SpatialKernel

Kostas Goulias

2/27/2018

Kernel Density

First the libraries we need.

```
library(dplyr)       ## Data manipulation

## Warning: package 'dplyr' was built under R version 3.4.2

##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union

library(maps)         ## Projections
library(maptools)     ## Data management

## Loading required package: sp

## Checking rgeos availability: TRUE

library(sp)           ## Data management
library(spdep)        ## Spatial autocorrelation

## Warning: package 'spdep' was built under R version 3.4.3

## Loading required package: Matrix

## Loading required package: spData

## Warning: package 'spData' was built under R version 3.4.3

## To access larger datasets in this package, install the spDataLarge
## package with: `install.packages('spDataLarge')`

library(gstat)        ## Geostatistics
library(splancs)      ## Kernel Density

##
## Spatial Point Pattern Analysis Code in S-Plus
```

```
##
##  Version 2 - Spatial and Space-Time analysis

##
## Attaching package: 'splancs'

## The following object is masked from 'package:dplyr':
##
##     tribble

library(spatstat)     ## Geostatistics

## Warning: package 'spatstat' was built under R version 3.4.2

## Loading required package: spatstat.data

## Warning: package 'spatstat.data' was built under R version 3.4.2

## Loading required package: nlme

##
## Attaching package: 'nlme'

## The following object is masked from 'package:dplyr':
##
##     collapse

## Loading required package: rpart

##
## spatstat 1.53-2       (nickname: 'Quantum Entanglement')
## For an introduction to spatstat, type 'beginner'

## Warning in strptime(x, "%Y-%m-%d-%H-%M-%OS", tz = tz): unknown timezone
## 'zone/tz/2018c.1.0/zoneinfo/America/Los_Angeles'

##
## Note: spatstat version 1.53-2 is out of date by more than 4 months; we rec
ommend upgrading to the latest version.

##
## Attaching package: 'spatstat'

## The following object is masked from 'package:gstat':
##
##     idw

library(RColorBrewer) ## Visualization
library(classInt)     ## Class intervals
library(spgwr)        ## GWR (not sure we need this for now)

## Warning: package 'spgwr' was built under R version 3.4.2
## NOTE: This package does not constitute approval of GWR
## as a method of spatial analysis; see example(gwr)
```

```
library(lattice)      ## Data visualization
```

```
##
## Attaching package: 'lattice'
```

```
## The following object is masked from 'package:spatstat':
##
##     panel.histogram
```

```
library(rgdal)        ## Geospatial data abstraction library
```

```
## Warning: package 'rgdal' was built under R version 3.4.2
```

```
## rgdal: version: 1.2-13, (SVN revision 686)
##  Geospatial Data Abstraction Library extensions to R successfully loaded
##  Loaded GDAL runtime: GDAL 2.1.3, released 2017/20/01
##  Path to GDAL shared files: /Library/Frameworks/R.framework/Versions/3.4/R
esources/library/rgdal/gdal
##  Loaded PROJ.4 runtime: Rel. 4.9.3, 15 August 2016, [PJ_VERSION: 493]
##  Path to PROJ.4 shared files: /Library/Frameworks/R.framework/Versions/3.4
/Resources/library/rgdal/proj
##  Linking to sp version: 1.2-5
```

Display of different kernels

From spastat:

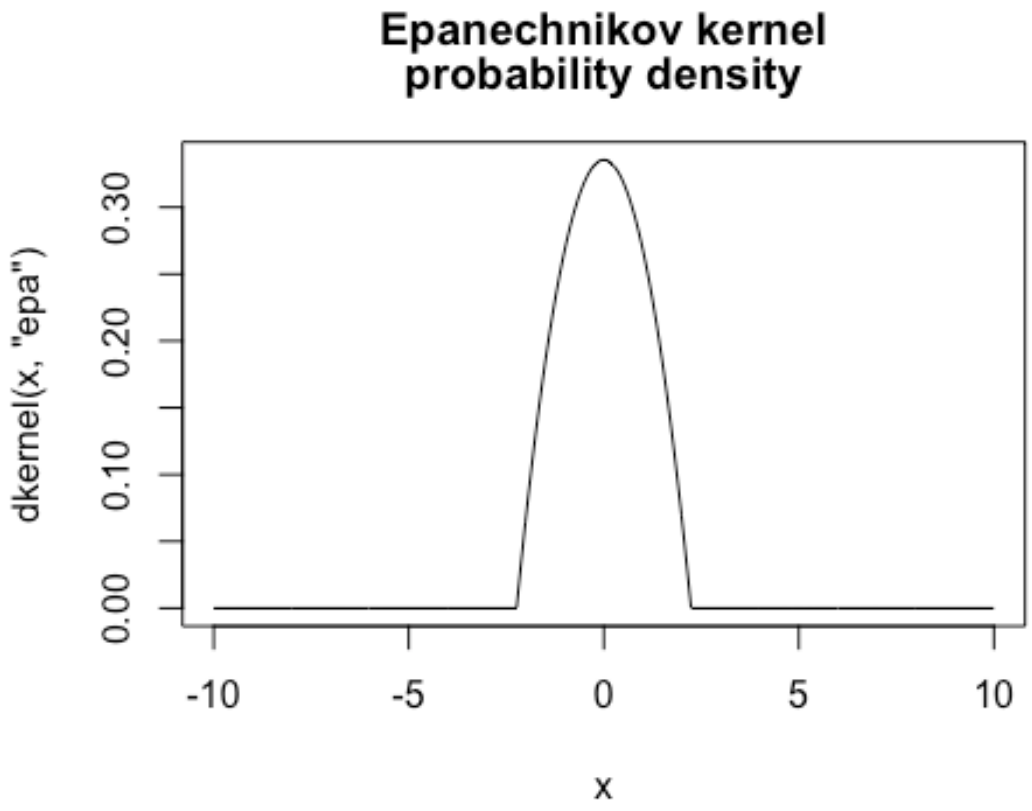
These functions give the probability density, cumulative distribution function, quantile function and random generation for several distributions used in kernel estimation for one-dimensional (numerical) data.

The available kernels are those used in density.default, namely "gaussian", "rectangular", "triangular", "epanechnikov", "biweight", "cosine" and "optcosine". For more information about these kernels, see density.default.

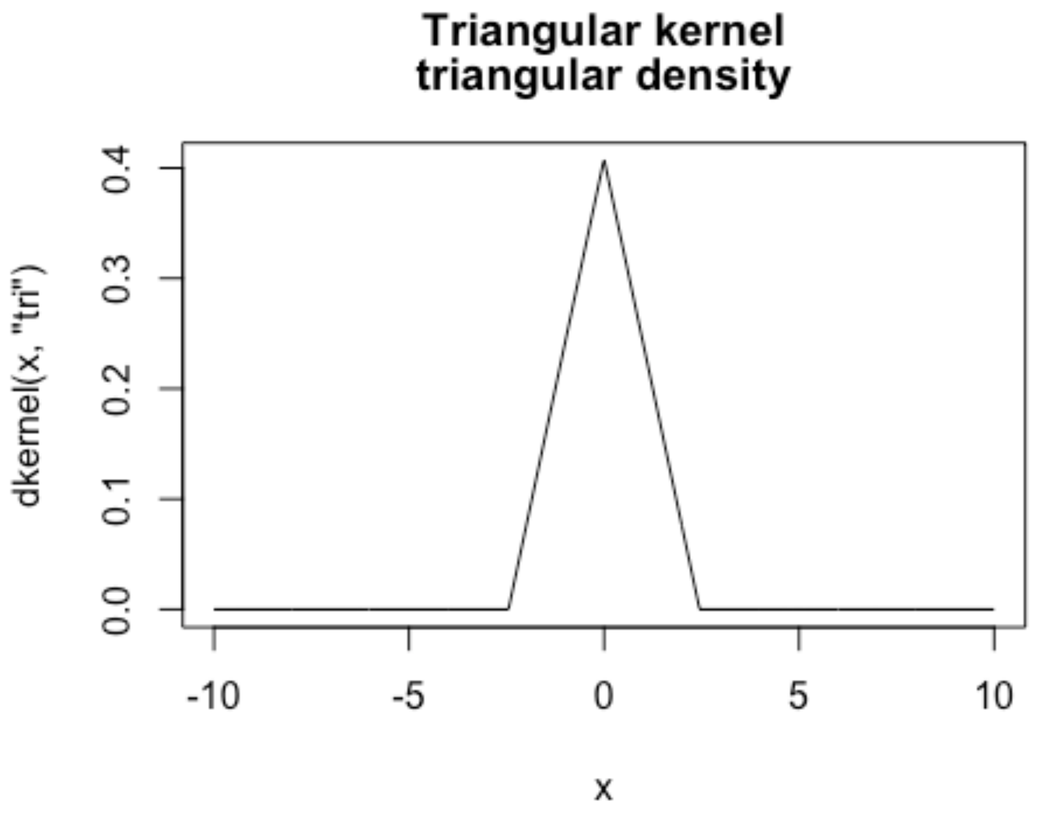
dkernel gives the probability density, pkernel gives the cumulative distribution function, qkernel gives the quantile function, and rkernel generates random deviates.

Densities:

```
x <- seq(-10,10,length=1000)
  plot(x, dkernel(x, "epa"), type="l",
           main=c("Epanechnikov kernel", "probability density"))
```



```r
plot(x, dkernel(x, "tri"), type="l",
           main=c("Triangular kernel", "triangular density"))
```



```
```

```r
plot(x, dkernel(x, "gaussian"), type="l",
           main=c("gaussian", "gaussian density distribution function"))
```

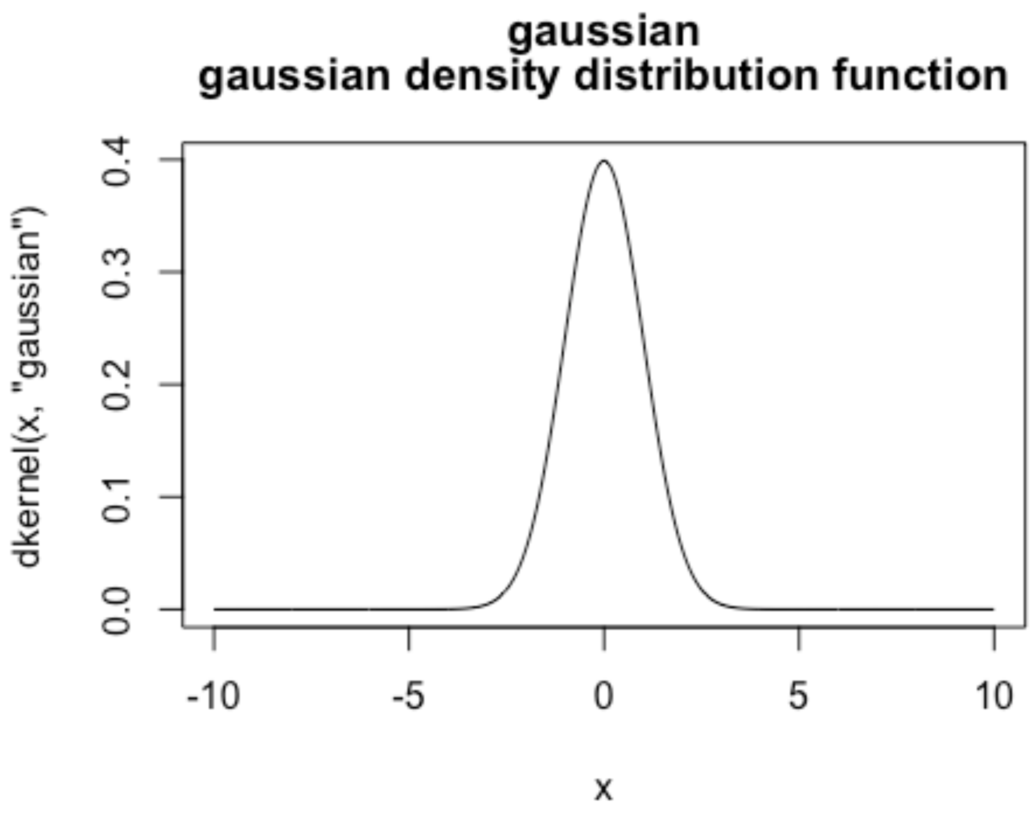


Illustration with examples of kernel density estimation in 1d (from kedd package)

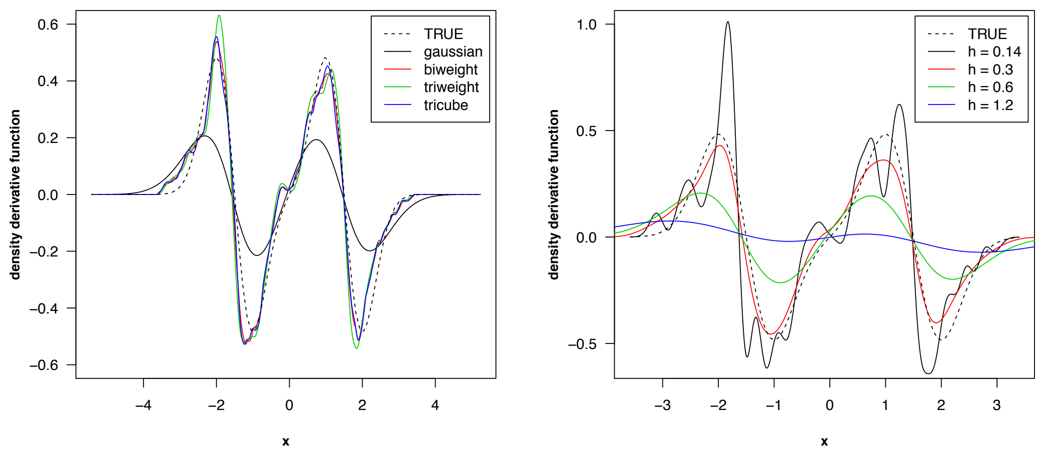


Figure 2: (Left) Different kernels for estimation, with $h = 0.6$. (Right) Effect of the bandwidth on the kernel estimator.

Kernel	$K(x;r)$
Gaussian	$K(x;\infty) = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{x^2}{2}\right) 1_{]-\infty,+\infty[}$
Epanechnikov	$K(x;2) = \frac{3}{4}\left(1-x^2\right) 1_{(\lvert x\rvert \le 1)}$
Uniform	$K(x;0) = \frac{1}{2} 1_{(\lvert x\rvert \le 1)}$
Triangular	$K(x;1) = (1-\lvert x\rvert) 1_{(\lvert x\rvert \le 1)}$
Triweight	$K(x;6) = \frac{35}{32}\left(1-x^2\right)^3 1_{(\lvert x\rvert \le 1)}$
Tricube	$K(x;9) = \frac{70}{81}\left(1-\lvert x\rvert^3\right)^3 1_{(\lvert x\rvert \le 1)}$
Biweight	$K(x;4) = \frac{15}{16}\left(1-x^2\right)^2 1_{(\lvert x\rvert \le 1)}$
Cosine	$K(x;\infty) = \frac{\pi}{4} \cos\left(\frac{\pi}{2}x\right) 1_{(\lvert x\rvert \le 1)}$

r is the number of derivatives the function allows (first=1, second=2, and so forth).

Some clear youtube videos:

Nonparametrics basics by Luc Anselin: https://www.youtube.com/watch?v=0kHy7dzyT90

Illustration of kernel density: https://www.youtube.com/watch?v=PBZVTjmhl74

A complete example of point pattern analysis in R with an example and code: http://r-video-tutorial.blogspot.com/2015/05/introductory-point-pattern-analysis-of.html

2 D Kernel Density Estimation

The data. The file h2kb.rds is on gauchospace

```
my2k = readRDS('~/Documents/COURSES UCSB/Course Winter 2018/California/h2kb.r
ds')
class(my2k)

## [1] "data.frame"

summary(my2k)

##        Z1                  Z2              XCORD             YCORD      
##  Min.   :   0.000   Min.   : 0.000   Min.   :-124.2   Min.   :32.56  
##  1st Qu.:   5.524   1st Qu.: 2.000   1st Qu.:-121.9   1st Qu.:34.07  
##  Median :  32.154   Median : 6.000   Median :-119.9   Median :36.59  
##  Mean   :  68.084   Mean   : 8.414   Mean   :-120.0   Mean   :36.16  
##  3rd Qu.:  78.969   3rd Qu.:12.000   3rd Qu.:-118.2   3rd Qu.:37.84  
##  Max.   :1383.520   Max.   :95.000   Max.   :-114.4   Max.   :41.95  

data <- my2k
class(data)

## [1] "data.frame"
```

This is just a data frame like the files we created before for data analysis. It contains 2,000 locations in California with their X and Y coordinates and the value of two variables Z1 and Z2. In this example we are going to use only the points (XCORD and YCORD) and not the variables Z1 and Z2.

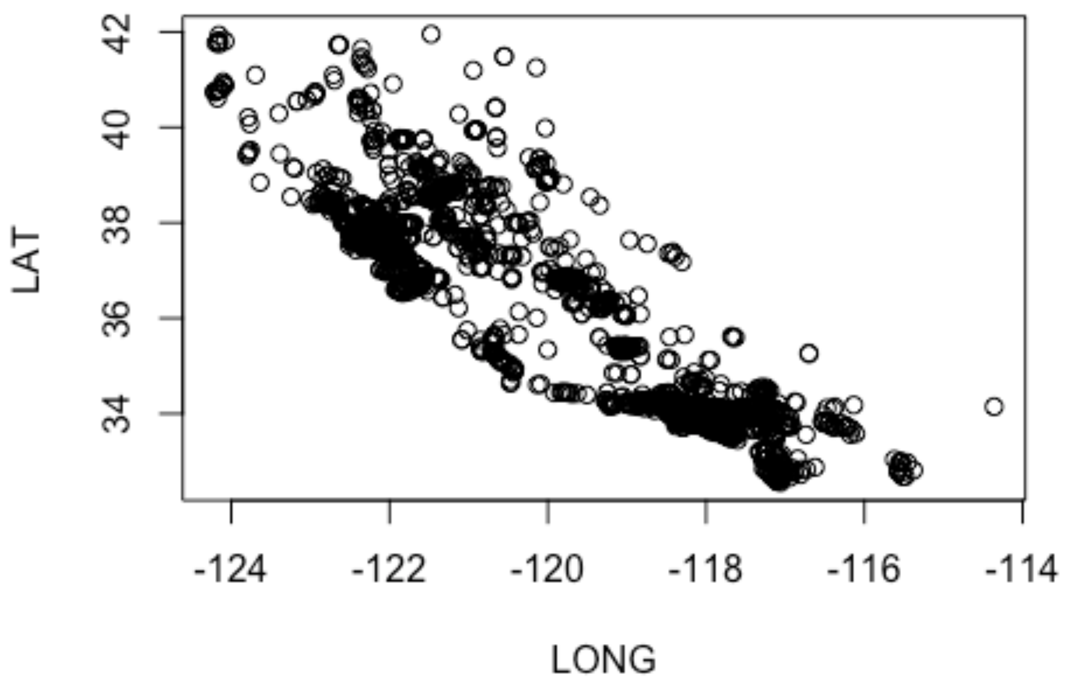
I also rename the my2k to data to eliminate too many names in the code.

We need to make this data frame into a spatial object

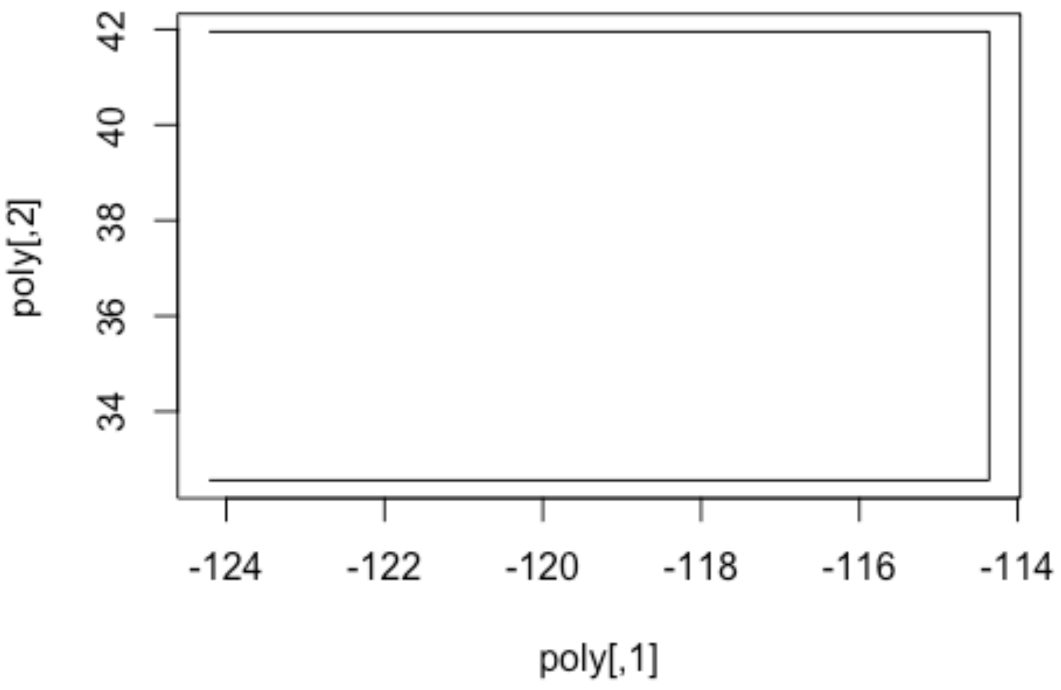
Create a matrix of coordinates from the point coordinates of the data file my2k (renamed data)

```
sp_point <- cbind(data$XCORD, data$YCORD)
colnames(sp_point) <- c("LONG","LAT")
head(sp_point)

##            LONG      LAT
## [1,] -118.1920 34.06835
## [2,] -121.4452 38.63569
## [3,] -119.6622 36.34007
## [4,] -122.6362 38.22512
## [5,] -119.0181 35.32511
## [6,] -122.2539 37.85491

class(sp_point)

## [1] "matrix"
```

```r
plot(sp_point)
```



```r
poly <- as.points(c(min(sp_point[,1]),max(sp_point[,1]),max(sp_point[,1]),min
(sp_point[,1])),c(max(sp_point[,2]),max(sp_point[,2]),min(sp_point[,2]),min(s
p_point[,2])))
plot(poly, type="l")
```



```
class(poly)

## [1] "matrix"
```

Considering my data I need to find a bandwidth that yields a minimum squared error.

In essence we try to balance between bias and variance of the estimator of the data distribution.
The literature has many different equations for the MSE and one of them is implemented in mse2d

```
mserw <- mse2d(sp_point, poly=poly, nsmse=100, range=0.1)
summary(mserw)

##     Length Class  Mode
## mse 100    -none- numeric
## h   100    -none- numeric

class(mserw)

## [1] "list"

summary(mserw$mse, mserw$h)
```

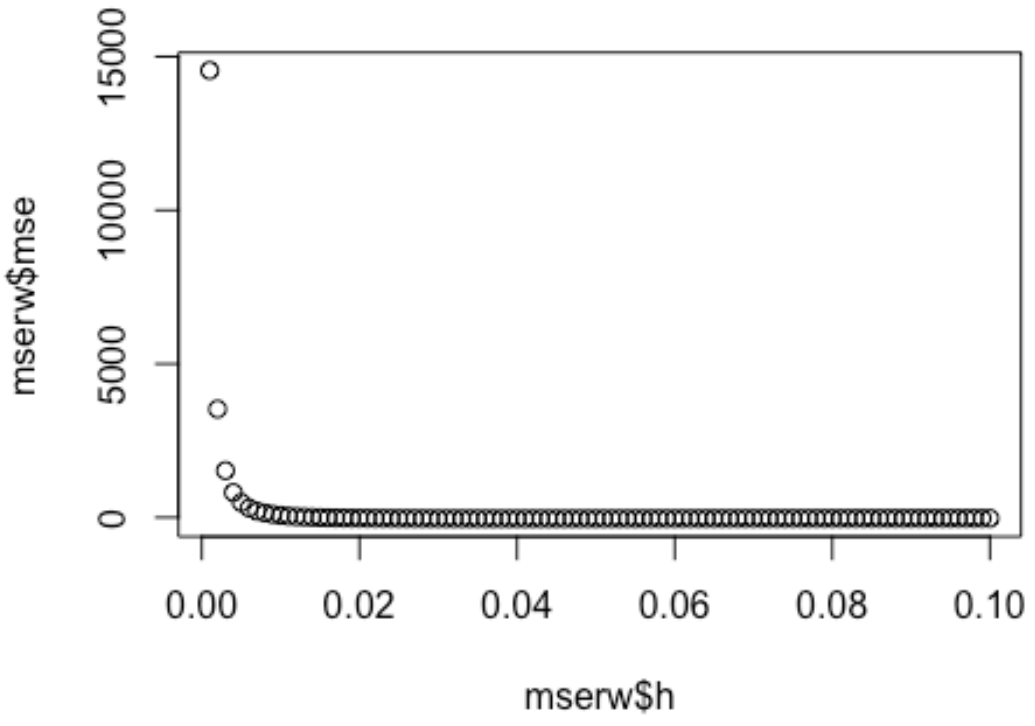
```
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
##   -39.34   -36.56   -31.54   190.48   -27.90 14548.07
```

```
print(mserw)
```

```
## $mse
##   [1] 14548.065174  3528.552430  1525.961435   812.226359   487.949393
##   [6]   307.022040   205.848356   145.510387   100.714976    69.034237
##  [11]    44.258897    27.054545    16.546933     6.550603    -2.842129
##  [16]   -10.423693   -15.953669   -19.235131   -22.586840   -23.945662
##  [21]   -25.835312   -27.651225   -29.566718   -31.049582   -32.614498
##  [26]   -32.710623   -33.667996   -34.440448   -36.010829   -37.397579
##  [31]   -37.339740   -37.422675   -37.918626   -37.994552   -38.459743
##  [36]   -38.321509   -38.965216   -39.104373   -39.122741   -39.203869
##  [41]   -39.339610   -39.086768   -38.785756   -38.703801   -38.612691
##  [46]   -38.647101   -38.559791   -38.286798   -37.811139   -37.628699
##  [51]   -37.483307   -37.138080   -37.002425   -36.826826   -36.475117
##  [56]   -36.053218   -35.856010   -35.678838   -35.366621   -35.079683
##  [61]   -34.828820   -34.400073   -34.176530   -33.995798   -33.680350
##  [66]   -33.430802   -33.170080   -32.903188   -32.661373   -32.322054
##  [71]   -32.071001   -31.894028   -31.815219   -31.665847   -31.415496
##  [76]   -31.353039   -31.230458   -31.021047   -30.750829   -30.577895
##  [81]   -30.473351   -30.336189   -30.121304   -29.984881   -29.868464
##  [86]   -29.727728   -29.545104   -29.291671   -29.213347   -29.063180
##  [91]   -28.935291   -28.751177   -28.609890   -28.468920   -28.224554
##  [96]   -28.054579   -27.939378   -27.792318   -27.683715   -27.509605
## 
## $h
##   [1] 0.001 0.002 0.003 0.004 0.005 0.006 0.007 0.008 0.009 0.010 0.011
##  [12] 0.012 0.013 0.014 0.015 0.016 0.017 0.018 0.019 0.020 0.021 0.022
##  [23] 0.023 0.024 0.025 0.026 0.027 0.028 0.029 0.030 0.031 0.032 0.033
##  [34] 0.034 0.035 0.036 0.037 0.038 0.039 0.040 0.041 0.042 0.043 0.044
##  [45] 0.045 0.046 0.047 0.048 0.049 0.050 0.051 0.052 0.053 0.054 0.055
##  [56] 0.056 0.057 0.058 0.059 0.060 0.061 0.062 0.063 0.064 0.065 0.066
##  [67] 0.067 0.068 0.069 0.070 0.071 0.072 0.073 0.074 0.075 0.076 0.077
##  [78] 0.078 0.079 0.080 0.081 0.082 0.083 0.084 0.085 0.086 0.087 0.088
##  [89] 0.089 0.090 0.091 0.092 0.093 0.094 0.095 0.096 0.097 0.098 0.099
## [100] 0.100
```

```
help(mse2d)
```

```
plot(mserw$h, mserw$mse)
```

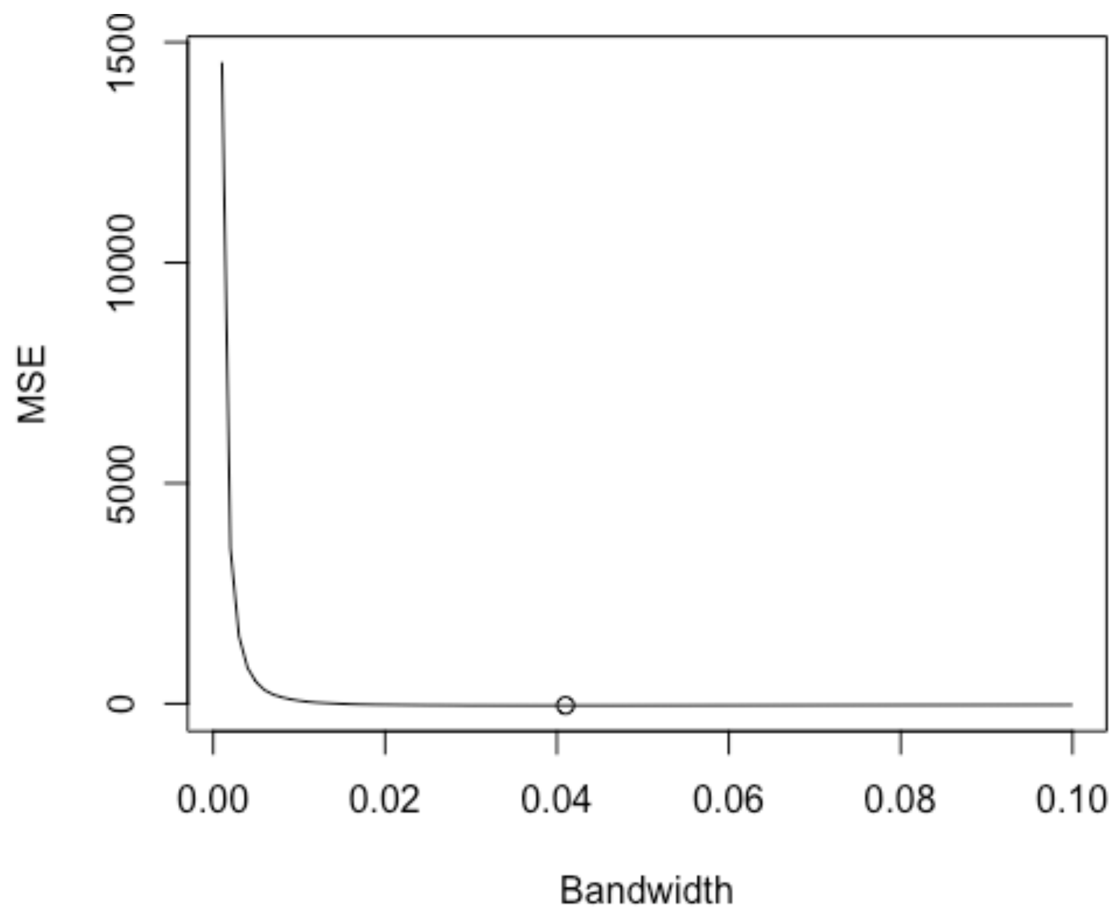


Extract from the list mserw and variable mserw$mse the minimum value and stored it in bw

```
bw <- mserw$h[which.min(mserw$mse)] ## Bandwidth=.01
summary(bw)

##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   0.041   0.041   0.041   0.041   0.041   0.041
```

A nicer way of displaying where the minimum MSE is (i)

```
par(mar=c(4,4,0.5,0.5))
plot(x=mserw$h, y=mserw$mse, xlab="Bandwidth", ylab="MSE", type="l")
i<-which.min(mserw$mse)
points(mserw$h[i], mserw$mse[i])
```



I inserted class for each object to track the sequence of steps

```
sp_points <- SpatialPoints(coords=sp_point, proj4string=CRS("+proj=utm +zone=
10 +datum=WGS84"))
class(sp_points)

## [1] "SpatialPoints"
## attr(,"package")
## [1] "sp"

grd <- Sobj_SpatialGrid(sp_points,maxDim=100)$SG
class(grd)

## [1] "SpatialGrid"
## attr(,"package")
## [1] "sp"

grd <- GridTopology(summary(grd)$grid[,1],cellsize=summary(grd)$grid[,2],cell
s.dim=summary(grd)$grid[,3])
class(grd)

## [1] "GridTopology"
## attr(,"package")
## [1] "sp"
```

In this part we estimate 4 kernel densities with different bandwidths. The first is based on minimizing the mean squared error with an equation derived by Diggle. the other three bandwidths (h0=.0005, h0=0.1, and h0=.15) are just trials to see what we get.

We use the points in sp_point, in the polygon poly defined from the locations of these points, and a grid also defined by these points

```
kernel1 <- spkernel2d(sp_point, poly=poly, h0=bw, grd=grd)
kernel2 <- spkernel2d(sp_point, poly=poly, h0=.005, grd=grd)
kernel3 <- spkernel2d(sp_point, poly=poly, h0=.1, grd=grd)
kernel4 <- spkernel2d(sp_point, poly=poly, h0=.15, grd=grd)
```

Just to see what this function does.

```
help(spkernel2d)
```

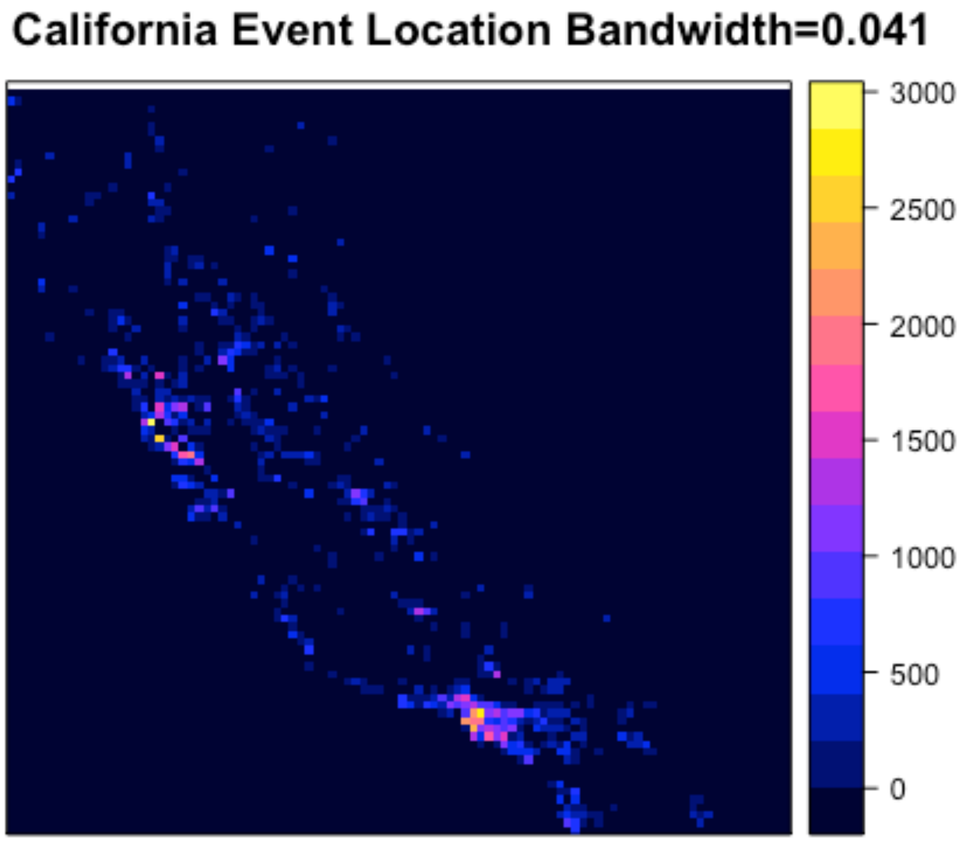
Just to check the contents and class of object kernel1

```
class(kernel1)

## [1] "numeric"

summary(kernel1)

##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
##    0.00    0.00    0.00   20.54    0.00 2844.22     100
```

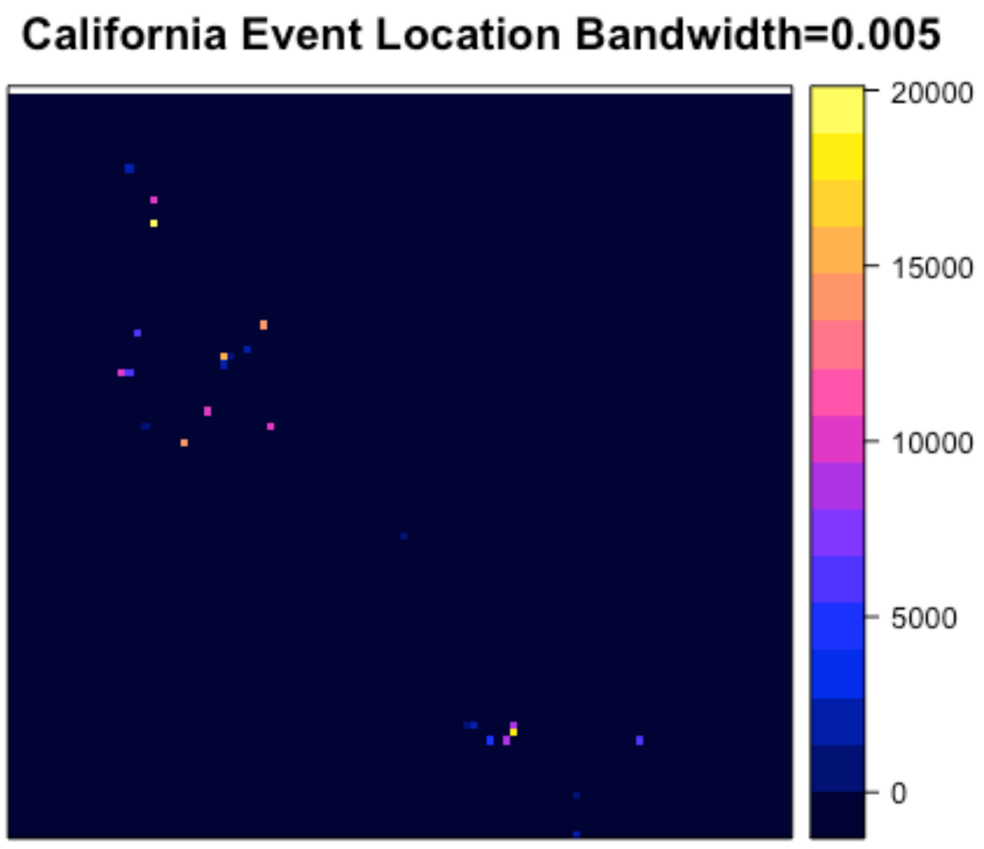
Saving the kernel deinsity estimates in one database and a grided databse (SpatialGridDataFrame) Then, plot the kernel densities

Just to observe the kernel density of the otpimized bandwidth h0=0.041

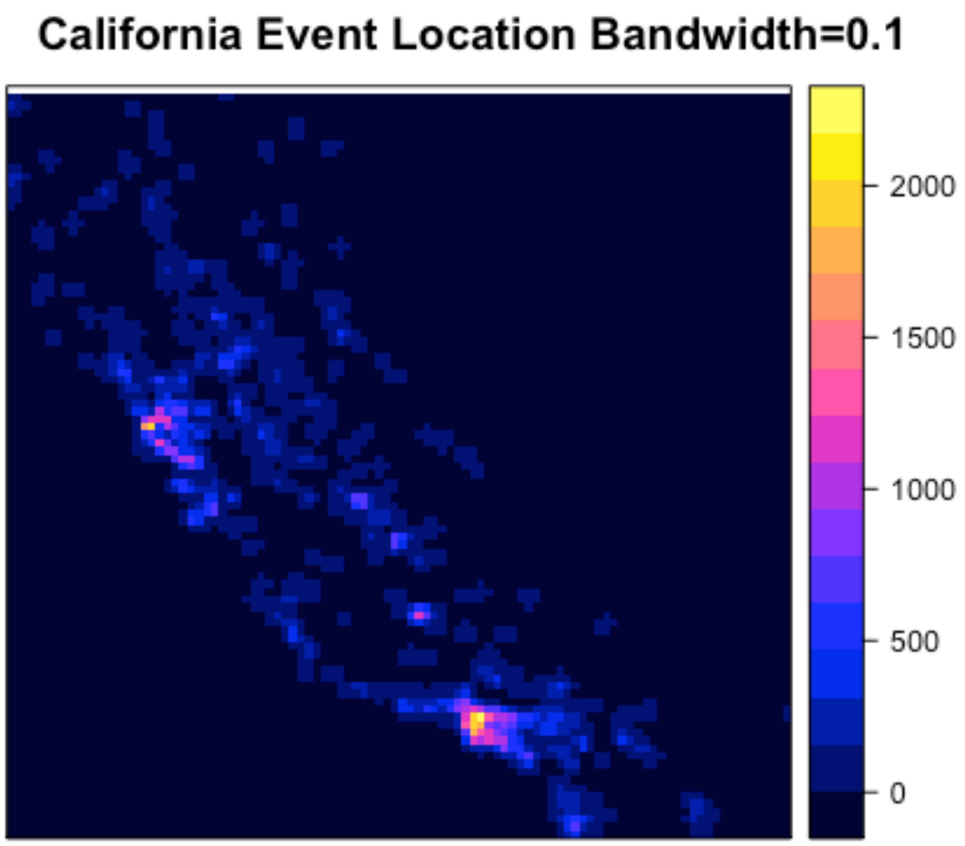
```
CAdf <- data.frame(kernel1=kernel1)
CAsg <- SpatialGridDataFrame(grd, data=CAdf)
spplot(CAsg, main="California Event Location Bandwidth=0.041")
```



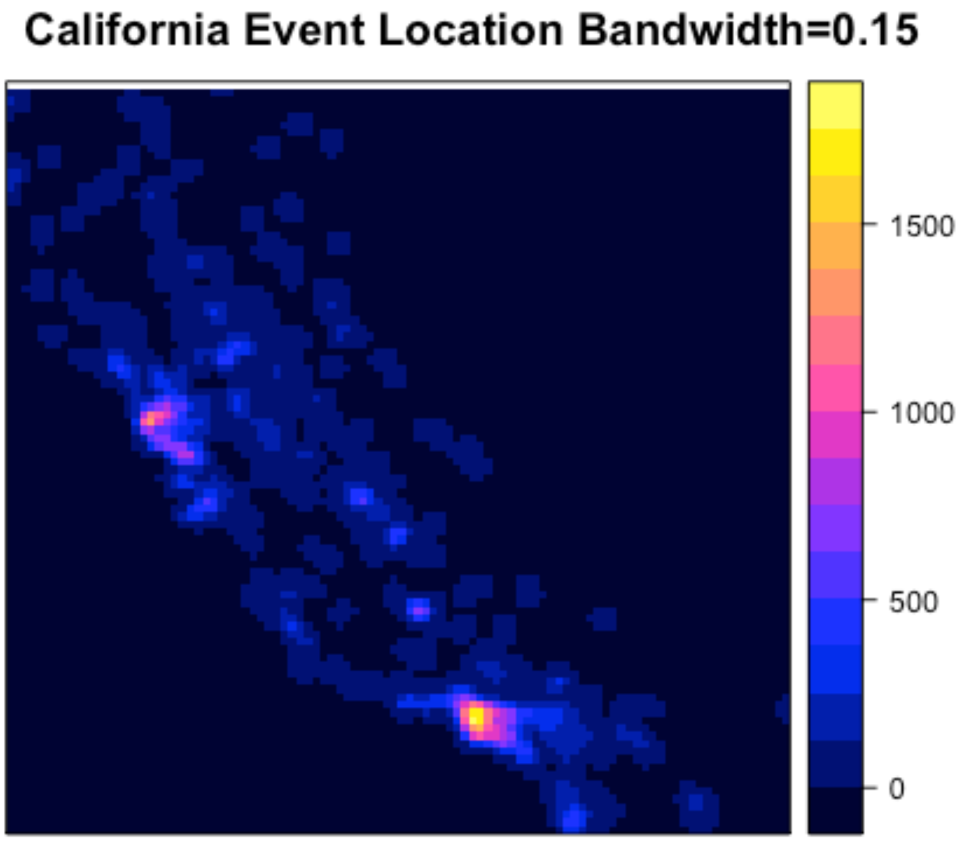
```
CAdf2 <- data.frame(kernel2=kernel2)
CAsg2 <- SpatialGridDataFrame(grd, data=CAdf2)
spplot(CAsg2, main="California Event Location Bandwidth=0.005")
```



```
CAdf3 <- data.frame(kernel3=kernel3)
CAsg3 <- SpatialGridDataFrame(grd, data=CAdf3)
spplot(CAsg3, main="California Event Location Bandwidth=0.1")
```



```
CAdf4 <- data.frame(kernel4=kernel4)
CAsg4 <- SpatialGridDataFrame(grd, data=CAdf4)
spplot(CAsg4, main="California Event Location Bandwidth=0.15")
```

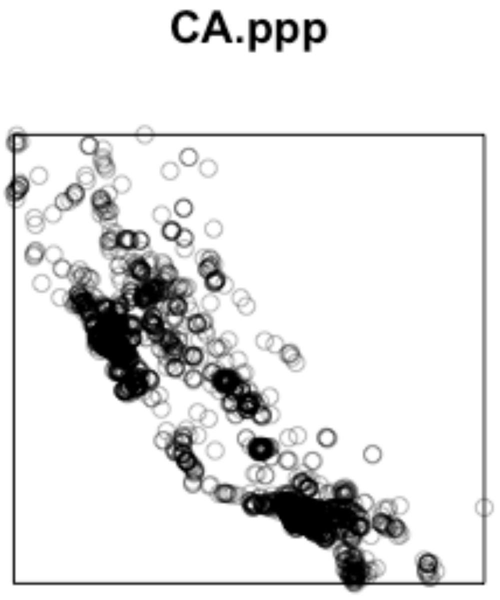


We can do the same with planar point patterns (see Tuesdays examples) and use spatstats. The terminology this time uses sigma for bandwidth (and the h reported here should 2 time the h in splancs)I checked this with a plot.

```
CA.ppp <-
ppp(x=sp_point[,1],y=sp_point[,2],window=owin(bbox(sp_point)[1,],bbox(sp_poin
t)[2,]))
```

```
## Warning: data contain duplicated points
```

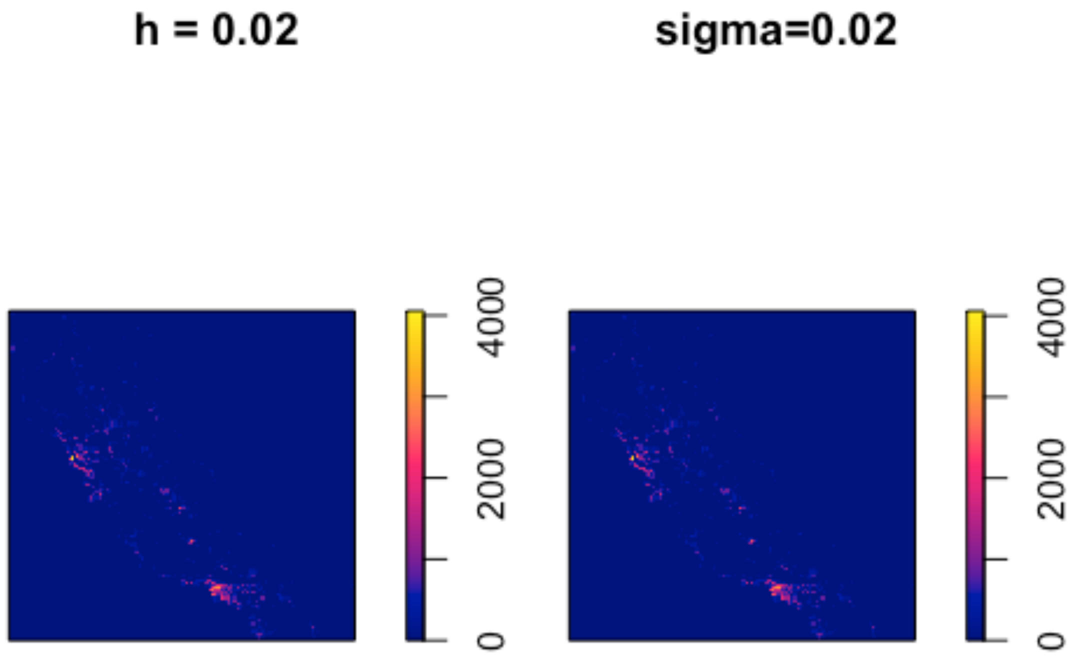
```
plot(CA.ppp)
```



```
summary(CA.ppp)
```

```
## Planar point pattern:  2000 points
## Average intensity 21.59824 points per square unit
##
## *Pattern contains duplicated points*
##
## Coordinates are given to 6 decimal places
##
## Window: rectangle = [-124.21882, -114.3603] x [32.55885, 41.95175] units
## Window area = 92.6001 square units
```

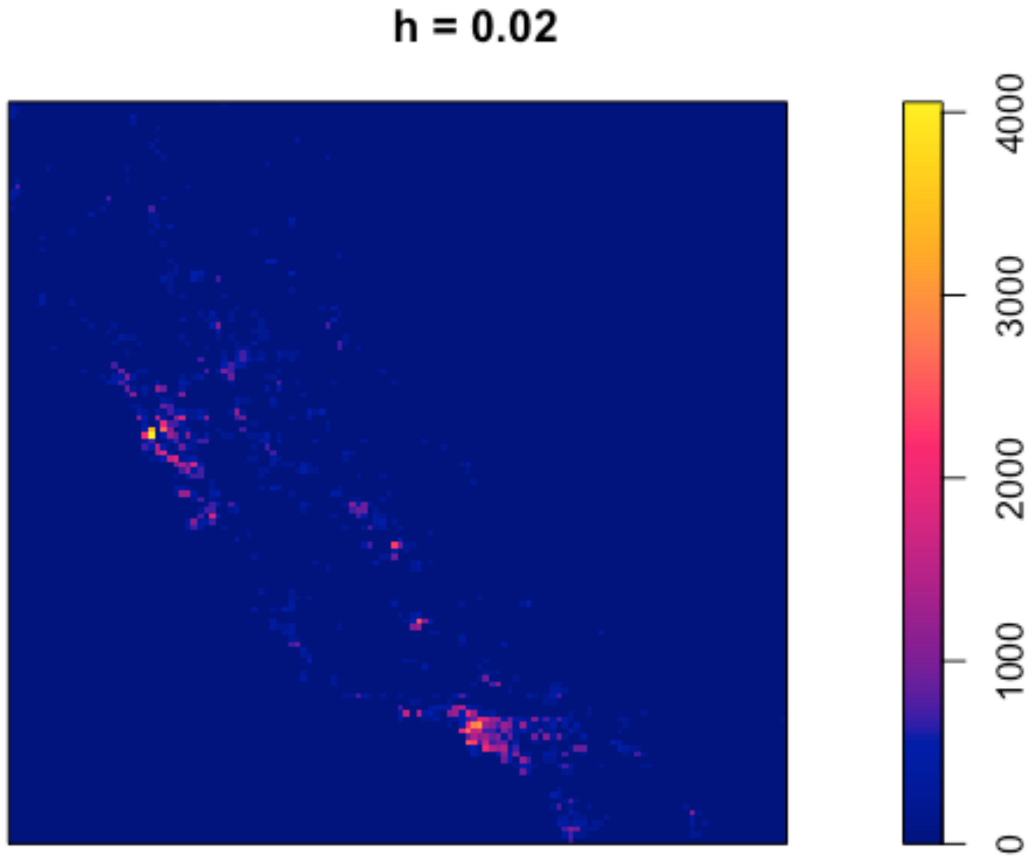
```r
class(CA.ppp)

## [1] "ppp"

par(mfrow=c(1,2), mar=c(0, 1, 1, 1.5))
plot(density.ppp(CA.ppp, sigma = bw.diggle(CA.ppp),edge=T),main=paste("h =",r
ound(bw.diggle(CA.ppp),2)))
plot(density.ppp(CA.ppp, sigma = 0.02,edge=T),main="sigma=0.02")
```



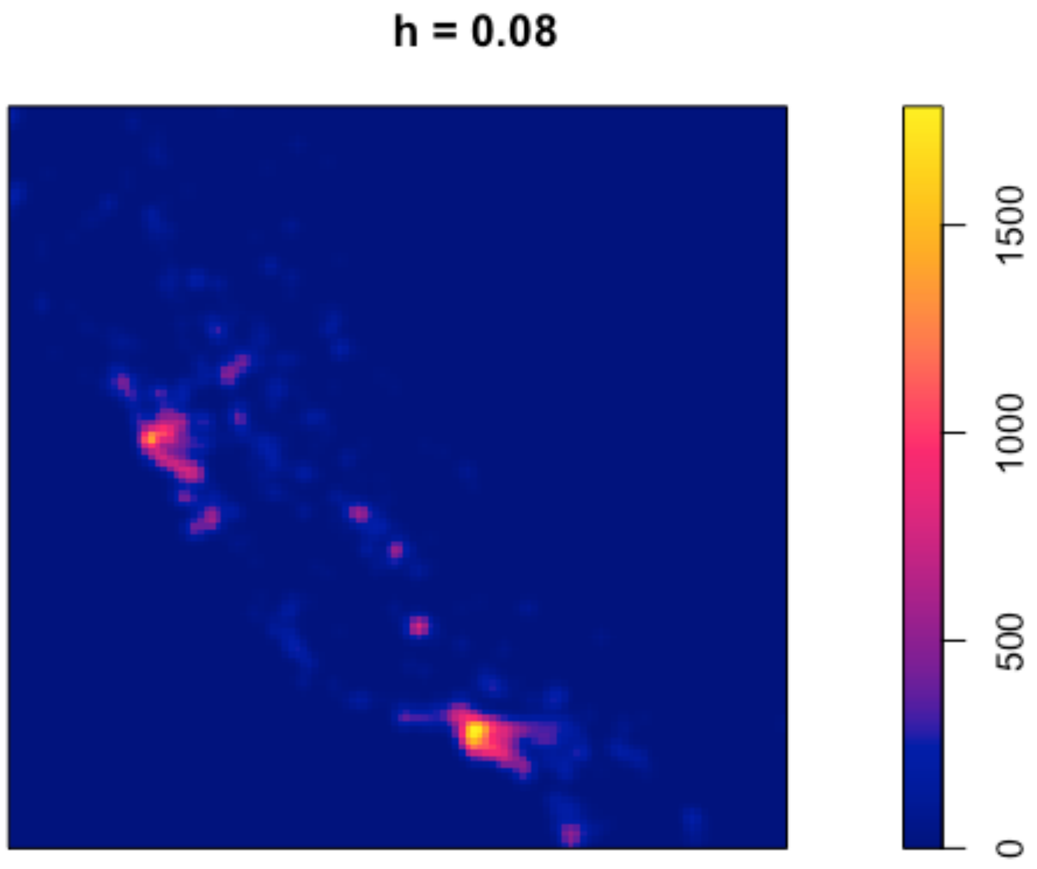
the options one uses can make a big difference because the bandwidth can be very different.

?density.ppp

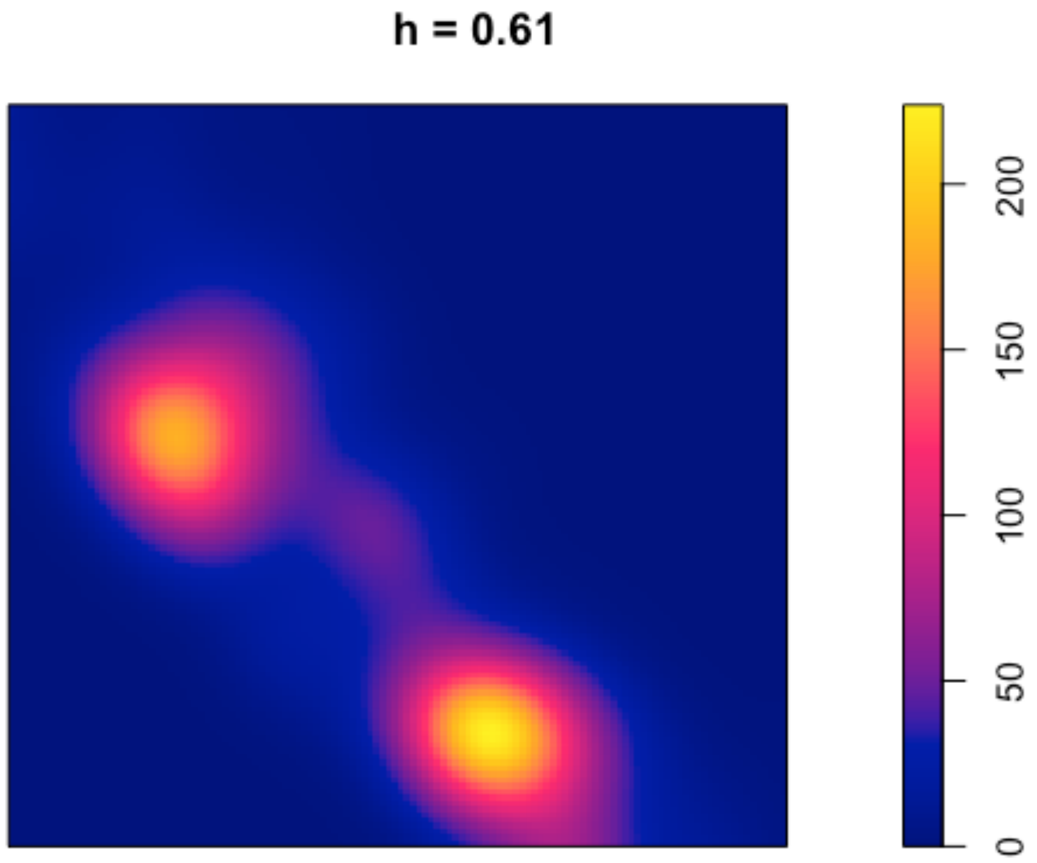
```
par(mar=c(0, 0, 0, 0), mar=c(0, 1, 1, 1.5))

plot(density.ppp(CA.ppp, sigma = bw.diggle(CA.ppp),edge=T),main=paste("h =",r
ound(bw.diggle(CA.ppp),2)))
```



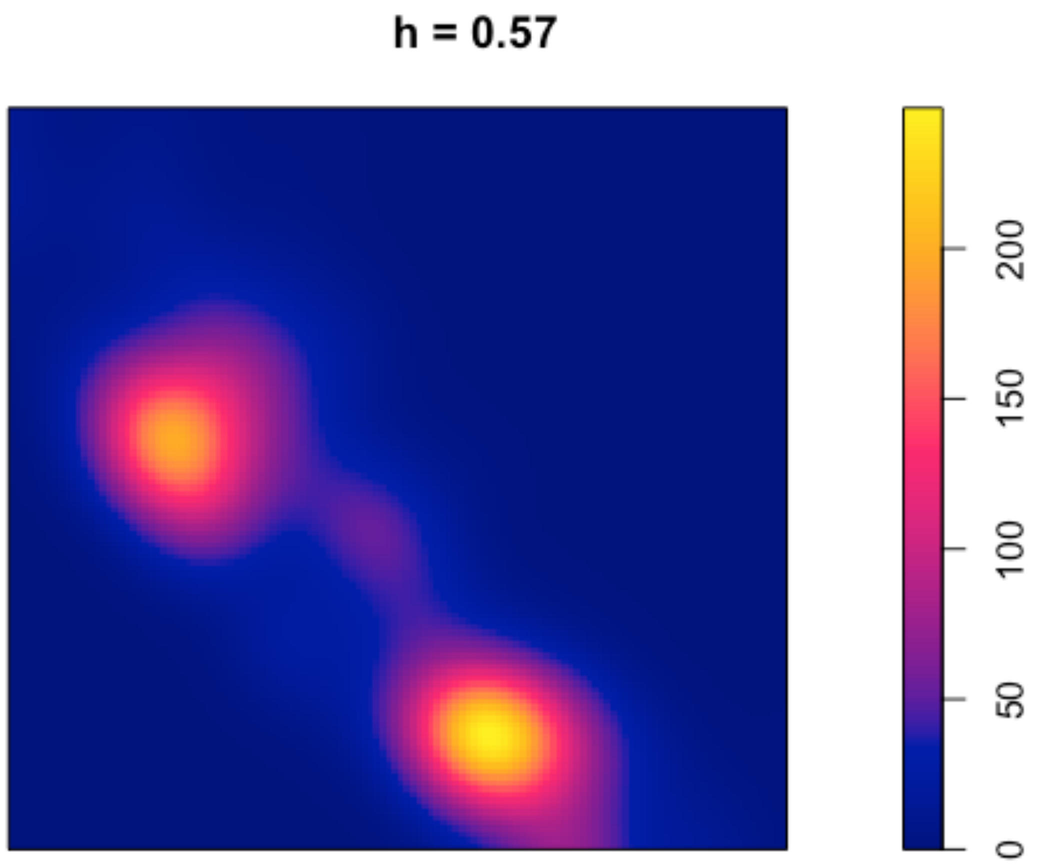
```
par(mar=c(0, 0, 0, 0), mar=c(0, 1, 1, 1.5))

plot(density.ppp(CA.ppp, sigma = bw.ppl(CA.ppp),edge=T),main=paste("h =",roun
d(bw.ppl(CA.ppp),2)))
```



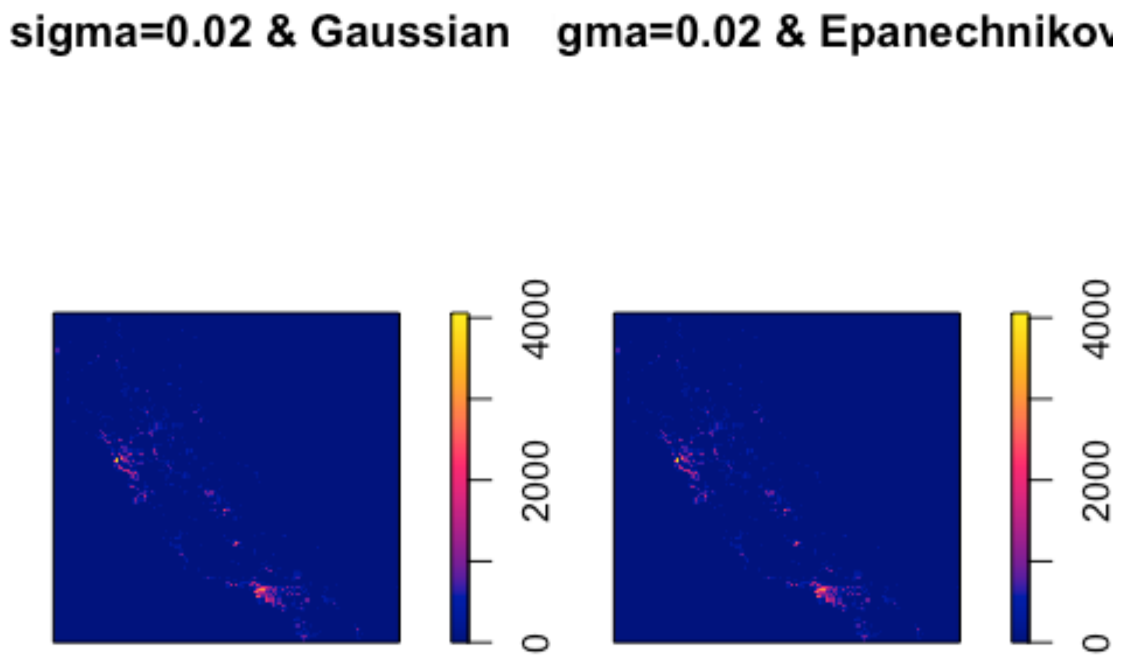
```
par(mar=c(0, 0, 0, 0), mar=c(0, 1, 1, 1.5))

plot(density.ppp(CA.ppp, sigma = bw.scott(CA.ppp)[2],edge=T),main=paste("h ="
,round(bw.scott(CA.ppp)[2],2)))
```



```
par(mar=c(0, 0, 0, 0), mar=c(0, 1, 1, 1.5))
plot(density.ppp(CA.ppp, sigma = bw.scott(CA.ppp)[1],edge=T),main=paste("h ="
,round(bw.scott(CA.ppp)[1],2)))
```

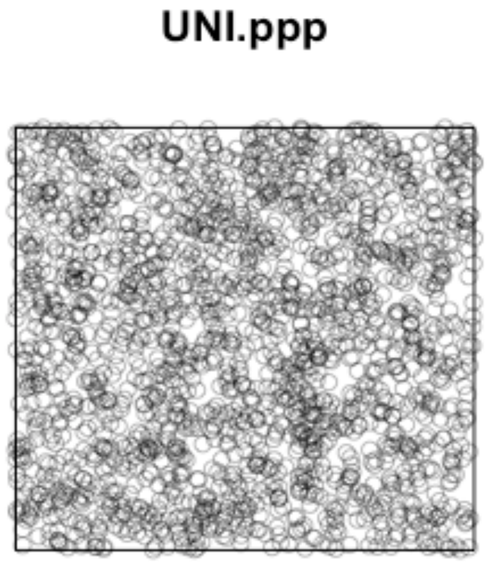


```
par(mfrow=c(1,2), mar=c(0, 1, 1, 1.5))
plot(density.ppp(CA.ppp, sigma = 0.02,edge=T),main="sigma=0.02 & Gaussian", k
ernel="gaussian" )
plot(density.ppp(CA.ppp, sigma = 0.02,edge=T),main="sigma=0.02 & Epanechnikov
", kernel="epa" )
```



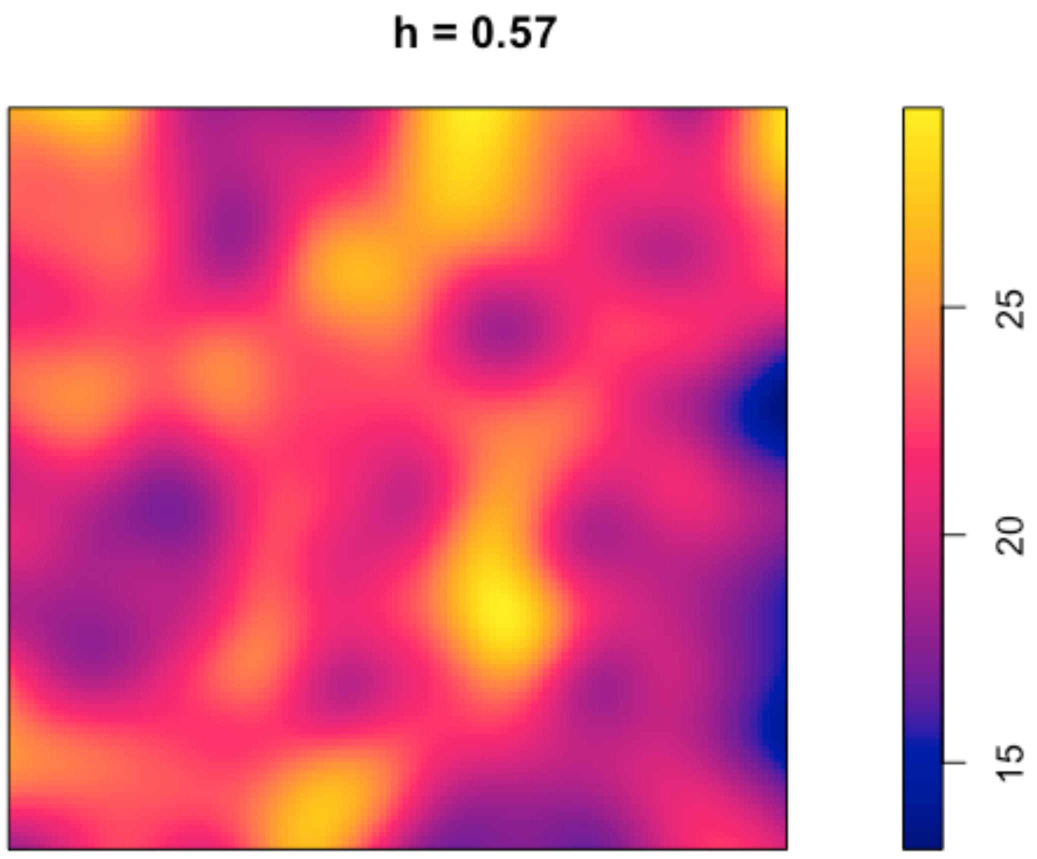
What happens if we have a CSR?

```
## Random points

u.x <- runif(n=nrow(sp_point), min=bbox(sp_point)[1,1], max=bbox(sp_point)[1,
2])
u.y <- runif(n=nrow(sp_point), min=bbox(sp_point)[2,1], max=bbox(sp_point)[2,
2])

UNI.ppp <- ppp(x=u.x,y=u.y,window=owin(bbox(sp_point)[1,],bbox(sp_point)[2,])
)
plot(UNI.ppp)
```

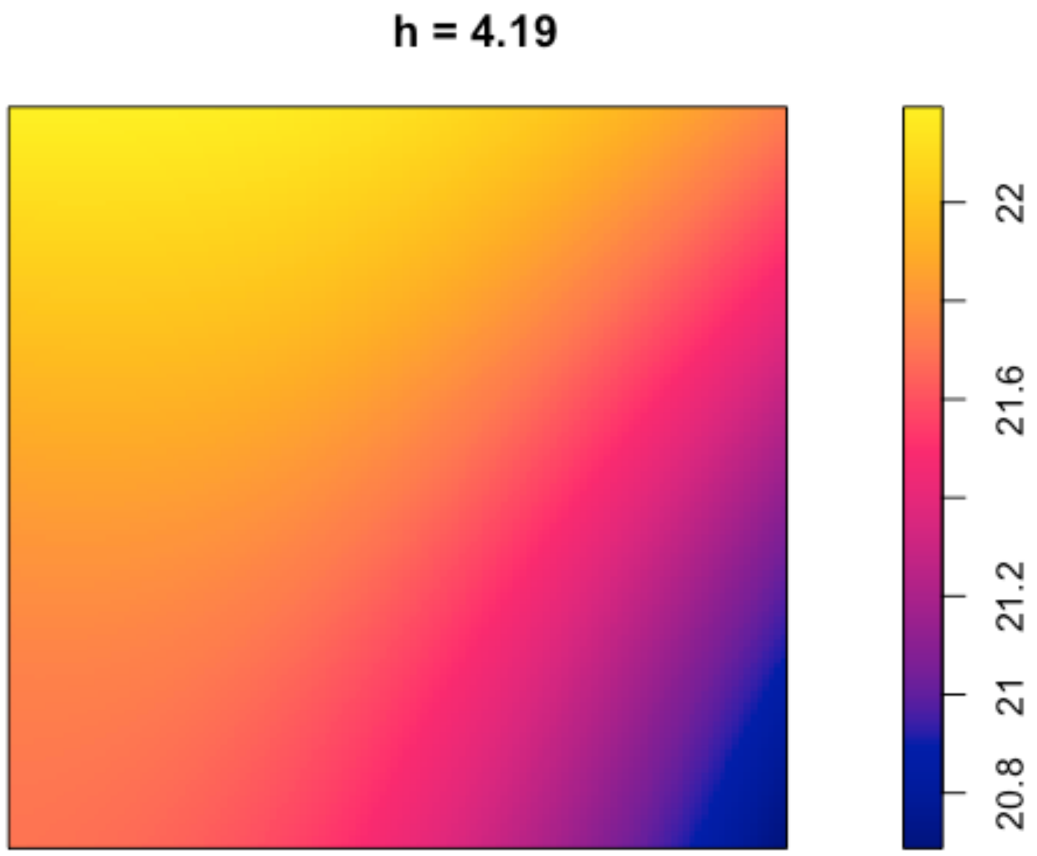


```
summary(UNI.ppp)

## Planar point pattern:  2000 points
## Average intensity 21.59824 points per square unit
##
## Coordinates are given to 6 decimal places
##
## Window: rectangle = [-124.21882, -114.3603] x [32.55885, 41.95175] units
## Window area = 92.6001 square units
```

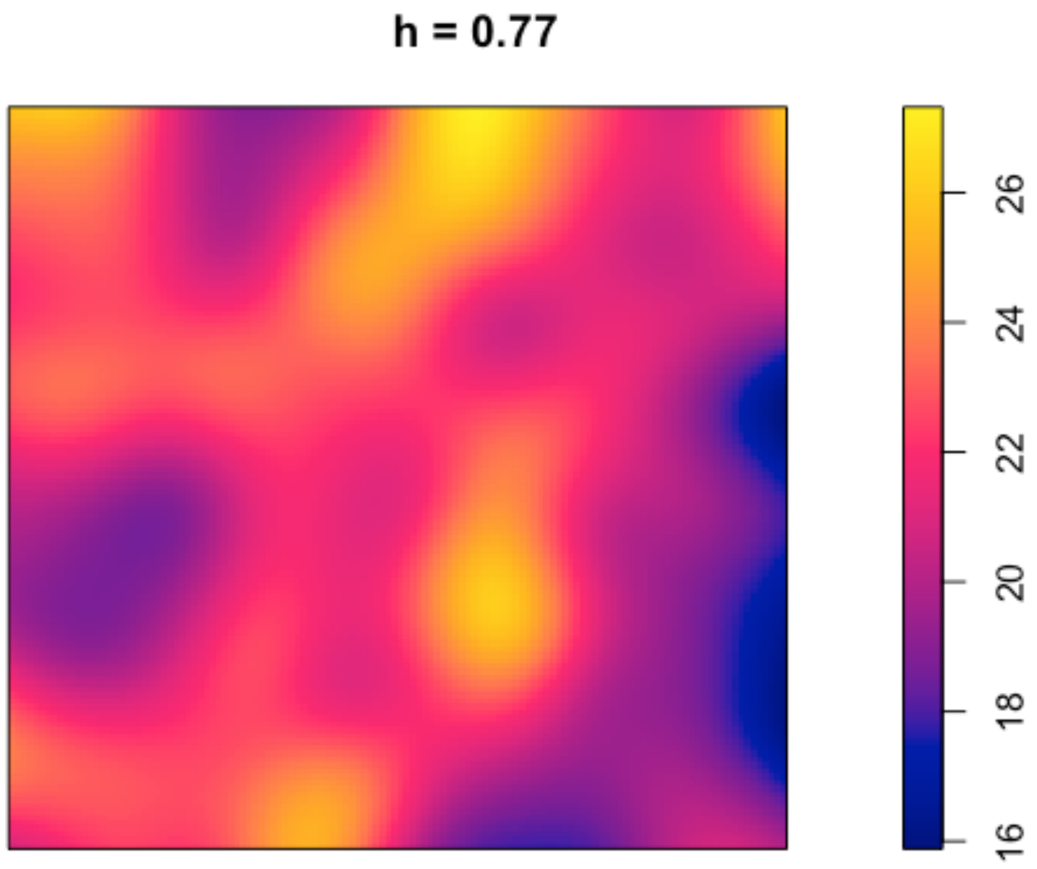
```r
par( mar=c(0, 1, 1, 1.5))
plot(density.ppp(UNI.ppp, sigma = bw.diggle(UNI.ppp),edge=T),main=paste("h ="
,round(bw.diggle(UNI.ppp),2)))
```



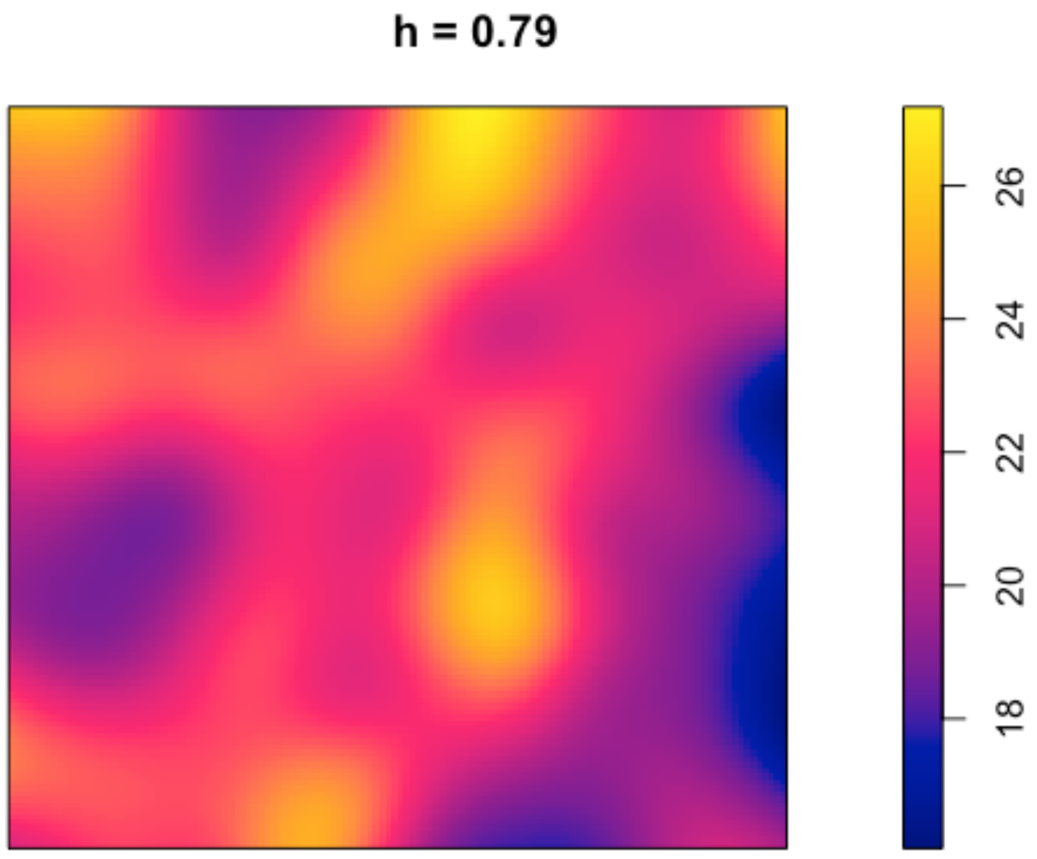
```
plot(density.ppp(UNI.ppp, sigma = bw.ppl(UNI.ppp),edge=T),main=paste("h =",ro
und(bw.ppl(UNI.ppp),2)))
```



```
plot(density.ppp(UNI.ppp, sigma = bw.scott(UNI.ppp)[2],edge=T),main=paste("h
=",round(bw.scott(UNI.ppp)[2],2)))
```



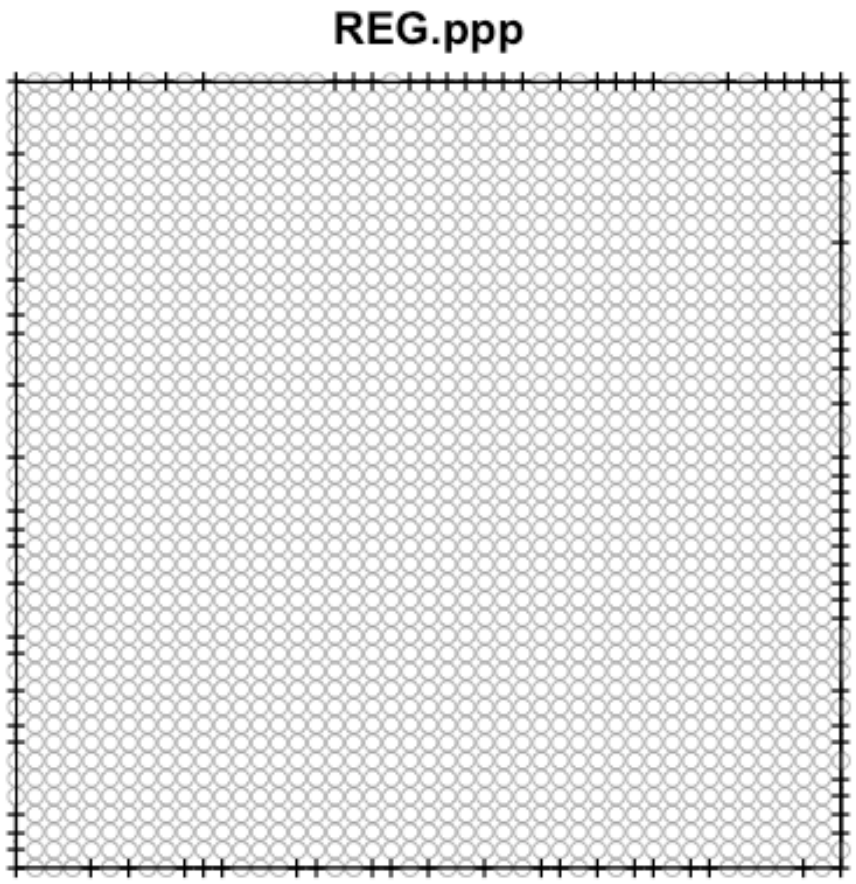
```
plot(density.ppp(UNI.ppp, sigma = bw.scott(UNI.ppp)[1],edge=T),main=paste("h
=",round(bw.scott(UNI.ppp)[1],2)))
```



What happens if we have regular points?

```
## Regular points

r.x <- seq(from=min(sp_point[,1]),to=max(sp_point[,1]),length=sqrt(nrow(sp_po
int)))
r.y <- seq(from=min(sp_point[,2]),to=max(sp_point[,2]),length=sqrt(nrow(sp_po
int)))
r.x <- jitter(rep(r.x,length(r.x)),.001)
r.y <- jitter(rep(r.y,each=length(r.y)),.001)
```

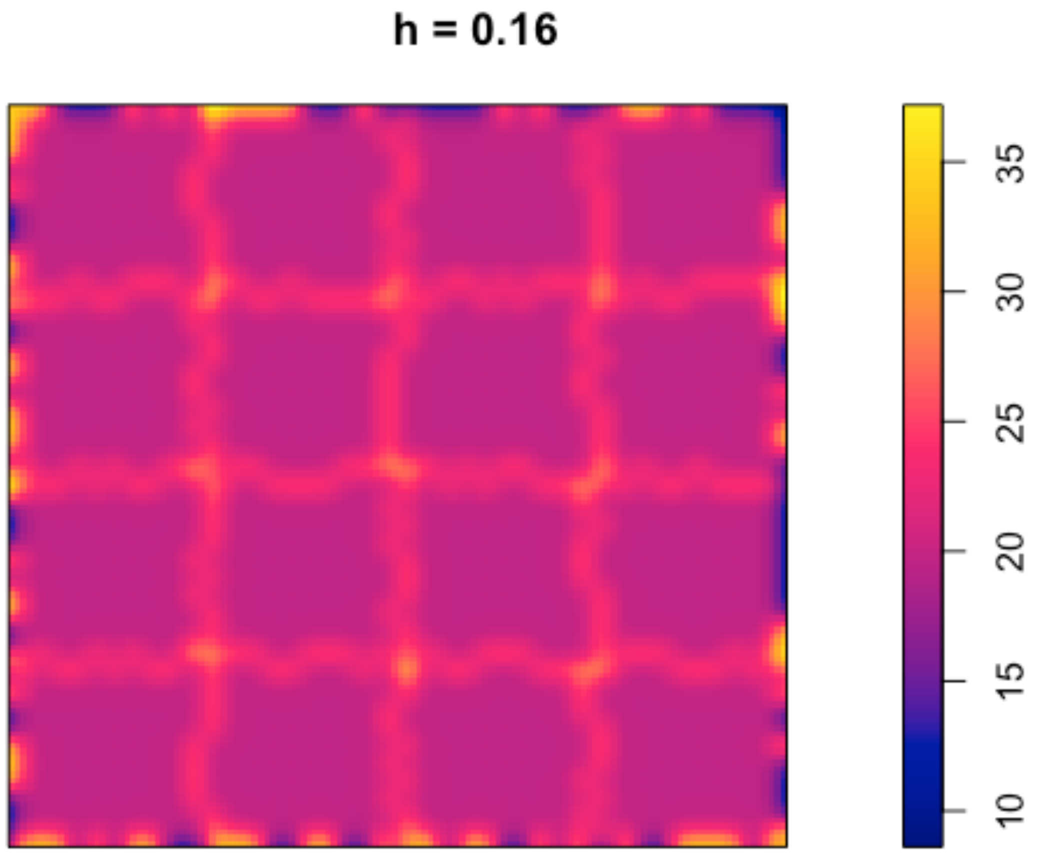
```
par( mar=c(0, 1, 1, 1.5))
REG.ppp <- ppp(x=r.x,y=r.y,window=owin(bbox(sp_point)[1,],bbox(sp_point)[2,])
)

## Warning: 97 points were rejected as lying outside the specified window

plot(REG.ppp)

## Warning: Interpretation of arguments maxsize and markscale has changed (in
## spatstat version 1.37-0 and later). Size of a circle is now measured by it
s
## diameter.

## Warning in plot.ppp(REG.ppp): 97 illegal points also plotted
```



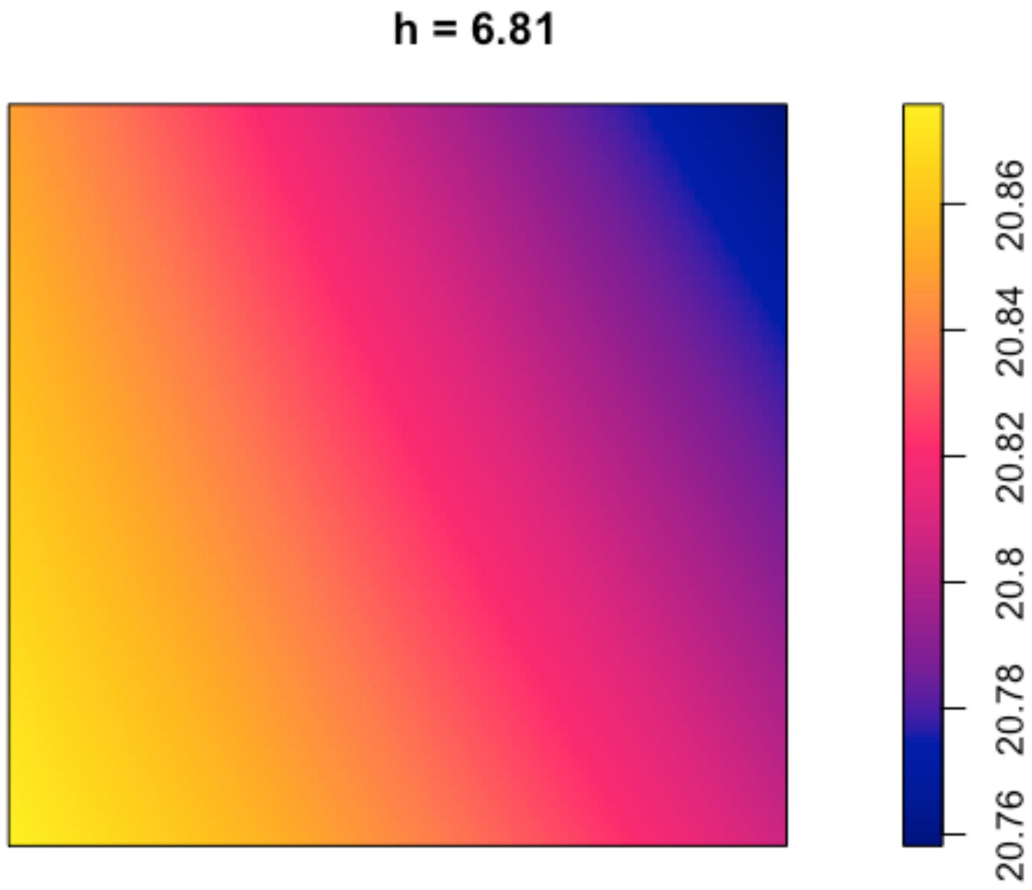
```
summary(REG.ppp)
```

```
## Planar point pattern:  1928 points
## Average intensity 20.8207 points per square unit
##
## Coordinates are given to 6 decimal places
##
## Window: rectangle = [-124.21882, -114.3603] x [32.55885, 41.95175] units
## Window area = 92.6001 square units
```

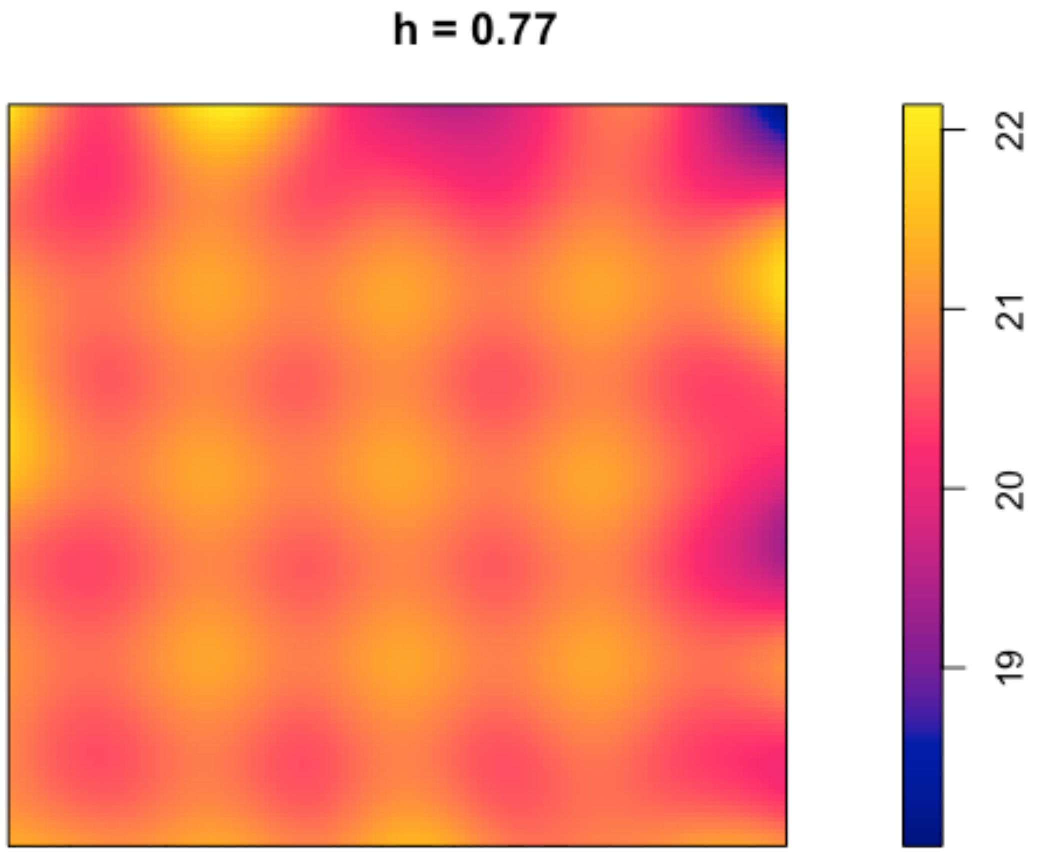
```
par( mar=c(0, 1, 1, 1.5))

plot(density.ppp(REG.ppp, sigma = bw.diggle(REG.ppp),edge=T),main=paste("h ="
,round(bw.diggle(REG.ppp),2)))
```



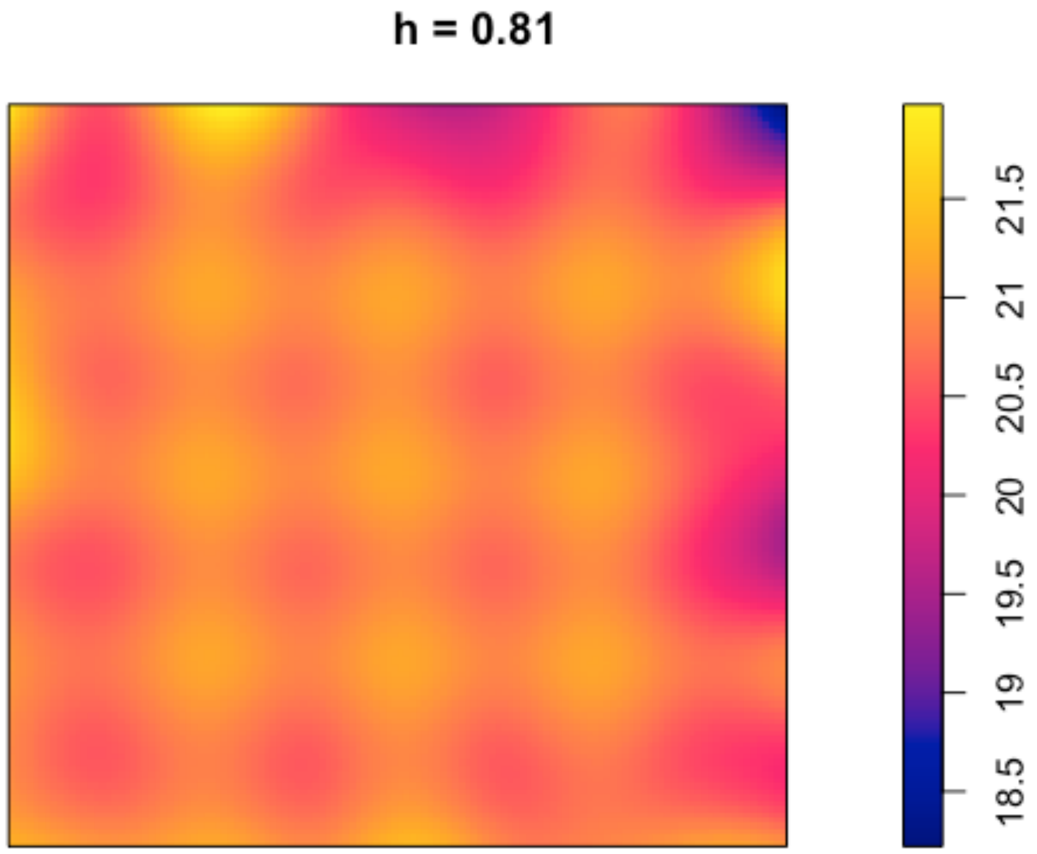
```
plot(density.ppp(REG.ppp, sigma = bw.ppl(REG.ppp),edge=T),main=paste("h =",ro
und(bw.ppl(REG.ppp),2)))
```



```r
plot(density.ppp(REG.ppp, sigma = bw.scott(REG.ppp)[2],edge=T),main=paste("h
=",round(bw.scott(REG.ppp)[2],2)))
```



```
plot(density.ppp(REG.ppp, sigma = bw.scott(REG.ppp)[1],edge=T),main=paste("h
=",round(bw.scott(REG.ppp)[1],2)))
```


h = 0.81
21.5
21
20.5
20
19.5
19
18.5