectfun() calculates predicted intensity vs a covariate. The arguments are a fitted model and the name of the variable to display on the X axis, in quotes. se.fit=T adds standard errors and 95% confidence intervals. Feeding the result of effectfun() into plot(), as done above, draws a plot, with 95% confidence intervals when computed by effectfun (i.e. se.fit=T).

Multiple covariates can be specified in the usual R formula way. These can be different types, e.g. depth (a pixel image) and x (a coordiate).

```
cypress.depthx <- ppm(cypress.pppw ~ depth + x)
par(mfrow=c(2,1), mar=c(3,3,0,0)+0.2, mgp=c(2,0.8,0))
plot(effectfun(cypress.depthx, 'depth', x=100))
plot(effectfun(cypress.depthx, 'x', depth=30) )
```



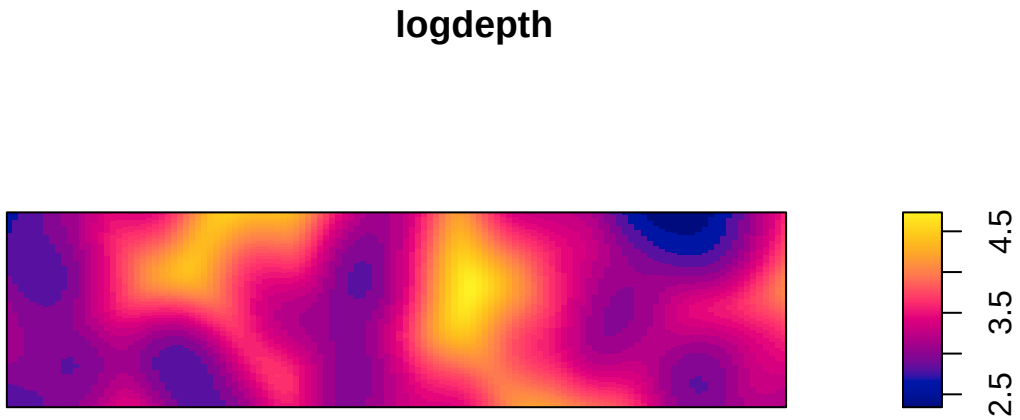

```
par(mfrow=c(1,1))
```

When a model has more than one covariate, effectfun() calculates partial effects. The target variable is specified by name and specific values are given for all the other variables. The predicted intensity is then conditional on the values of other variables. cypress.depthx has two covariates: depth and x. The commands above plot intensity vs depth when x = 100 and intensity vs x when depth = 30.

manipulating pixel images

Spatstat can do arithmetic operations on pixel images. The result is another pixel image. For example, if you wanted to model log lambda = b0 + b1 log(depth), you would do the following:

```
logdepth <- log(depth)
plot(logdepth)
```




```
cypress.ld <- ppm(cypress.pppw ~ logdepth)
plot(effectfun(cypress.ld, 'logdepth', se.fit=T))
```