Point Pattern Analysis

HES 505 Fall 2024: Session 17

Carolyn Koehn

Objectives

- Define a point process and their utility for ecological applications
- Define first and second-order Complete Spatial Randomness
- Use several common functions to explore point patterns
- Leverage point patterns to interpolate missing data

What is a point pattern?

- *Point pattern*: A **set** of **events** within a study region (i.e., a *window*) generated by a random process
- **Set**: A collection of mathematical **events**
- **Events**: The existence of a point object of the type we are interested in at a particular location in the study region
- A *marked point pattern* refers to a point pattern where the events have additional descriptors

Some notation:

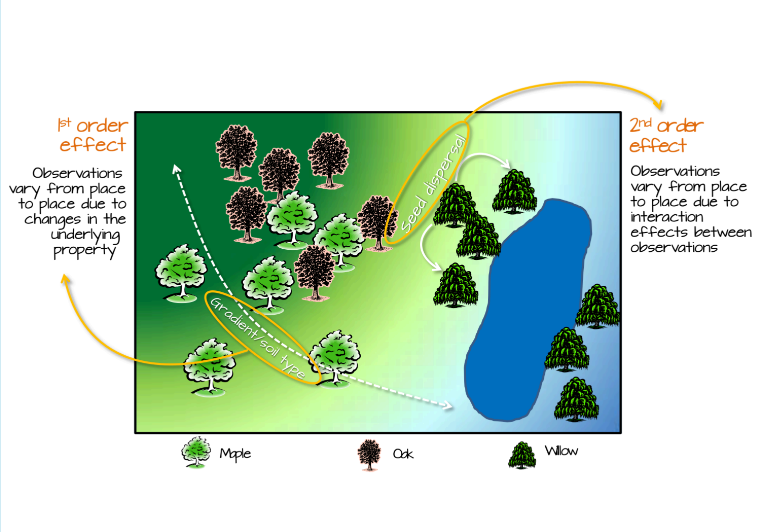
- S: refers to the entire set
- s_i refers to one event (point) in set S
- $\mathbf{s_i}$ denotes the vector of data describing point s_i in set S
- $\#(S \in A)$ refers to the number of points in S within study area A

Requirements for a set to be considered a point pattern

- The pattern must be mapped on a plane to preserve distance
- The study area, A, should be objectively determined
- There should be a $1:1$ correspondence between objects in A and events in the pattern (no undetected points)
- Events must be *proper* i.e., refer to actual locations of the event

Describing Point Patterns

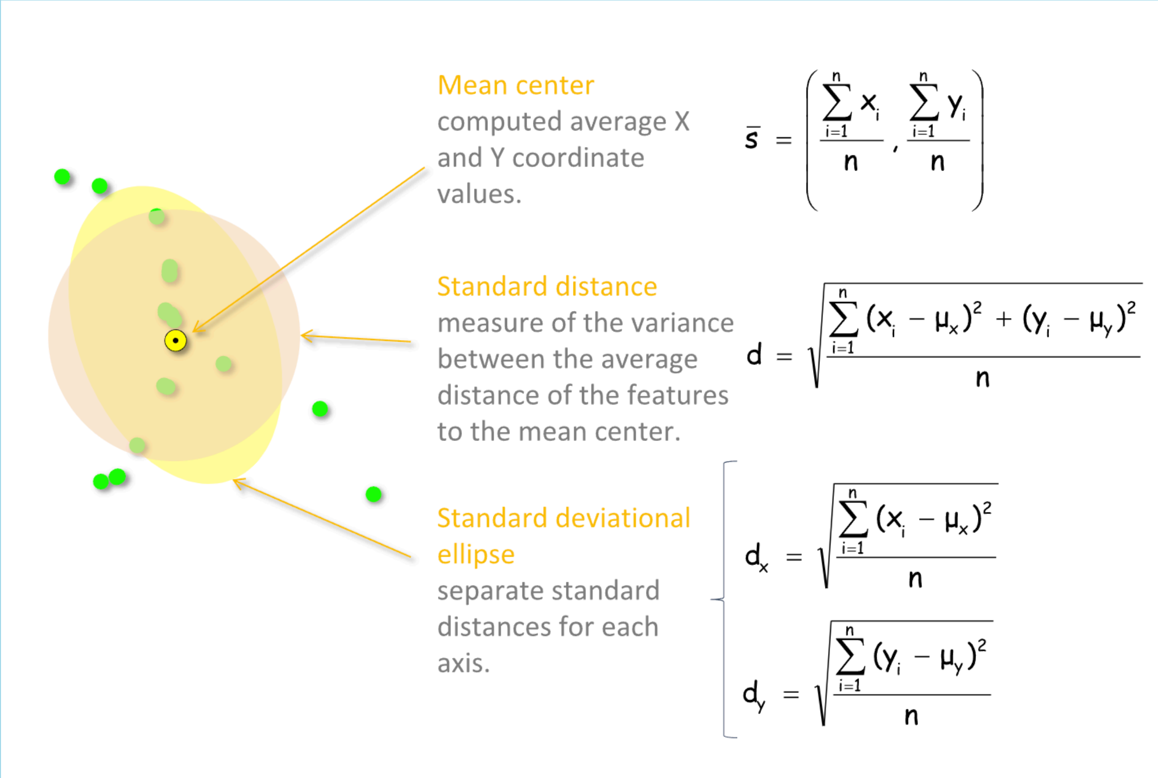
- *Density-based metrics*: the # of points within area, a, in study area A
- *Distance-based metrics*: based on nearest neighbor distances or the distance matrix for all points
- *First order* effects reflect variation in **intensity** due to variation in the ‘attractiveness’ of locations
- *Second order* effects reflect variation in **intensity** due to the presence of points themselves



from Manuel Gimond

Centrography

- *Mean center*: the point, $\hat{\mathbf{s}}$, whose coordinates are the average of all events in the pattern
- *Standard distance*: a measure of the dispersion of points around the *mean center*
- *Standard ellipse*: dispersion in one dimension



From Manuel Gimond

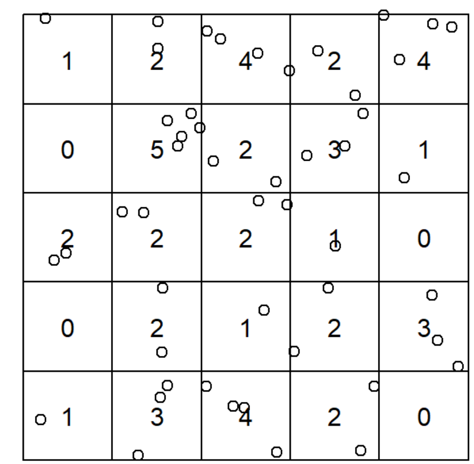
Analyzing Point Patterns

- Modeling random processes means we are interested in probability densities of the points (first-order;density)
- Also interested in how the presence of some events affects the probability of other events (second-order;distance)
- Finally interested in how the attributes of an event affect location (marked)
- Need to introduce a few new packages (`spatstat` and `gstat`)

Density based methods

- The overall *intensity* of a point pattern is a crude density estimate

$$\hat{\lambda} = \frac{\#(S \in A)}{a}$$



* Local density = quadrat counts

Analyzing Point Patterns

Kernel Density Estimates (KDE)

$$\hat{f}(x) = \frac{1}{nh_x h_y} \sum_{i=1}^{n} k\left(\frac{x - x_i}{h_x}, \frac{y - y_i}{h_y} \right)$$

- Assume each location in $\mathbf{s_i}$ drawn from unknown distribution
- Distribution has probability density $f(\mathbf{x})$
- Estimate $f(\mathbf{x})$ by averaging probability "bumps" around each location

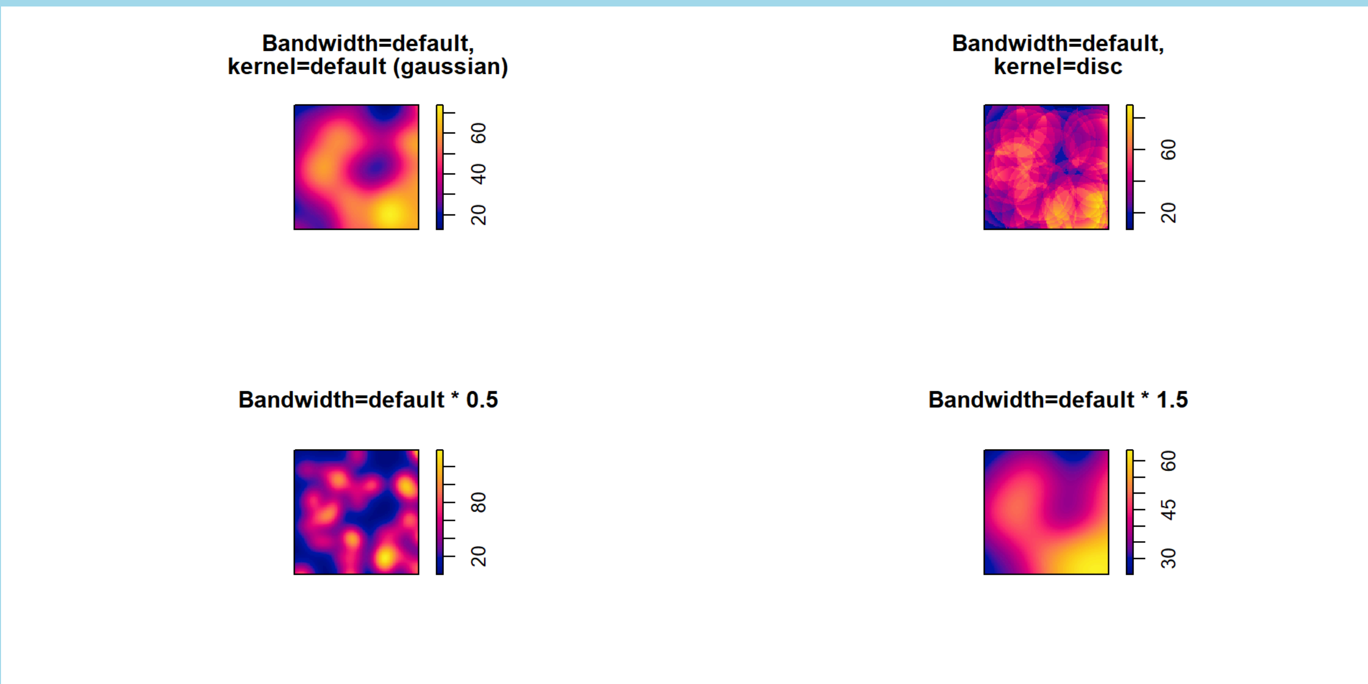
Kernel Density Estimates (KDE)

- h is the bandwidth and k is the kernel
- We can use `stats::density` to explore
- **kernel**: defines the shape, size, and weight assigned to observations in the window
- **bandwidth** often assigned based on distance from the window center

Kernel Density Estimates in Action

```r
library(spatstat)

x <- rpoispp(lambda =50)
K0 <- density(x)
K1 <- density(x, adjust=0.5)
K2 <- density(x, adjust=1.5)
K3 <- density(x, kernel="disc")
```



Choosing bandwidths and kernels

- Small values for h give 'spiky' densities
- Large values for h smooth much more
- Some kernels have optimal bandwidth detection
- `tmap` package provides additional functionality

Second-Order Analysis

Second-Order Analysis

- KDEs assume independence of points (first order randomness)
- Second-order methods allow dependence among points (second-order randomness)
- Several functions for assessing second order dependence (K, L, and G)

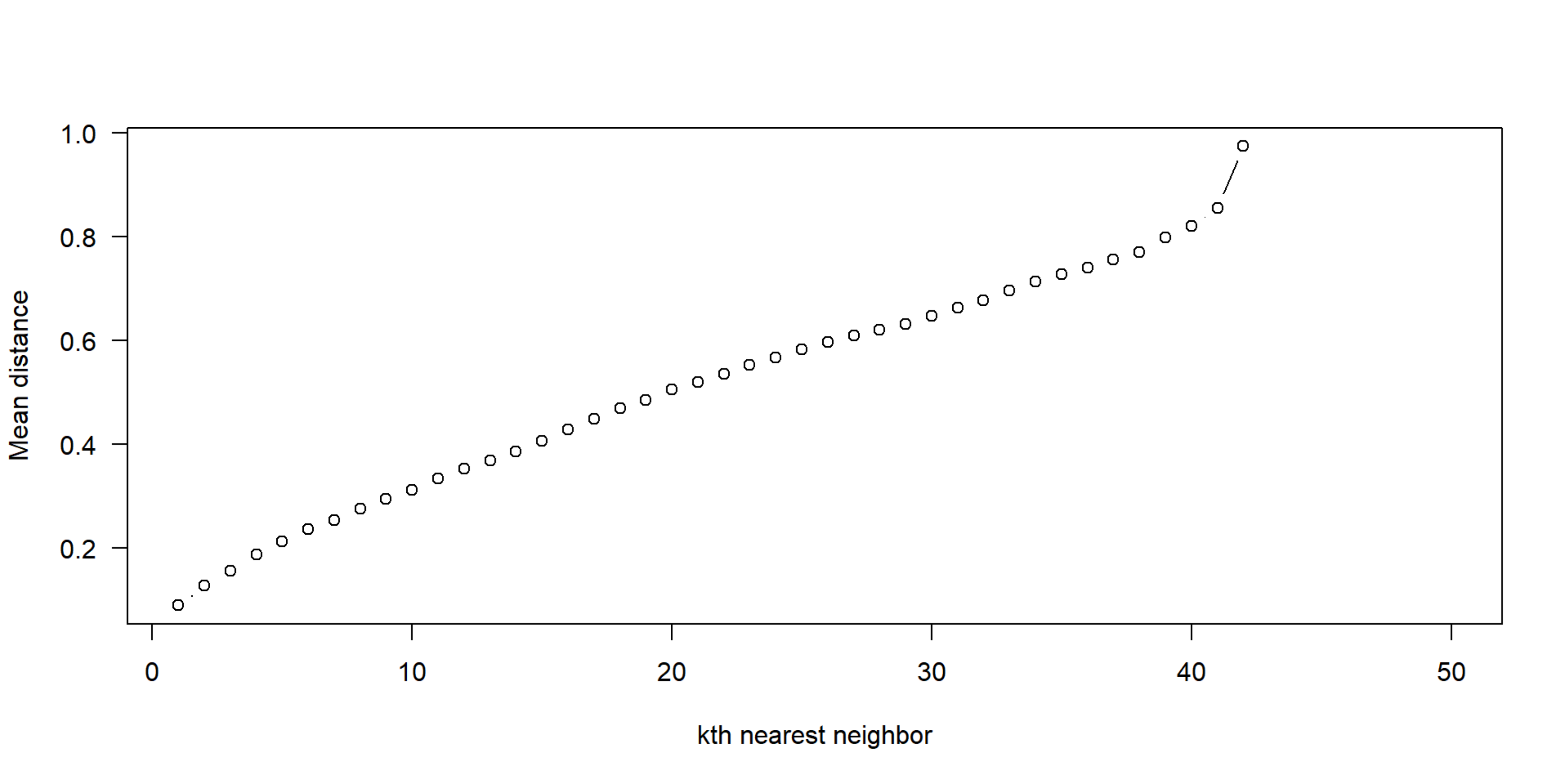
Distance based metrics

- Provide an estimate of the *second order* effects
- *Mean nearest-neighbor distance*:

$$\hat{d}_{min} = \frac{\sum_{i=1}^{m} d_{min}(\mathbf{s_i})}{n}$$

Nearest-neighbor distance

```
ANN <- apply(nndist(x, k=1:50),2,FUN=mean)
plot(ANN ~ eval(1:50), type="b", main=NULL, las=1,
     xlab="kth nearest neighbor", ylab="Mean distance")
```



Ripley's K Function

- Nearest neighbor methods throw away a lot of information
- If points have independent, fixed marginal densities, then they exhibit *complete, spatial randomness* (CSR)
- The *K* function is an alternative, based on a series of circles with increasing radius

$$K(d) = \lambda^{-1} E(N_d)$$

- We can test for clustering by comparing to the expectation:

$$K_{CSR}(d) = \pi d^2$$

- if $k(d) > K_{CSR}(d)$ then there is clustering at the scale defined by d

Ripley's K Function

- When working with a sample the distribution of K is unknown
- Estimate with

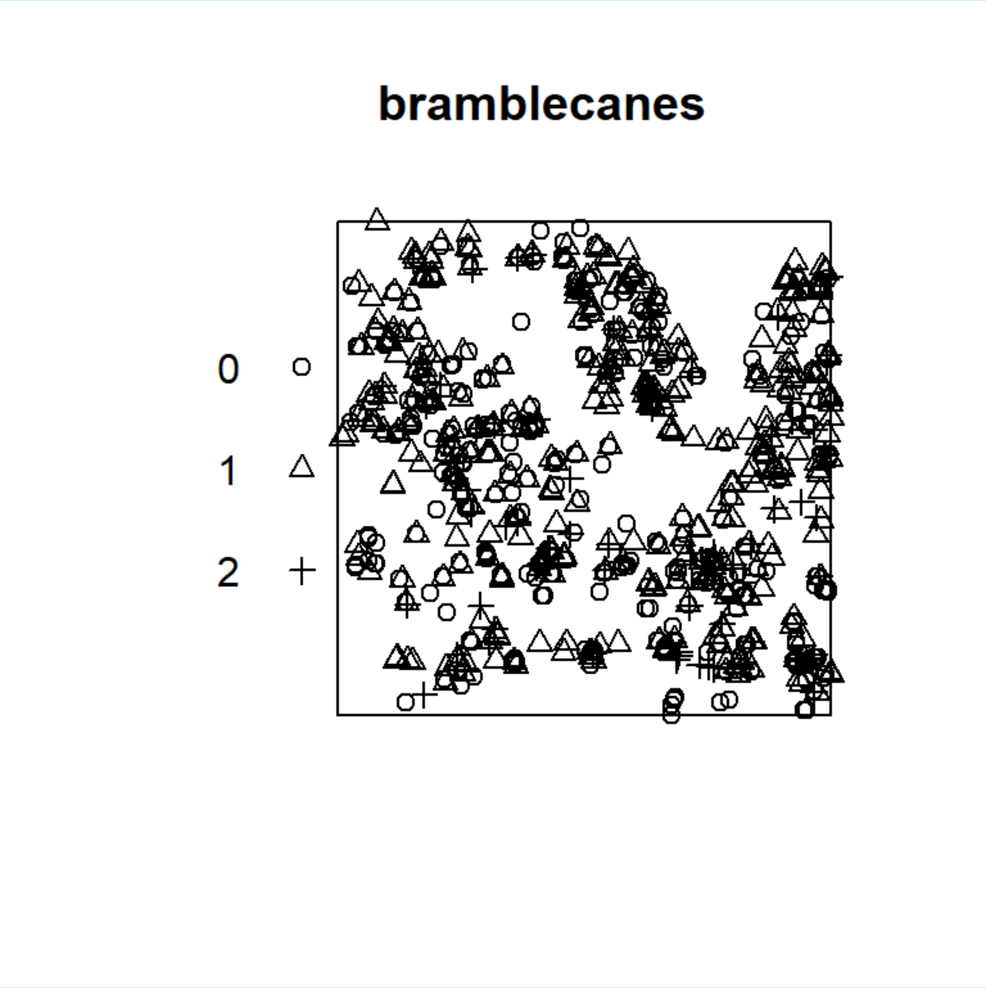
$$\hat{K}(d) = \hat{\lambda}^{-1} \sum_{i=1}^{n} \sum_{j=1}^{n} \frac{I(d_{ij} < d)}{n(n-1)}$$

where:

$$\hat{\lambda} = \frac{n}{|A|}$$

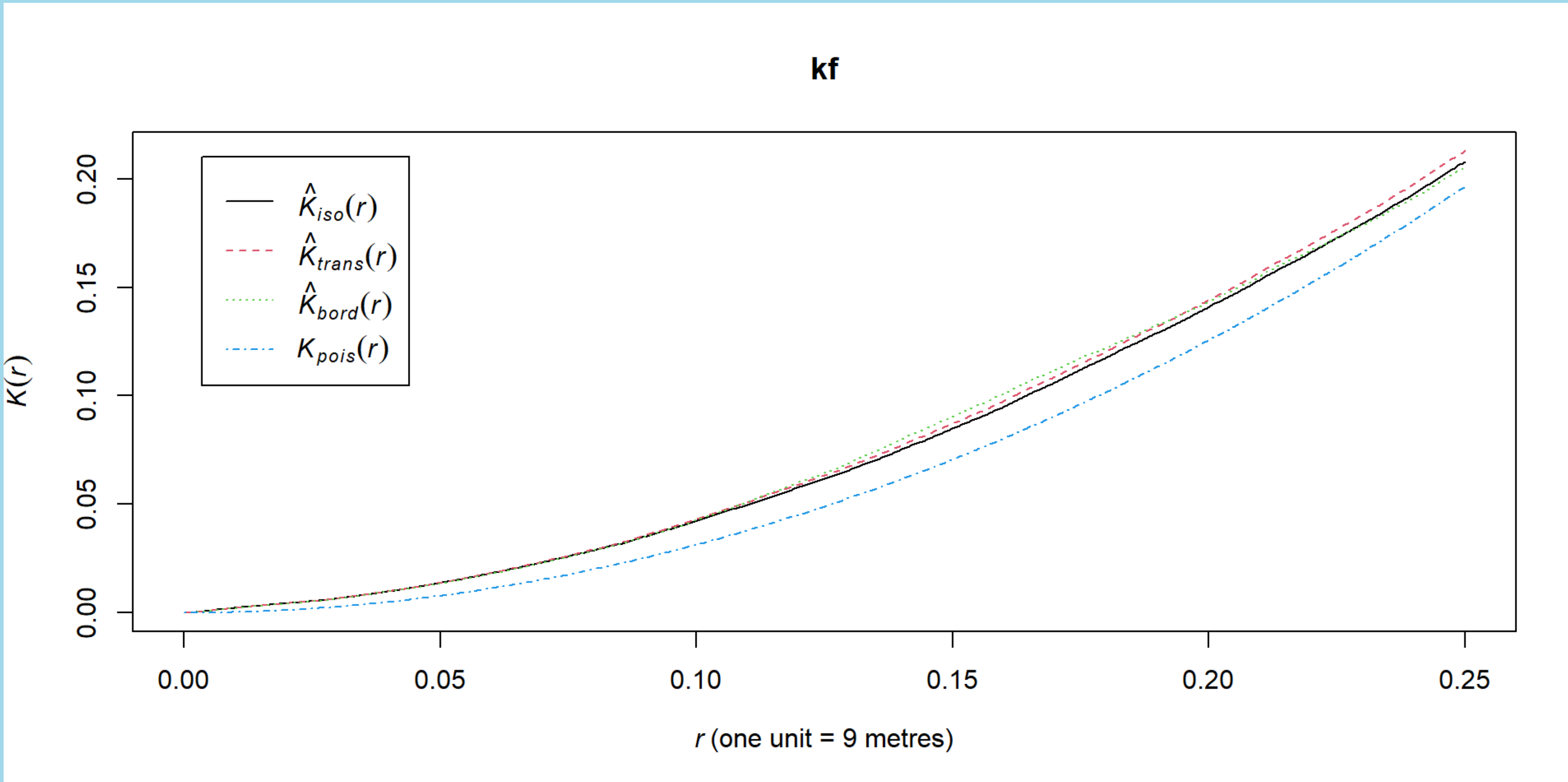
Ripley's K Function

Using the `spatstat` package:



Ripley's K Function

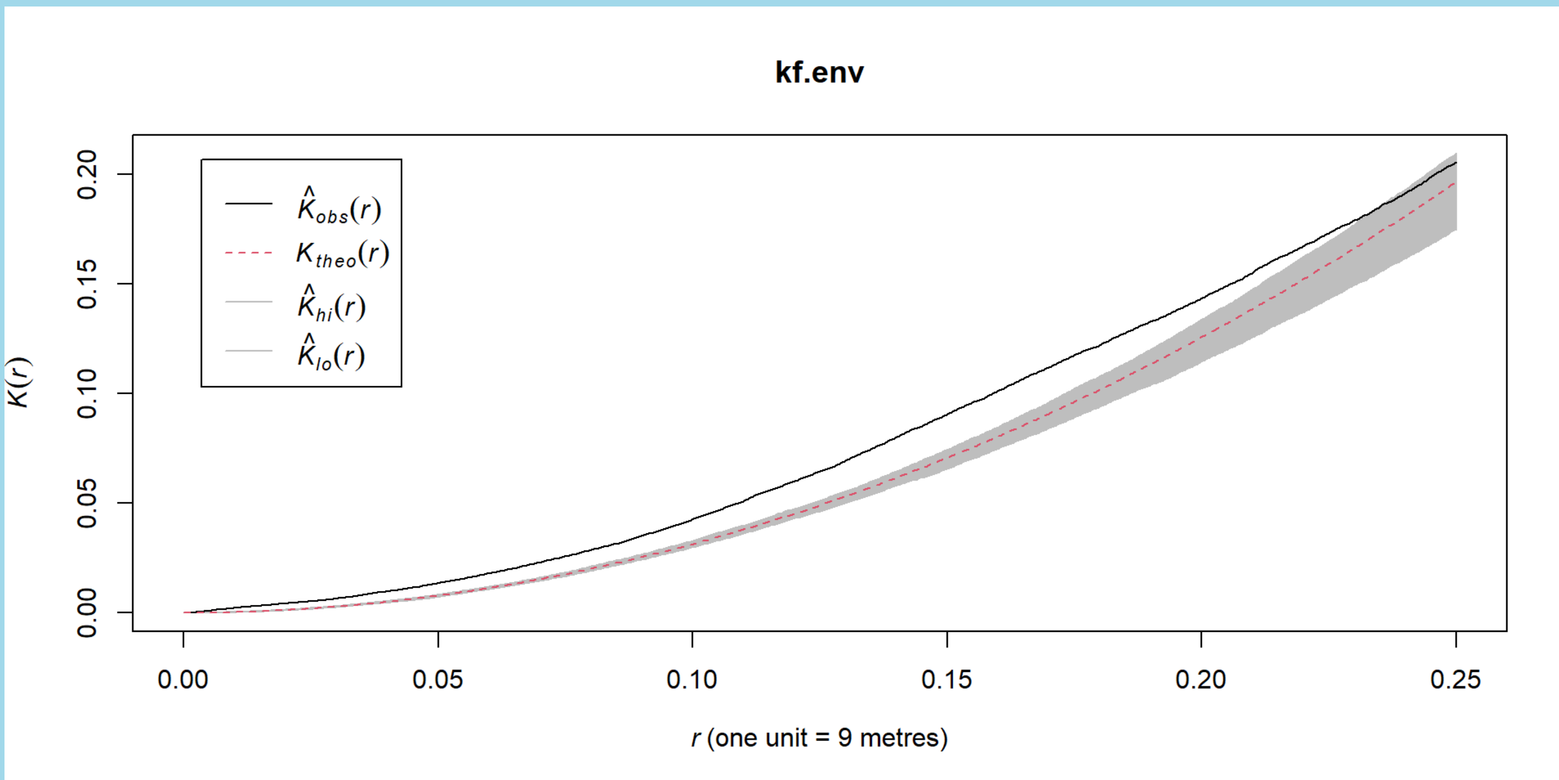
```r
kf <- Kest(bramblecanes, correction-"border")
plot(kf)
```



Ripley's K Function

- accounting for variation in d

```r
kf.env <- envelope(bramblecanes, correction="border", envelope = FALSE, ver
plot(kf.env)
```



Other functions

- L function: square root transformation of K
- G function: the cummulative frequency distribution of the nearest neighbor distances
- F function: similar to G but based on randomly located points