Hands-on Exercise 4: Spatial Point Patterns Analysis-spatstat methods

In this hands-on exercise, you will gain hands-on experience on using appropriate functions of spatstat package to perform spatial point patterns analysis.

Contents

Overview

Spatial Point Pattern Analysis is the evaluation of the pattern or distribution, of a set of points on a surface. The point can be location of:

- events such as crime, traffic accident and disease onset, or
- business services (coffee and fastfood outlets) or facilities such as childcare and eldercare.

Using appropriate functions of spatstat, this hands-on exercise aims to discover the spatial point processes of childecare centres in Singapore.

The specific questions we would like to answer are as follows:

- are the childcare centres in Singapore randomly distributed throughout the country?
- if the answer is not, then the next logical question is where are the locations with higher concentration of childcare centres?

The data

To provide answers to the questions above, three data sets will be used. They are:

- `CHILDCARE`, a point feature data providing both location and attribute information of childcare centres. It was downloaded from Data.gov.sg and is in geojson format.
- `MP14_SUBZONE_WEB_PL`, a polygon feature data providing information of URA 2014 Master Plan Planning Subzone boundary data. It is in ESRI shapefile format. This data set was also downloaded from Data.gov.sg.
- `CostalOutline`, a polygon feature data showing the national boundary of Singapore. It is provided by SLA and is in ESRI shapefile format.

Installing and Loading the R packages

In this hands-on exercise, five R packages will be used, they are:

- **sf**, a relatively new R package specially designed to import, manage and process vector-based geospatial data in R.
- **spatstat**, which has a wide range of useful functions for point pattern analysis. In this hands-on exercise, it will be used to perform 1st- and 2nd-order spatial point patterns analysis and derive kernel density estimation (KDE) layer.
- **raster** which reads, writes, manipulates, analyses and model of gridded spatial data (i.e. raster). In this hands-on exercise, it will be used to convert image output generate by spatstat into raster format.

- **maptools** which provides a set of tools for manipulating geographic data. In this hands-on exercise, we mainly use it to convert *Spatial* objects into *ppp* format of **spatstat**.
- **tmap** which provides functions for plotting cartographic quality static point patterns maps or interactive maps by using leaflet API.

Use the code chunk below to install and launch the five R packages.

```
packages = c('maptools', 'sf', 'raster','spatstat', 'tmap')
)
for (p in packages ) {
if (!require(p, character.only = T
)) {
install.packages(p)
}
library(p,character.only = T)
}
```

Spatial Data Wrangling

Importing the spatial data

In this section, *st_read()* of **sf** package will be used to import these three geospatial data sets into R.

```
childcare_sf <- st_read("data/child-care-services-geojson.geojson")
%>%
  st_transform(crs = 3414)
```

```
Reading layer `child-care-services-geojson' from data source
  `D:\tskam\IS415\Hands-on_Ex\Hands-on_Ex04\data\child-care-services-geojson.geojson'
  using driver `GeoJSON'
Simple feature collection with 1545 features and 2 fields
Geometry type: POINT
Dimension:     XYZ
Bounding box:  xmin: 103.6824 ymin: 1.248403 xmax: 103.9897 ymax: 1.462134
z_range:       zmin: 0 zmax: 0
Geodetic CRS:  WGS 84
```

```
sg_sf <- st_read(dsn = "data", layer="CostalOutline"
)
```

```
Reading layer `CostalOutline' from data source
  `D:\tskam\IS415\Hands-on_Ex\Hands-on_Ex04\data' using driver `ESRI Shapefile'
Simple feature collection with 60 features and 4 fields
Geometry type: POLYGON
Dimension:     XY
Bounding box:  xmin: 2663.926 ymin: 16357.98 xmax: 56047.79 ymax: 50244.03
```

```
Bounding box:  xmin: 2663.926 ymin: 16357.98 xmax: 56047.79 ymax: 50244.03
Projected CRS: SVY21
```

```r
mpsz_sf <- st_read(dsn = "data",
                   layer = "MP14_SUBZONE_WEB_PL")
```

```
Reading layer `MP14_SUBZONE_WEB_PL' from data source
  `D:\tskam\IS415\Hands-on_Ex\Hands-on_Ex04\data' using driver `ESRI Shapefile'
Simple feature collection with 323 features and 15 fields
Geometry type: MULTIPOLYGON
Dimension:     XY
Bounding box:  xmin: 2667.538 ymin: 15748.72 xmax: 56396.44 ymax: 50256.33
Projected CRS: SVY21
```

Before we can use these data for analysis, it is important for us to ensure that they are projected in same projection system.

> *DIY: Using the appropriate **sf** function you learned in Hands-on Exercise 2, retrieve the referencing system information of these geospatial data.*

Notice that except `childcare_sf`, both `mpsz_sf` and `sg_sf` do not have proper crs information.

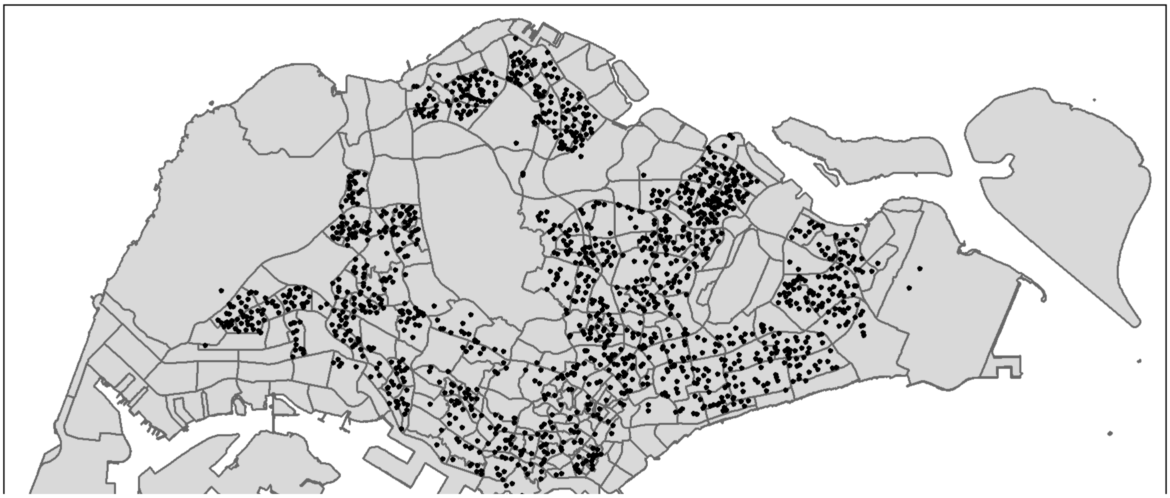
> *DIY: Using the method you learned in Lesson 2, assign the correct crs to mpsz_sf and sg_sf simple feature data frames.*

> *DIY: If necessary, changing the referencing system to Singapore national projected coordinate system.*

Mapping the geospatial data sets

After checking the referencing system of each geospatial data data frame, it is also useful for us to plot a map to show their spatial patterns.

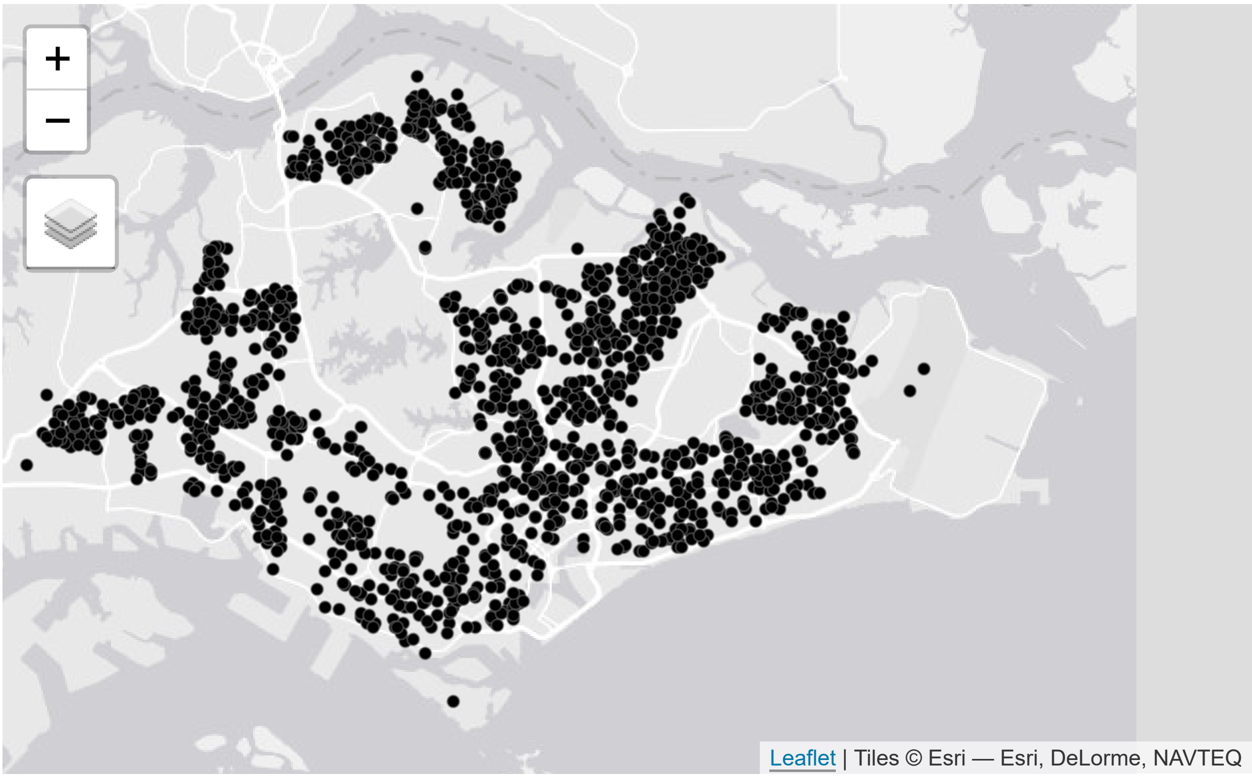
> *DIY: Using the mapping methods you learned in Hands-on Exercise 3, prepare a map as shown below.*



Notice that all the geospatial layers are within the same map extend. This shows that their referencing system and coordinate values are referred to similar spatial context. This is very important in any geospatial analysis.

Alternatively, we can also prepare a pin map by using the code chunk below.

```
tmap_mode('view')
tm_shape(childcare_sf) +
  tm_dots()
```




```
tmap_mode('plot')
```

Notice that at the interactive mode, **tmap** is using **leaflet for R** API. The advantage of this interactive pin map is it allows us to navigate and zoom around the map freely. We can also query the information of each simple feature (i.e. the point) by clicking of them. Last but not least, you can also change the background of the internet map layer. Currently, three internet map layers are provided. They are: ESRI.WorldGrayCanvas, OpenStreetMap, and ESRI.WorldTopoMap. The default is ESRI.WorldGrayCanvas.

> *Reminder: Always remember to switch back to plot mode after the interactive map. This is because, each interactive mode will consume a connection. You should also avoid displaying ecessive numbers of*

> *interactive maps (i.e. not more than 10) in one RMarkdown document when publish on Netlify.*

Geospatial Data wrangling

Although simple feature data frame is gaining popularity again sp's Spatial* classes, there are, however, many geospatial analysis packages require the input geospatial data in sp's Spatial* classes. In this section, you will learn how to convert simple feature data frame to sp's Spatial* class.

Converting sf data frames to sp's Spatial* class

The code chunk below uses *as_Spatial()* of **sf** package to convert the three geospatial data from simple feature data frame to sp's Spatial* class.

```
childcare <-        as_Spatial(        childcare_sf)
mpsz      <-        as_Spatial(        mpsz_sf   )
sg        <-        as_Spatial(        sg_sf     )
```

> *DIY: Using appropriate function, display the information of these three Spatial* classes as shown below.*

```
childcare
```

```
class       : SpatialPointsDataFrame
features    : 1545
extent      : 11203.01, 45404.24, 25667.6, 49300.88  (xmin, xmax, ymin, ymax)
crs         : +proj=tmerc +lat_0=1.36666666666667 +lon_0=103.833333333333 +k=1 +x_0=28001.642 +y_0=38
variables   : 2
names       :    Name,
min values  :   kml_1, <center><table><tr><th colspan='2' align='center'><em>Attributes</em></th></tr
max values  : kml_999,                  <center><table><tr><th colspan='2' align='center'><em>Attribu
```

```
mpsz
```

```
class       : SpatialPolygonsDataFrame
features    : 323
extent      : 2667.538, 56396.44, 15748.72, 50256.33  (xmin, xmax, ymin, ymax)
crs         : +proj=tmerc +lat_0=1.36666666666667 +lon_0=103.833333333333 +k=1 +x_0=28001.642 +y_0=38
variables   : 15
names       : OBJECTID, SUBZONE_NO, SUBZONE_N, SUBZONE_C, CA_IND, PLN_AREA_N, PLN_AREA_C,       REGIO
min values  :        1,          1, ADMIRALTY,    AMSZ01,      N, ANG MO KIO,         AM, CENTRAL REG
```

```
max values  :      323,         17,    YUNNAN,    YSSZ09,      Y,     YISHUN,         YS,    WEST REG
```

```
sg
```

```
class       : SpatialPolygonsDataFrame
features    : 60
extent      : 2663.926, 56047.79, 16357.98, 50244.03  (xmin, xmax, ymin, ymax)
crs         : +proj=tmerc +lat_0=1.36666666666667 +lon_0=103.833333333333 +k=1 +x_0=28001.642 +y_0=38
variables   : 4
names       : GDO_GID, MSLINK, MAPID,              COSTAL_NAM
min values  :       1,      1,     0,             ISLAND LINK
max values  :      60,     67,     0, SINGAPORE - MAIN ISLAND
```

Notice that the geospatial data have been converted into their respective sp's Spatial* classes now.

Converting the Spatial* class into generic sp format

spatstat requires the analytical data in ***ppp*** object form. There is no direct way to convert a Spatial* classes into ***ppp*** object. We need to convert the ***Spatial* classes*** into ***Spatial*** object first.

The codes chunk below converts the Spatial* classes into generic sp objects.

```
childcare_sp <- as(childcare, "SpatialPoints")
sg_sp <- as(sg, "SpatialPolygons")
```

Next, you should display the sp objects properties as shown below.

```
childcare_sp
```

```
class       : SpatialPoints
features    : 1545
extent      : 11203.01, 45404.24, 25667.6, 49300.88  (xmin, xmax, ymin, ymax)
crs         : +proj=tmerc +lat_0=1.36666666666667 +lon_0=103.833333333333 +k=1 +x_0=28001.642 +y_0=38
```

```
sg_sp
```

```
class       : SpatialPolygons
features    : 60
extent      : 2663.926, 56047.79, 16357.98, 50244.03  (xmin, xmax, ymin, ymax)
crs         : +proj=tmerc +lat_0=1.36666666666667 +lon_0=103.833333333333 +k=1 +x_0=28001.642 +y_0=38
```

> *Challenge: Do you know what are the differences between Spatial* classes and generic sp object?*

Converting the generic sp format into spatstat's ppp format

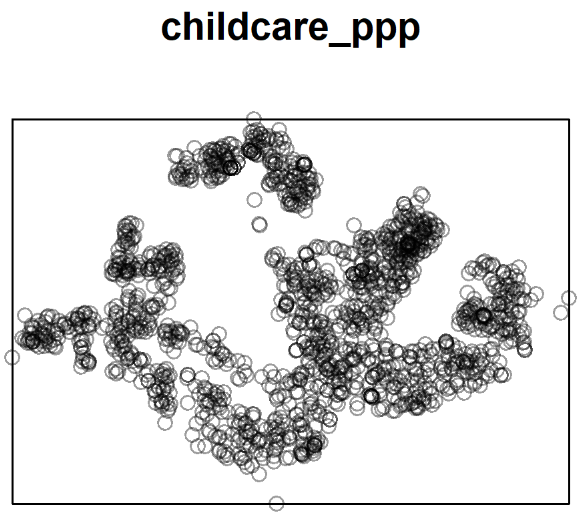
Now, we will use *as.ppp()* function of **spatstat** to convert the spatial data into **spatstat**'s ***ppp*** object format.

```
childcare_ppp <- as(childcare_sp, "ppp")
childcare_ppp
```

```
Planar point pattern: 1545 points
window: rectangle = [11203.01, 45404.24] x [25667.6, 49300.88] units
```

Now, let us plot ***childcare_ppp*** and examine the different.

```
plot(childcare_ppp)
```



You can take a quick look at the summary statistics of the newly created ppp object by using the code chunk below.

```
summary(childcare_ppp)
```

```
Planar point pattern:  1545 points
Average intensity 1.91145e-06 points per square unit

*Pattern contains duplicated points*

Coordinates are given to 3 decimal places
i.e. rounded to the nearest multiple of 0.001 units

Window: rectangle = [11203.01, 45404.24] x [25667.6, 49300.88] units
                    (34200 x 23630 units)
Window area = 808287000 square units
```

Notice the warning message about duplicates. In spatial point patterns analysis an issue of significant is the presence of duplicates. The statistical methodology used for spatial point patterns processes is based largely on the assumption that process are *simple*, that is, that the points cannot be coincident.

Handling duplicated points

We can check the duplication in a ***ppp*** object by using the code chunk below.

```
any(duplicated(childcare_ppp))
```

```
[1] TRUE
```

To count the number of co-indicence point, we will use the *multiplicity()* function as shown in the code chunk below.

```
multiplicity(childcare_ppp)
```

If we want to know how many locations have more than one point event, we can use the code chunk below.

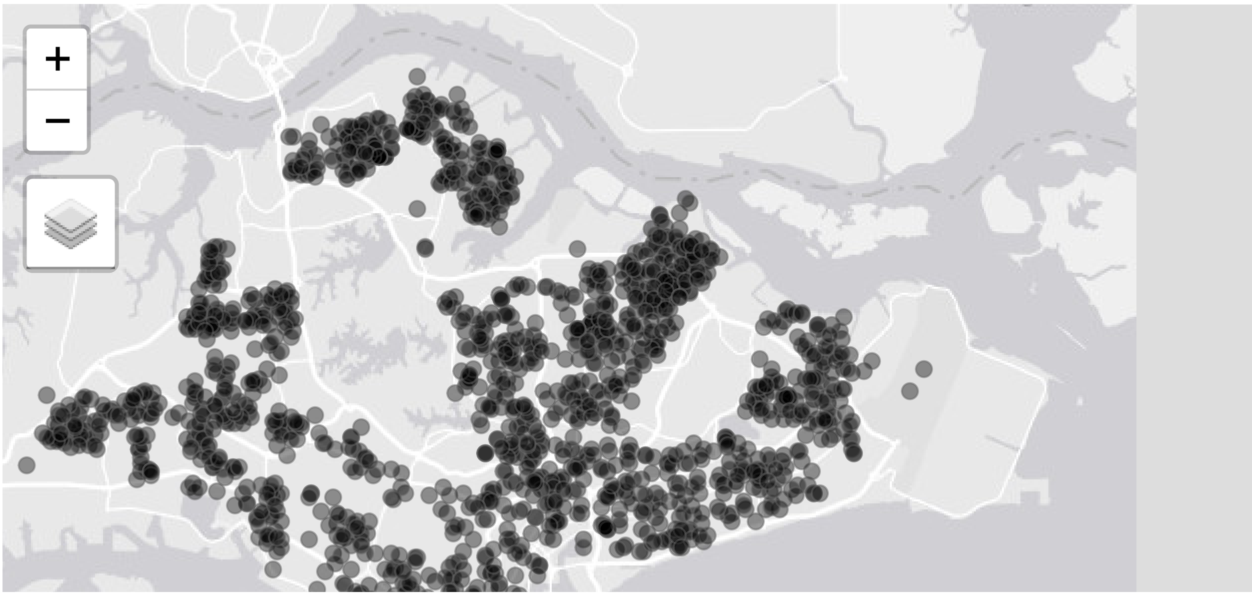
```
sum(multiplicity(childcare_ppp) > 1)
```

```
[1] 128
```

The output shows that there are 128 duplicated point events.

To view the locations of these duplicate point events, we will plot `childcare` data by using the code chunk below.

```
tmap_mode('view')
tm_shape(childcare) +
  tm_dots(alpha=0.4,
          size=0.05)
```




```
tmap_mode('plot')
```

> *Challenge: Do you know how to spot the duplicate points from the map shown above?*

There are three ways to overcome this problem. The easiest way is to delete the duplicates. But, that will also mean that some useful point events will be lost.

The second solution is use *jittering*, which will add a small perturbation to the duplicate points so that they do not occupy the exact same space.

The third solution is to make each point "unique" and then attach the duplicates of the points to the patterns as **marks**, as attributes of the points. Then you would need analytical techniques that take into account these marks.

The code chunk below implements the jittering approach.

```
childcare_ppp_jit <- rjitter(childcare_ppp,
                             retry=TRUE,
                             nsim=1,
                             drop=TRUE)
```

> *DIY: Using the method you learned in previous section, check if any dusplicated point in this geospatial data.*

```
any(duplicated(childcare_ppp_jit))
```

```
[1] FALSE
```

Creating *owin* object

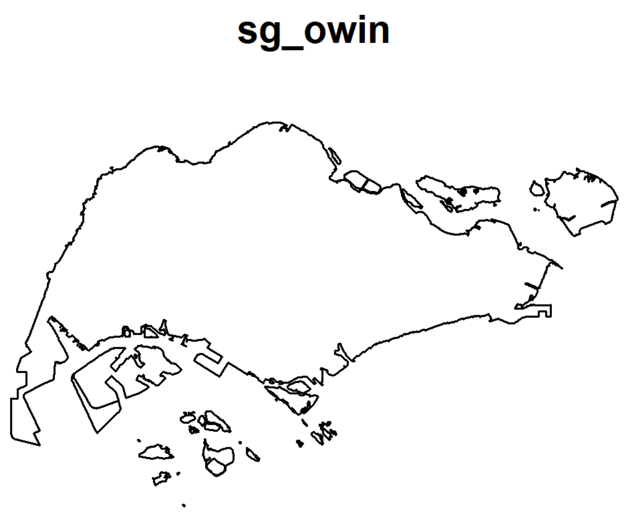
When analysing spatial point patterns, it is a good practice to confine the analysis with a geographical area like Singapore boundary. In **spatstat**, an object called ***owin*** is specially designed to represent this polygonal region.

The code chunk below is used to covert *sg* SpatialPolygon object into owin object of **spatstat**.

```
sg_owin <- as(sg_sp, "owin")
```

The ouput object can be displayed by using *plot()* function

```r
plot(sg_owin)
```



and *summary()* function of Base R.

```r
summary(sg_owin)
```

Combining point events object and owin object

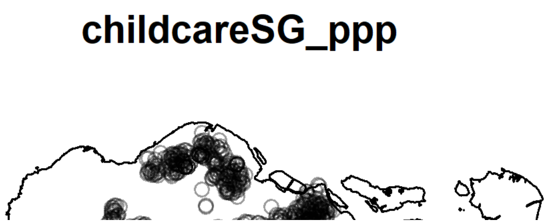
In this last step of geospatial data wrangling, we will extract childcare events that are located within Singapore by using the code chunk below.

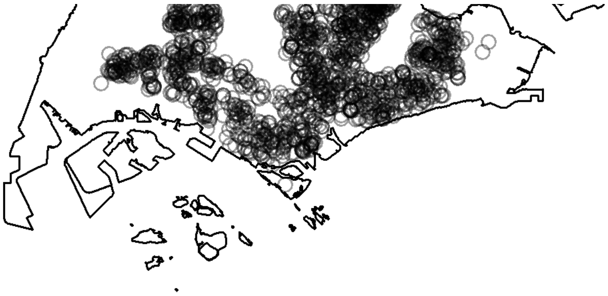
```r
childcareSG_ppp = childcare_ppp[sg_owin]
```

The output object combined both the point and polygon feature in one ppp object class as shown below.

```r
summary(childcareSG_ppp)
```

> *DIY: Using the method you learned in previous exercise, plot the newly derived childcareSG_ppp as shown below.*





First-order Spatial Point Patterns Analysis

In this section, you will learn how to perform first-order SPPA by using **spatstat** package. The hands-on exercise will focus on:

- deriving **kernel density estimation (KDE)** layer for visualising and exploring the intensity of point processes,
- performing **Confirmatory Spatial Point Patterns Analysis** by using **Nearest Neighbour** statistics.

Kernel Density Estimation

In this section, you will learn how to compute the kernel density estimation (KDE) of childcare services in Singapore.

Computing kernel density estimation using automatic bandwidth selection method

The code chunk below computes a kernel density by using the following configurations of *density()* of **spatstat**:

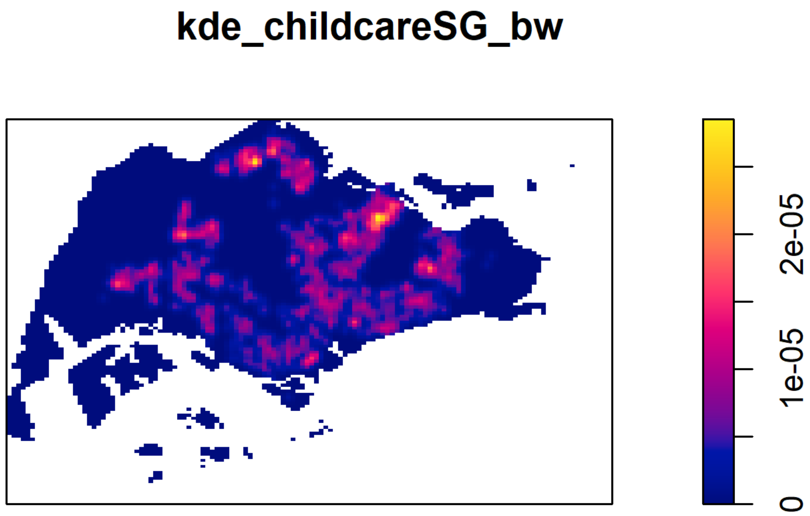
- *bw.diggle()* automatic bandwidth selection method. Other recommended methods are *bw.CvL()*, *bw.scott()* or *bw.ppl()*.
- The smoothing kernel used is *gaussian*, which is the default. Other smoothing methods are: “epanechnikov”, “quartic” or “disc”.
- The intensity estimate is corrected for edge effect bias by using method described by Jones (1993) and Diggle (2010, equation 18.9). The default is *FALSE*.

```
kde_childcareSG_bw <-        density (        childcareSG_ppp,
                              sigma=          bw.diggle
```

```
                              sigma=        bw.diggle,
                               edge=        TRUE     ,
                             kernel=        "gaussian")
```

The *plot()* function of Base R is then used to display the kernel density derived.

```
plot    (        kde_childcareSG_bw)
```



The density values of the output range from 0 to 0.000035 which is way too small to comprehend. This is because the default unit of measurement of svy21 is in meter. As a result, the density values computed is in "number of points per square meter".

Before we move on to next section, it is good to know that you can retrieve the bandwidth used to compute the kde layer by using the code chunk below.

```
bw        <-        bw.diggle(        childcareSG_ppp)
bw
```

```
   sigma 
298.4095 
```

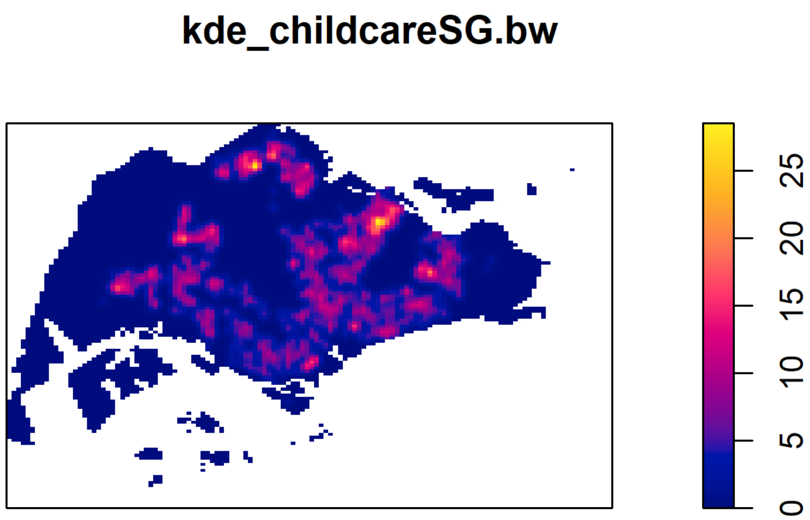
Rescalling KDE values

In the code chunk below, *rescale()* is used to covert the unit of measurement from meter to kilometer.

```
childcareSG_ppp.km <-        rescale   (        childcareSG_ppp, 1000     , "km"     )
```

Now, we can re-run *density()* using the resale data set and plot the output kde map.

```
kde_childcareSG.bw <- density(childcareSG_ppp.km, sigma=bw.diggle, edge=TRUE, kernel="gaussian")
plot(kde_childcareSG.bw)
```



Notice that output image looks identical to the earlier version, the only changes in the data values (refer to the legend).

Working with different automatic badwidth methods

Beside *bw.diggle()*, there are three other **spatstat** functions can be used to determine the bandwidth, they are: *bw.CvL()*, *bw.scott()*, and *bw.ppl()*.

Let us take a look at the bandwidth return by these automatic bandwidth calculation methods by using the code chunk below.

```
 bw.CvL(childcareSG_ppp.km)
```

```
   sigma 
4.543278 
```

```
bw.scott(childcareSG_ppp.km)
```

```
 sigma.x  sigma.y 
2.224898 1.450966 
```

```
bw.ppl(childcareSG_ppp.km)
```

```
    sigma
0.3897114
```

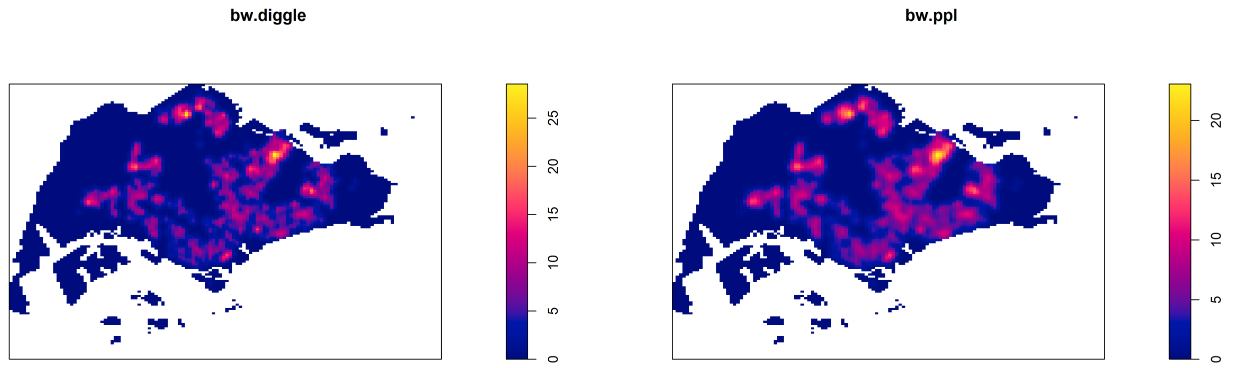
```
bw.diggle(childcareSG_ppp.km)
```

```
    sigma
0.2984095
```

Baddeley et. (2016) suggested the use of the *bw.ppl()* algorithm because in ther experience it tends to produce the more appropriate values when the pattern consists predominantly of tight clusters. But they also insist that if the purpose of once study is to detect a single tight cluster in the midst of random noise then the *bw.diggle()* method seems to work best.

The code chunk beow will be used to compare the output of using *bw.diggle* and *bw.ppl* methods.

```
kde_childcareSG.ppl <- density(childcareSG_ppp.km,
                               sigma=bw.ppl,
                               edge=TRUE,
                               kernel="gaussian")
par(mfrow=c(1,2))
plot(kde_childcareSG.bw, main = "bw.diggle")
plot(kde_childcareSG.ppl, main = "bw.ppl")
```



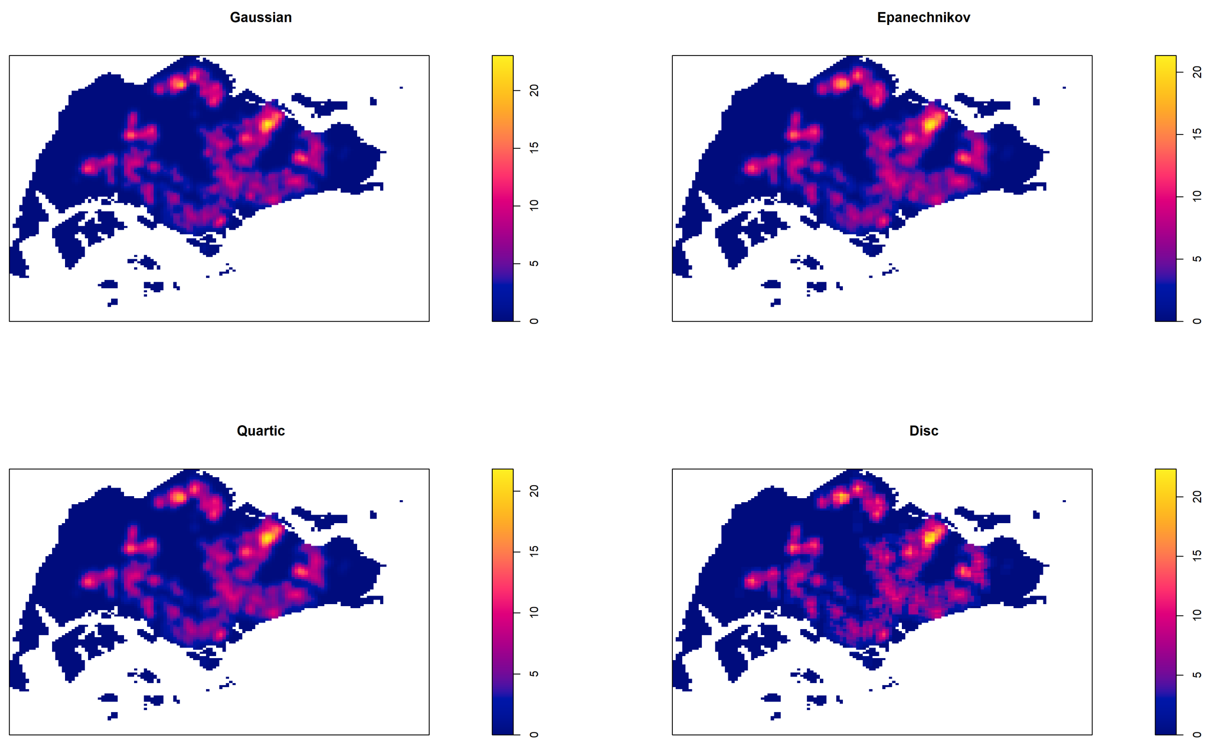
Working with different kernel methods

By default, the kernel method used in *density.ppp()* is *gaussian*. But there are three other options, namely: Epanechnikov, Quartic and Dics.

The code chunk below will be used to compute three more kernel density estimations by using these three kernel function.

```
par(mfrow=c(2,2))
plot(density(childcareSG_ppp.km,
```

```
                sigma=       bw.ppl   ,
                edge=        TRUE     ,
                kernel=        "gaussian")        ,
        main=        "Gaussian")
plot     (        density  (        childcareSG_ppp.km,
                sigma=       bw.ppl   ,
                edge=        TRUE     ,
                kernel=        "epanechnikov")        ,
        main=        "Epanechnikov")
plot     (        density  (        childcareSG_ppp.km,
                sigma=       bw.ppl   ,
                edge=        TRUE     ,
                kernel=        "quartic")        ,
        main=        "Quartic")
plot     (        density  (        childcareSG_ppp.km,
                sigma=       bw.ppl   ,
                edge=        TRUE     ,
                kernel=        "disc"   )        ,
        main=        "Disc"   )
```



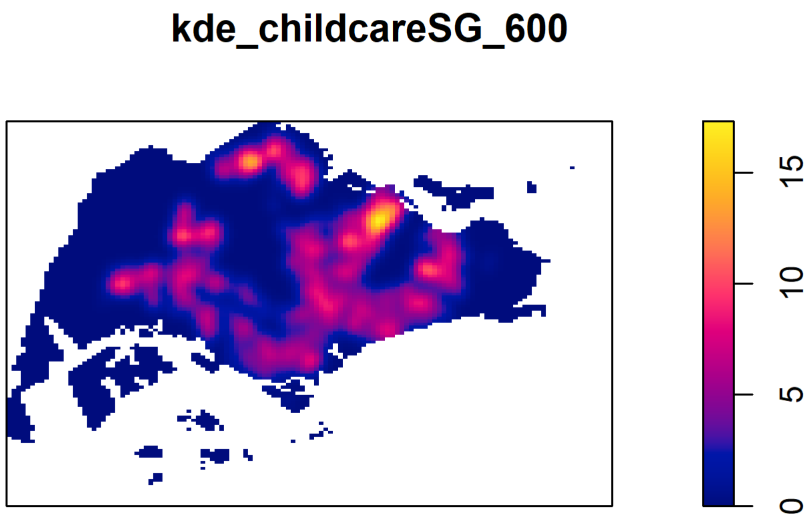
Fixed and Adaptive KDE

COMPUTING KDE BY USING FIXED BANDWIDTH

Next, you will compute a KDE layer by defining a bandwidth of 600 meter. Notice that in the code chunk below, the sigma value used is 0.6. This is because the unit of measurement of ***childcareSG_ppp.km*** object is in kilometer, hence the 600m is 0.6km.

is in kilometer, hence the 600m is 0.6km.

```
kde_childcareSG_600 <- density(childcareSG_ppp.km, sigma=0.6, edge=TRUE, kernel="gaussian")
plot(kde_childcareSG_600)
```

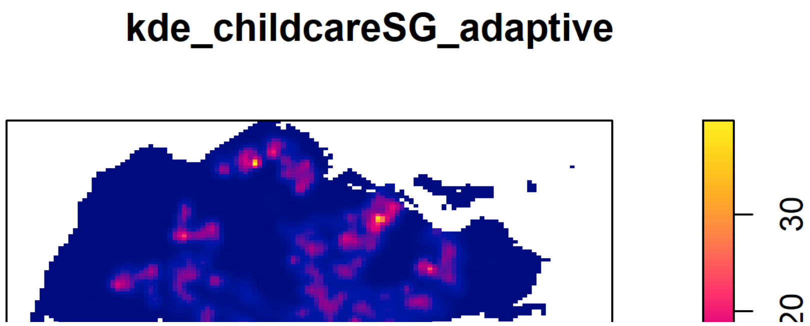


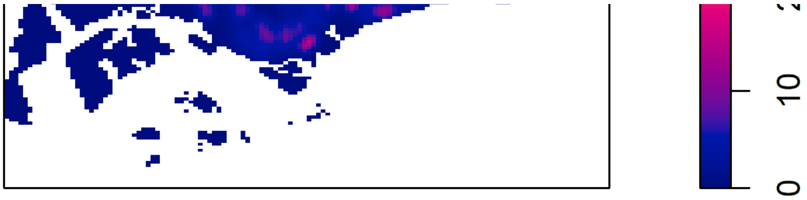
COMPUTING KDE BY USING ADAPTIVE BANDWIDTH

Fixed bandwidth method is very sensitive to highly skew distribution of spatial point patterns over geographical units for example urban versus rural. One way to overcome this problem is by using adaptive bandwidth instead.

In this section, you will learn how to derive adaptive kernel density estimation by using *density.adaptive()* of **spatstat**.

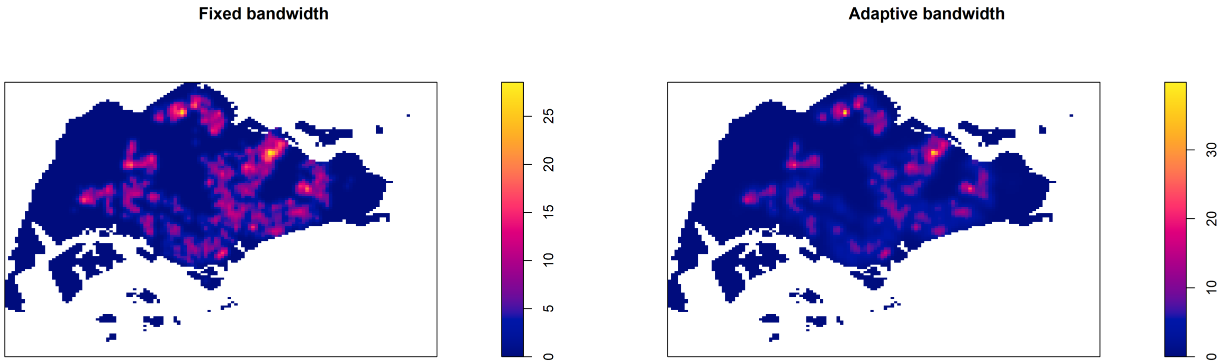
```
kde_childcareSG_adaptive <- adaptive.density(childcareSG_ppp.km, method="kernel")
plot(kde_childcareSG_adaptive)
```





We can compare the fixed and adaptive kernel density estimation outputs by using the code chunk below.

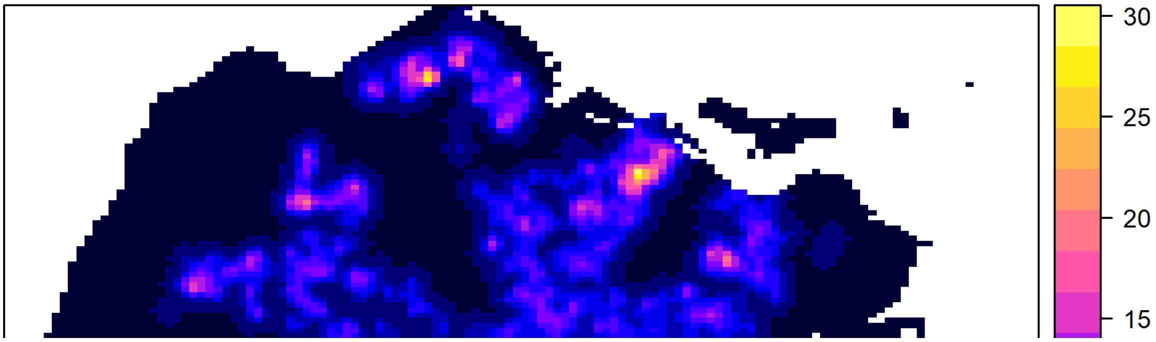
```
par(mfrow=c(1,2))
plot(kde_childcareSG.bw, main = "Fixed bandwidth")
plot(kde_childcareSG_adaptive, main = "Adaptive bandwidth")
```

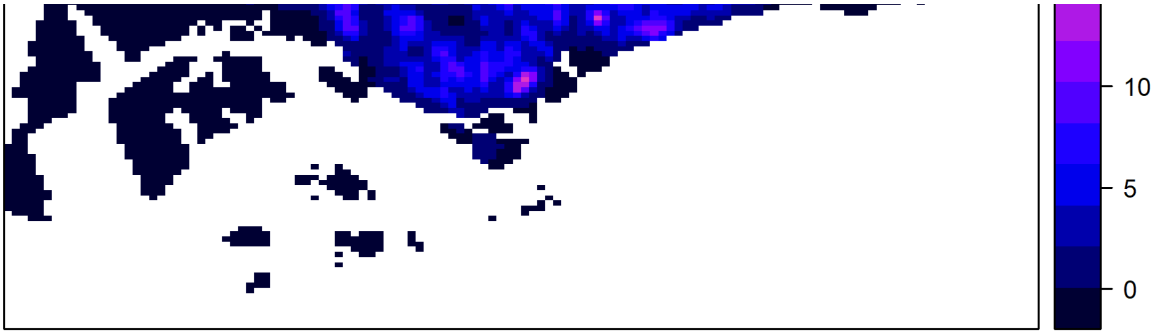


Converting KDE output into grid object.

The result is the same, we just convert it so that it is suitable for mapping purposes

```
gridded_kde_childcareSG_bw <- as.SpatialGridDataFrame.im(kde_childcareSG.bw)
spplot(gridded_kde_childcareSG_bw)
```





CONVERTING GRIDDED OUTPUT INTO RASTER

Next, we will convert the gridded kernal density objects into RasterLayer object by using *raster()* of **raster** package.

```
kde_childcareSG_bw_raster <- raster(gridded_kde_childcareSG_bw)
```

Let us take a look at the properties of *kde_childcareSG_bw_raster* RasterLayer.

```
kde_childcareSG_bw_raster
```

```
class      : RasterLayer
dimensions : 128, 128, 16384  (nrow, ncol, ncell)
resolution : 0.4170614, 0.2647348  (x, y)
extent     : 2.663926, 56.04779, 16.35798, 50.24403  (xmin, xmax, ymin, ymax)
crs        : NA
source     : memory
names      : v
values     : -8.476185e-15, 28.51831  (min, max)
```

Notice that the crs property is NA.

ASSIGNING PROJECTION SYSTEMS

The code chunk below will be used to include the CRS information on kde_childcareSG_bw_raster RasterLayer.

```
projection(kde_childcareSG_bw_raster) <- CRS("+init=EPSG:3414")
kde_childcareSG_bw_raster
```

```
class      : RasterLayer
dimensions : 128, 128, 16384  (nrow, ncol, ncell)
resolution : 0.4170614, 0.2647348  (x, y)
extent     : 2.663926, 56.04779, 16.35798, 50.24403  (xmin, xmax, ymin, ymax)
crs        : +proj=tmerc +lat_0=1.36666666666667 +lon_0=103.833333333333 +k=1 +x_0=28001.642 +y_0=38
source     : memory
```

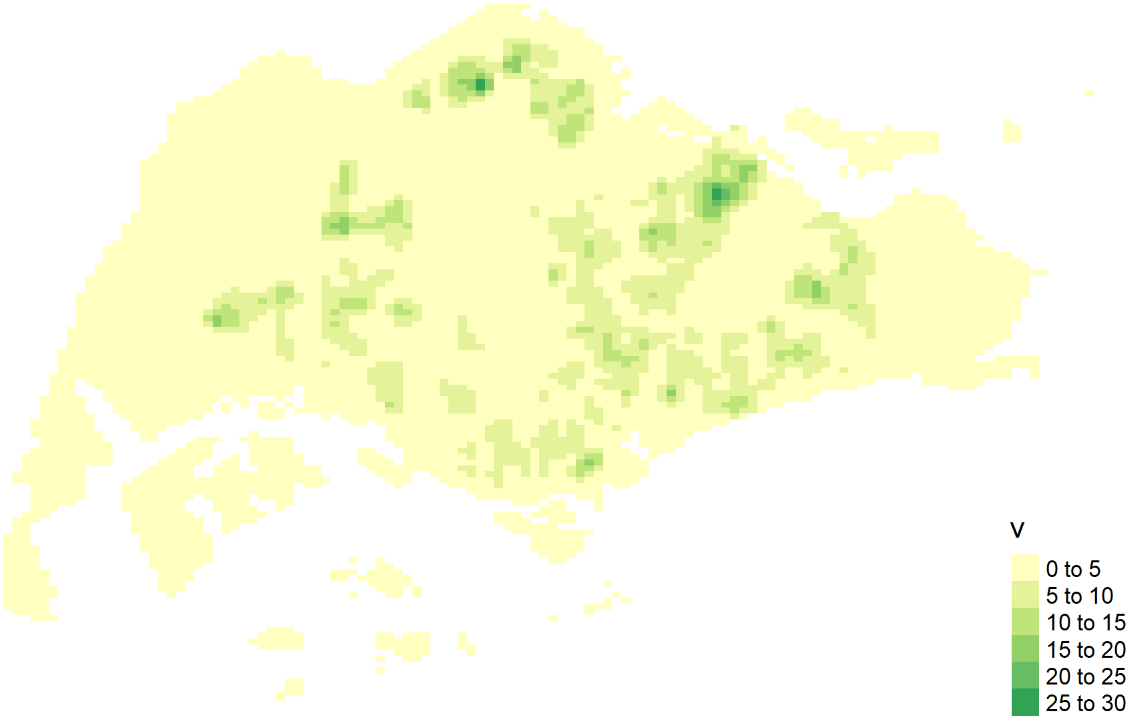
```
names       : v
values      : -8.476185e-15, 28.51831  (min, max)
```

Notice that the crs property is completed.

Visualising the output in tmap

Finally, we will display the raster in cartographic quality map using **tmap** package.

```
tm_shape(kde_childcareSG_bw_raster) +
  tm_raster("v") +
  tm_layout(legend.position = c("right", "bottom"), frame = FALSE)
```



Notice that the raster values are encoded explicitly onto the raster pixel using the values in "v"" field.

Comparing Spatial Point Patterns using KDE

In this section, you will learn how to compare KDE of childcare at Ponggol, Tampines, Chua Chu Kang and Jurong West planning areas.

EXTRACTING STUDY AREA

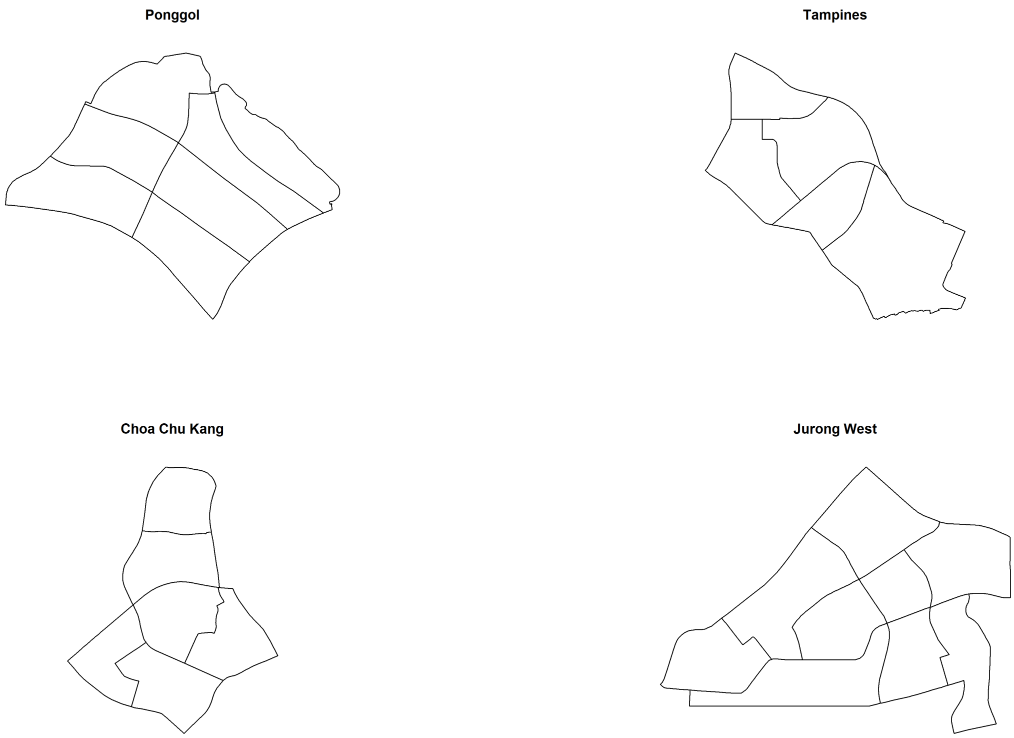
The code chunk below will be used to extract the target planning areas.

```
pg = mpsz[mpsz@data$PLN_AREA_N ==
```

```
"PUNGGOL",]
tm = mpsz [ mpsz @ data $ PLN_AREA_N ==
"TAMPINES",]
ck = mpsz [ mpsz @ data $ PLN_AREA_N ==
"CHOA CHU KANG",]
jw = mpsz [ mpsz @ data $ PLN_AREA_N ==
"JURONG WEST" ,]
```

Plotting target planning areas

```
par ( mfrow= c ( 2 ,2 ) )
plot ( pg , main = "Ponggol")
plot ( tm , main = "Tampines")
plot ( ck , main = "Choa Chu Kang")
plot ( jw , main = "Jurong West" )
```



CONVERTING THE SPATIAL POINT DATA FRAME INTO GENERIC SP FORMAT

Next, we will convert these SpatialPolygonsDataFrame layers into generic spatialpolygons layers.

```
pg_sp = as ( pg , "SpatialPolygons")
tm_sp = as ( tm , "SpatialPolygons")
ck_sp = as ( ck , "SpatialPolygons")
jw_sp = as ( jw , "SpatialPolygons")
```

CREATING *OWIN* OBJECT

Now, we will convert these SpatialPolygons objects into owin objects that is required by **spatstat**.

```
pg_owin = as(pg_sp, "owin")
tm_owin = as(tm_sp, "owin")
ck_owin = as(ck_sp, "owin")
jw_owin = as(jw_sp, "owin")
```

COMBINING CHILDCARE POINTS AND THE STUDY AREA

By using the code chunk below, we are able to extract childcare that is within the specific region to do our analysis later on.

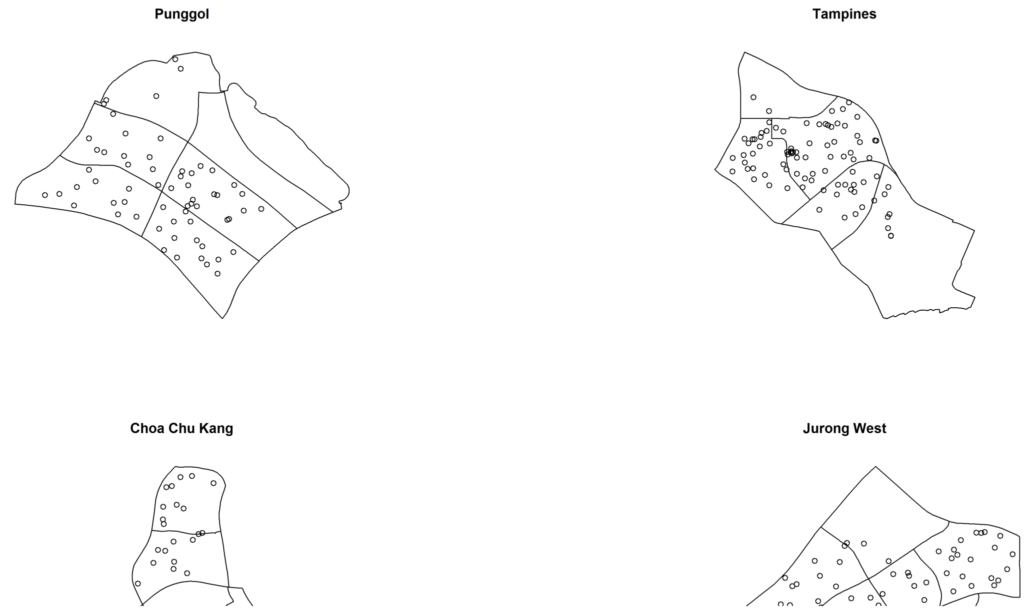
```
childcare_pg_ppp = childcare_ppp_jit[pg_owin]
childcare_tm_ppp = childcare_ppp_jit[tm_owin]
childcare_ck_ppp = childcare_ppp_jit[ck_owin]
childcare_jw_ppp = childcare_ppp_jit[jw_owin]
```

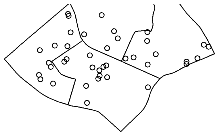
Next, *rescale()* function is used to trasnform the unit of measurement from metre to kilometre.

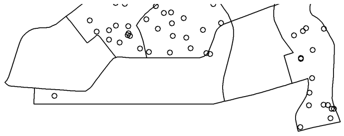
```
childcare_pg_ppp.km = rescale(childcare_pg_ppp, 1000, "km")
childcare_tm_ppp.km = rescale(childcare_tm_ppp, 1000, "km")
childcare_ck_ppp.km = rescale(childcare_ck_ppp, 1000, "km")
childcare_jw_ppp.km = rescale(childcare_jw_ppp, 1000, "km")
```

The code chunk below is used to plot these four study areas and the locations of the childcare centres.

```
par(mfrow=c(2,2))
plot(childcare_pg_ppp.km, main="Punggol")
plot(childcare_tm_ppp.km, main="Tampines")
plot(childcare_ck_ppp.km, main="Choa Chu Kang")
plot(childcare_jw_ppp.km, main="Jurong West")
```



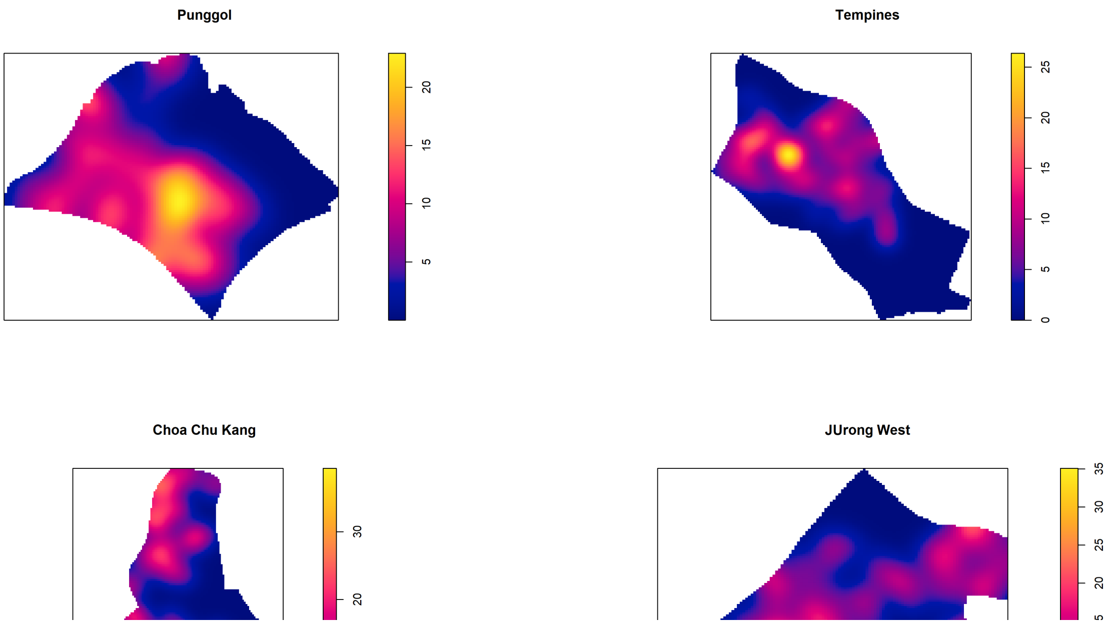




COMPUTING KDE

The code chunk below will be used to compute the KDE of these four planning area. ***bw.diggle*** method is used to derive the bandwidth of each

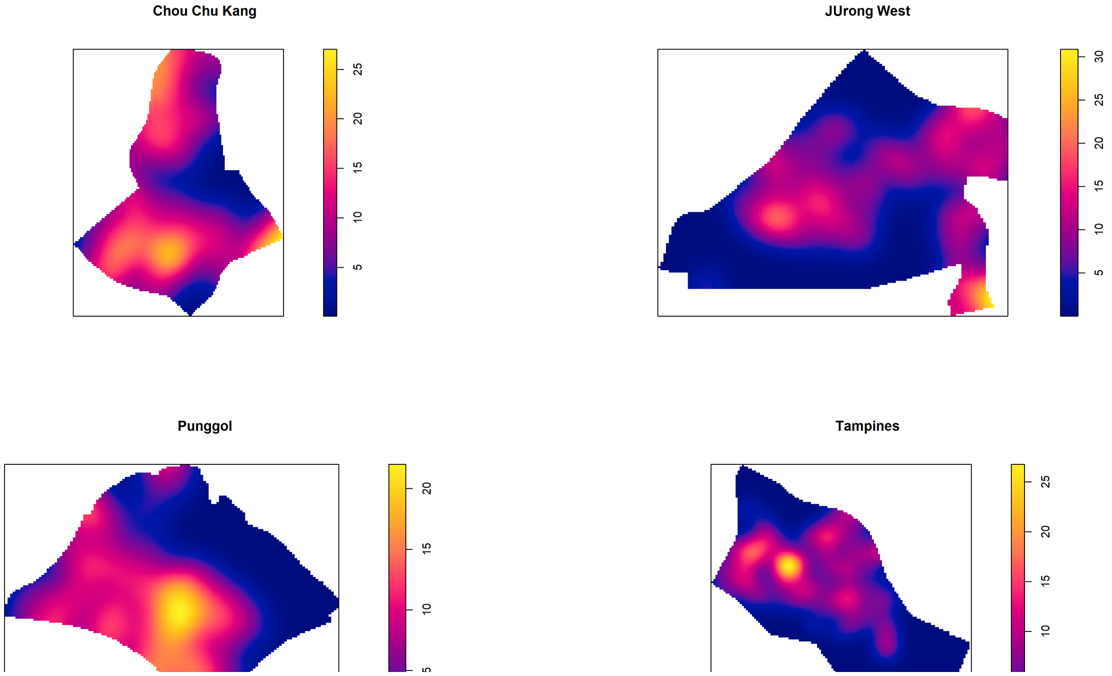
```
par(mfrow=c(2,2))
plot(density(childcare_pg_ppp.km,
             sigma=bw.diggle,
             edge=TRUE,
             kernel="gaussian"),
     main="Punggol")
plot(density(childcare_tm_ppp.km,
             sigma=bw.diggle,
             edge=TRUE,
             kernel="gaussian"),
     main="Tempines")
plot(density(childcare_ck_ppp.km,
             sigma=bw.diggle,
             edge=TRUE,
             kernel="gaussian"),
     main="Choa Chu Kang")
plot(density(childcare_jw_ppp.km,
             sigma=bw.diggle,
             edge=TRUE,
             kernel="gaussian"),
     main="JUrong West")
```

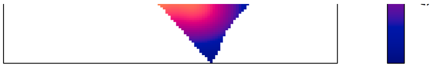


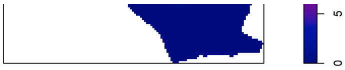
COMPUTING FIXED BANDWIDTH KDE

For comparison purposes, we will use 250m as the bandwidth.

```
par(mfrow=c(2,2))
plot(density(childcare_ck_ppp.km,
             sigma=0.25,
             edge=TRUE,
             kernel="gaussian"),
     main="Chou Chu Kang")
plot(density(childcare_jw_ppp.km,
             sigma=0.25,
             edge=TRUE,
             kernel="gaussian"),
     main="JUrong West")
plot(density(childcare_pg_ppp.km,
             sigma=0.25,
             edge=TRUE,
             kernel="gaussian"),
     main="Punggol")
plot(density(childcare_tm_ppp.km,
             sigma=0.25,
             edge=TRUE,
             kernel="gaussian"),
     main="Tampines")
```







Nearest Neighbour Analysis

In this section, we will perform the Clark-Evans test of aggregation for a spatial point pattern by using *clarkevans.test()* of **statspat**.

The test hypotheses are:

Ho = The distribution of childcare services are randomly distributed.

H1= The distribution of childcare services are not randomly distributed.

The 95% confident interval will be used.

Testing spatial point patterns using Clark and Evans Test

```
clarkevans.test(        childcareSG_ppp,
                correction=        "none"    ,
                clipregion=        "sg_owin",
                alternative=        c         (         "clustered")         ,
                nsim=        99        )
```

```

    Clark-Evans test
    No edge correction
    Monte Carlo test based on 99 simulations of CSR with fixed n

data:  childcareSG_ppp
R = 0.54756, p-value = 0.01
alternative hypothesis: clustered (R < 1)
```

What conclusion can you draw from the test result?

Clark and Evans Test: Choa Chu Kang planning area

In the code chunk below, *clarkevans.test()* of **spatstat** is used to performs Clark-Evans test of aggregation for childcare centre in Choa Chu Kang planning area.

```
clarkevans.test(        childcare_ck_ppp,
                correction=        "none"    ,
                clipregion=        NULL     ,
                alternative=        c         (         "two.sided")         ,
```

```
                 nsim=        999      )
```

```
    Clark-Evans test
    No edge correction
    Monte Carlo test based on 999 simulations of CSR with fixed n

data:  childcare_ck_ppp
R = 0.86606, p-value = 0.014
alternative hypothesis: two-sided
```

Clark and Evans Test: Tampines planning area

In the code chunk below, the similar test is used to analyse the spatial point patterns of childcare centre in Tampines planning area.

```
clarkevans.test(        childcare_tm_ppp,
                 correction=        "none"    ,
                 clipregion=        NULL      ,
                 alternative=        c        (         "two.sided")        ,
                 nsim=        999      )
```

```
    Clark-Evans test
    No edge correction
    Monte Carlo test based on 999 simulations of CSR with fixed n

data:  childcare_tm_ppp
R = 0.79934, p-value = 0.002
alternative hypothesis: two-sided
```

Second-order Spatial Point Patterns Analysis

Analysing Spatial Point Process Using G-Function

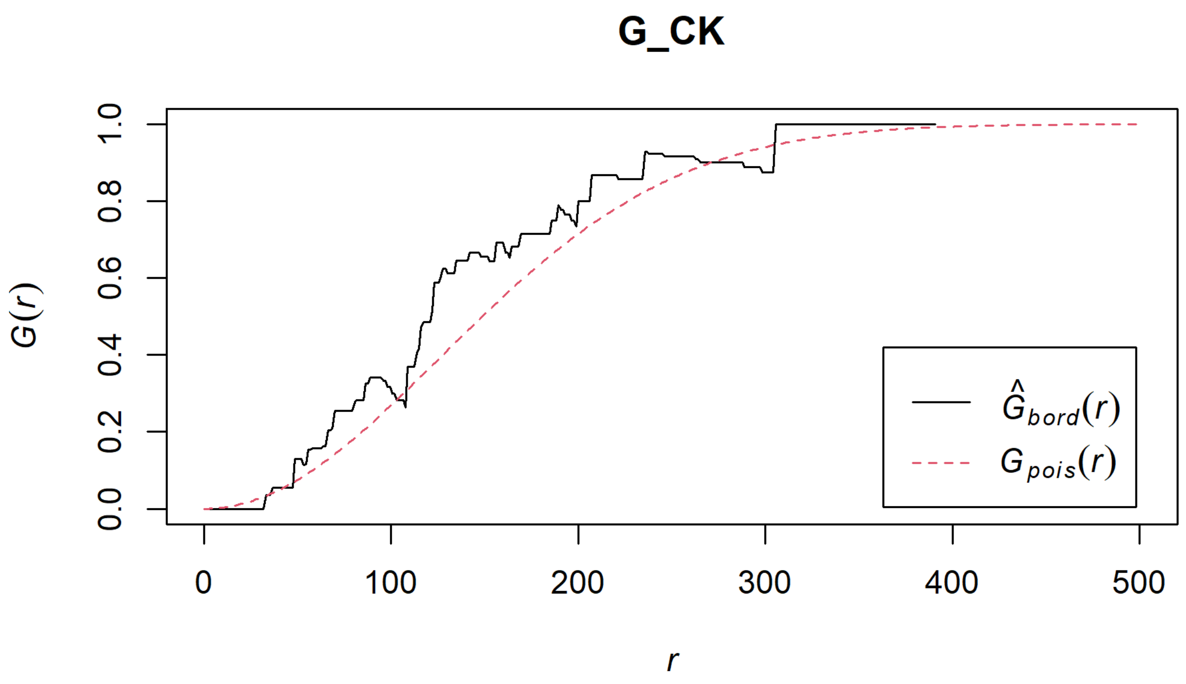
The G function measures the distribution of the distances from an arbitrary event to its nearest event. In this section, you will learn how to compute G-function estimation by using *Gest()* of **spatstat** package. You will also learn how to perform monta carlo simulation test using *envelope()* of **spatstat** package.

Choa Chu Kang planning area

COMPUTING G-FUNCTION ESTIMATION

The code chunk below is used to compute G-function using *Gest()* of **spatat** package.

```
G_CK = Gest(childcare_ck_ppp, correction = "border")
plot(G_CK, xlim=c(0,500))
```



PERFORMING COMPLETE SPATIAL RANDOMNESS TEST

To confirm the observed spatial patterns above, a hypothesis test will be conducted. The hypothesis and test are as follows:

Ho = The distribution of childcare services at Choa Chu Kang are randomly distributed.

H1= The distribution of childcare services at Choa Chu Kang are not randomly distributed.

The null hypothesis will be rejected if p-value is smaller than alpha value of 0.001.

Monte Carlo test with G-fucntion

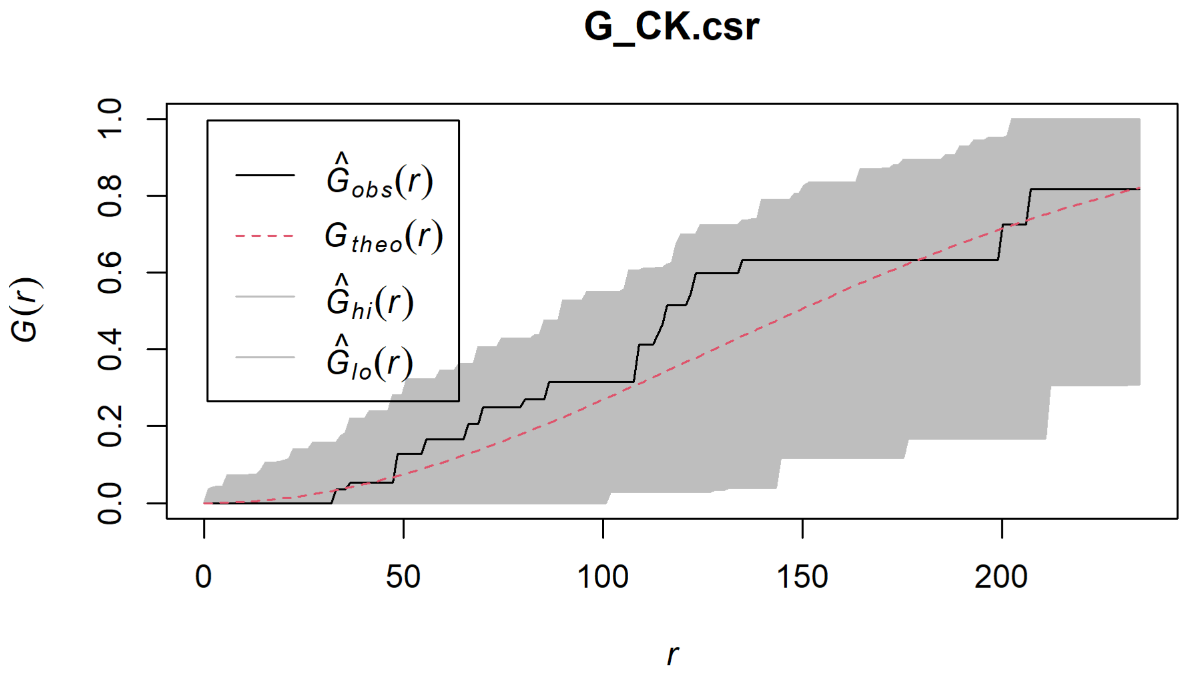
```
G_CK.csr <- envelope(childcare_ck_ppp, Gest, nsim = 999)
```

```
Generating 999 simulations of CSR ...
1, 2, 3, ......10.........20.........30.........40.........50.........60
.........70.........80.........90.........100.........110.........120
.........130.........140.........150.........160.........170.........180
.........190.........200.........210.........220.........230.........240
.........250.........260.........270.........280.........290.........300
.........310.........320.........330.........340.........350.........360
.........370.........380.........390.........400.........410.........420
.........430.........440.........450.........460.........470.........480
```

```
.........490.........500.........510.........520.........530.........540
.........550.........560.........570.........580.........590.........600
.........610.........620.........630.........640.........650.........660
.........670.........680.........690.........700.........710.........720
.........730.........740.........750.........760.........770.........780
.........790.........800.........810.........820.........830.........840
.........850.........860.........870.........880.........890.........900
.........910.........920.........930.........940.........950.........960
.........970.........980.........990........ 999.

Done.
```

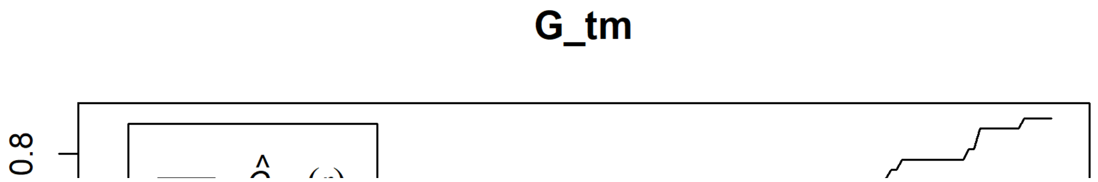
```
plot ( G_CK.csr )
```

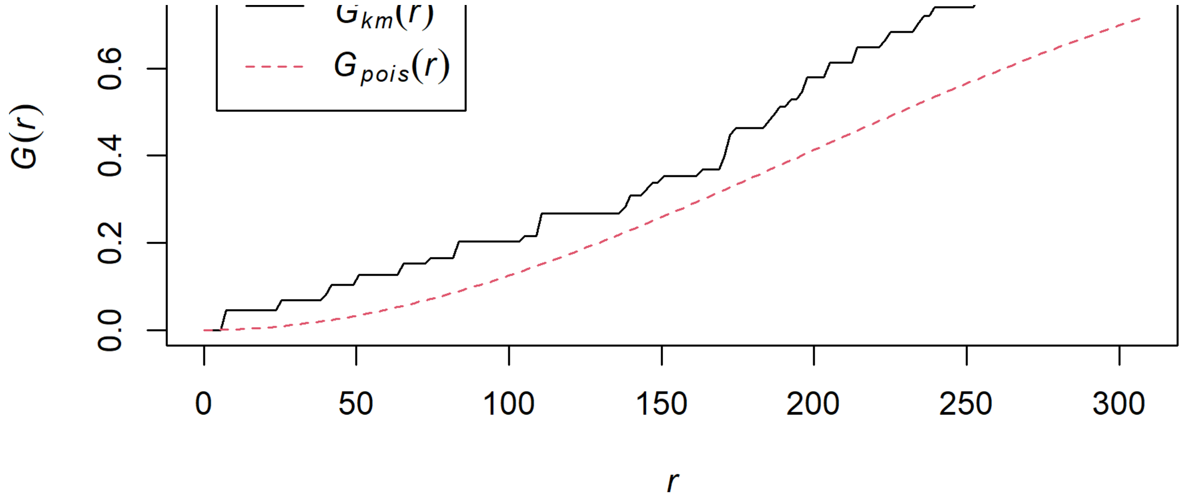


Tampines planning area

COMPUTING G-FUNCTION ESTIMATION

```
G_tm = Gest ( childcare_tm_ppp, correction = "best" )
plot ( G_tm )
```





PERFORMING COMPLETE SPATIAL RANDOMNESS TEST

To confirm the observed spatial patterns above, a hypothesis test will be conducted. The hypothesis and test are as follows:

Ho = The distribution of childcare services at Tampines are randomly distributed.

H1= The distribution of childcare services at Tampines are not randomly distributed.

The null hypothesis will be rejected is p-value is smaller than alpha value of 0.001.

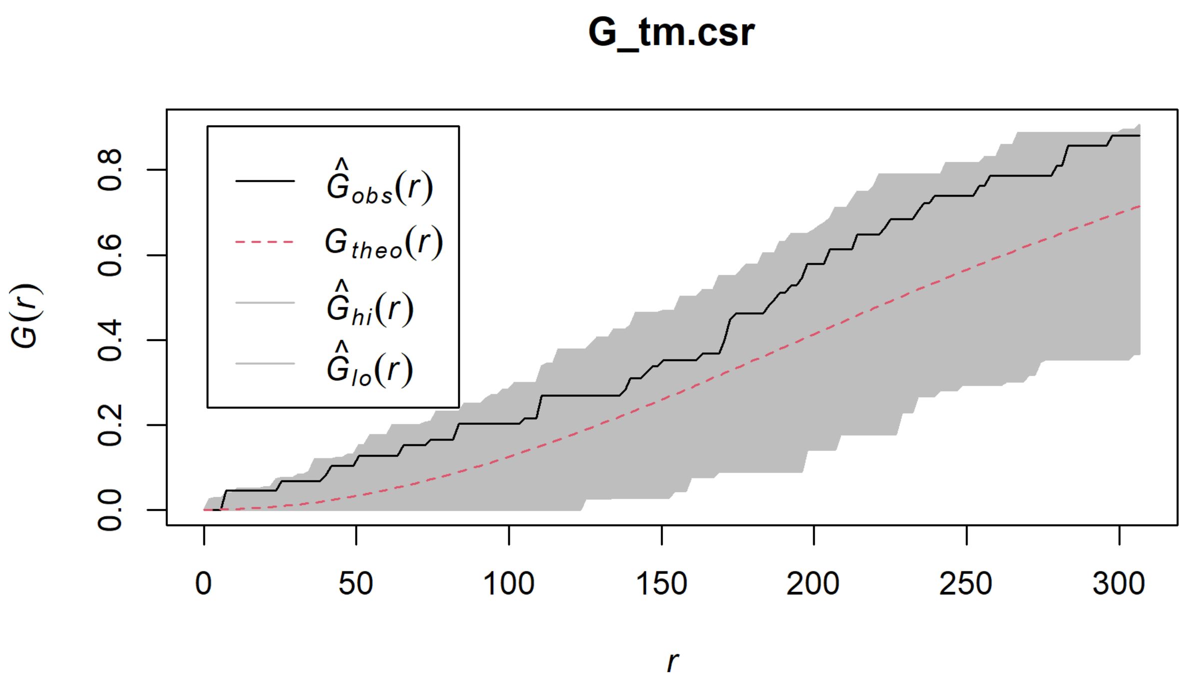
The code chunk below is used to perform the hypothesis testing.

```
G_tm.csr <- envelope(childcare_tm_ppp, Gest, correction = "all", nsim = 999)
```

```
Generating 999 simulations of CSR  ...
1, 2, 3, ......10.........20.........30.........40.........50.........60
.........70.........80.........90.........100.........110.........120
.........130.........140.........150.........160.........170.........180
.........190.........200.........210.........220.........230.........240
.........250.........260.........270.........280.........290.........300
.........310.........320.........330.........340.........350.........360
.........370.........380.........390.........400.........410.........420
.........430.........440.........450.........460.........470.........480
.........490.........500.........510.........520.........530.........540
.........550.........560.........570.........580.........590.........600
.........610.........620.........630.........640.........650.........660
.........670.........680.........690.........700.........710.........720
.........730.........740.........750.........760.........770.........780
.........790.........800.........810.........820.........830.........840
.........850.........860.........870.........880.........890.........900
.........910.........920.........930.........940.........950.........960
.........970.........980.........990........ 999.
```

```
Done.
```

```
plot(G_tm.csr)
```



Analysing Spatial Point Process Using F-Function

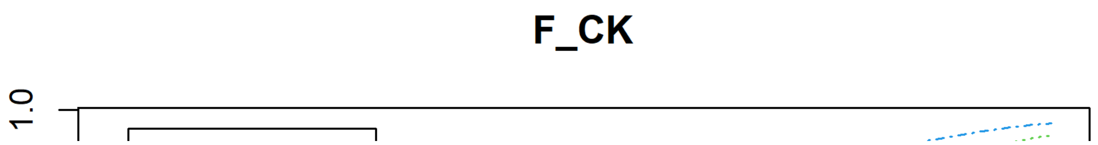
The F function estimates the empty space function F(r) or its hazard rate h(r) from a point pattern in a window of arbitrary shape. In this section, you will learn how to compute F-function estimation by using *Fest()* of **spatstat** package. You will also learn how to perform monta carlo simulation test using *envelope()* of **spatstat** package.

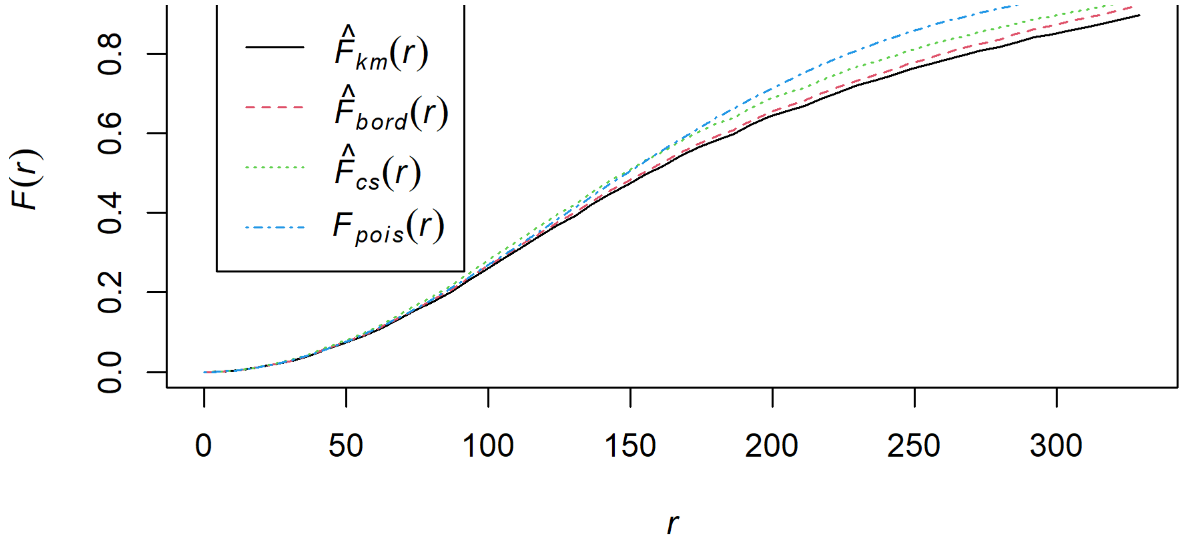
Choa Chu Kang planning area

COMPUTING F-FUNCTION ESTIMATION

The code chunk below is used to compute F-function using *Fest()* of **spatat** package.

```
F_CK = Fest(childcare_ck_ppp)
plot(F_CK)
```





Performing Complete Spatial Randomness Test

To confirm the observed spatial patterns above, a hypothesis test will be conducted. The hypothesis and test are as follows:

Ho = The distribution of childcare services at Choa Chu Kang are randomly distributed.

H1= The distribution of childcare services at Choa Chu Kang are not randomly distributed.

The null hypothesis will be rejected if p-value is smaller than alpha value of 0.001.

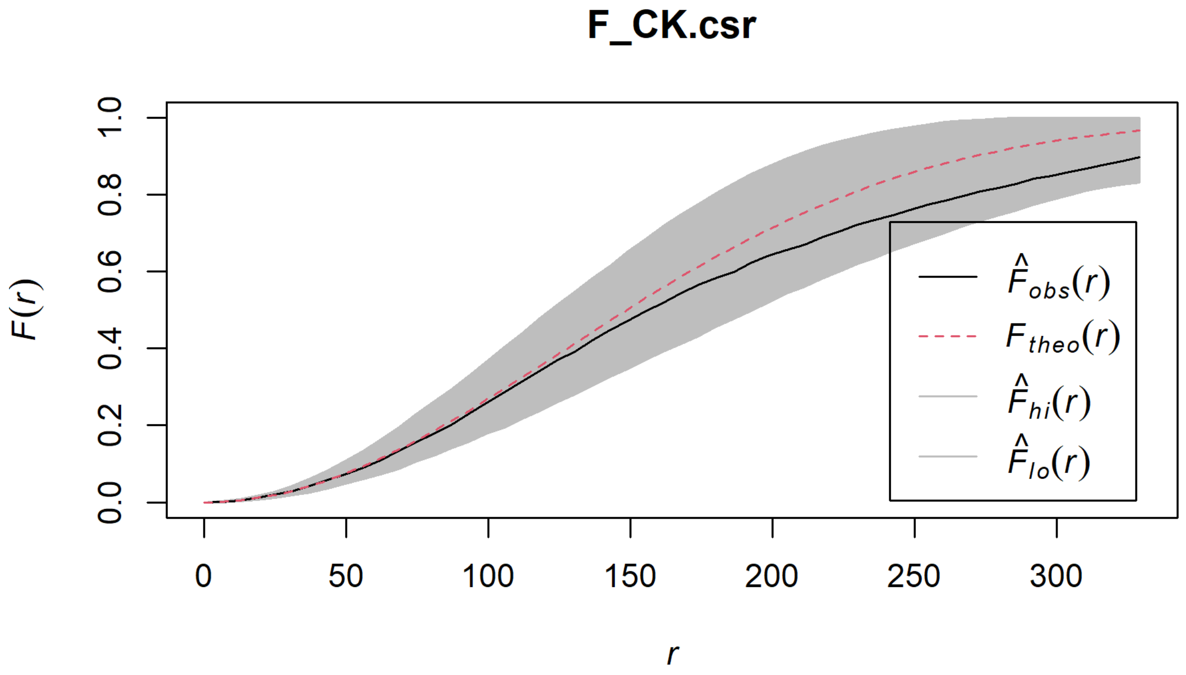
Monte Carlo test with F-fucntion

```
F_CK.csr <- envelope(childcare_ck_ppp, Fest, nsim = 999)
```

```
Generating 999 simulations of CSR ...
1, 2, 3, ......10.........20.........30.........40.........50.........60
.........70.........80.........90.........100.........110.........120
.........130.........140.........150.........160.........170.........180
.........190.........200.........210.........220.........230.........240
.........250.........260.........270.........280.........290.........300
.........310.........320.........330.........340.........350.........360
.........370.........380.........390.........400.........410.........420
.........430.........440.........450.........460.........470.........480
.........490.........500.........510.........520.........530.........540
.........550.........560.........570.........580.........590.........600
.........610.........620.........630.........640.........650.........660
.........670.........680.........690.........700.........710.........720
.........730.........740.........750.........760.........770.........780
.........790.........800.........810.........820.........830.........840
.........850.........860.........870.........880.........890.........900
.........910.........920.........930.........940.........950.........960
.........970.........980.........990........ 999.
```

```
Done.
```

```
plot(F_CK.csr)
```

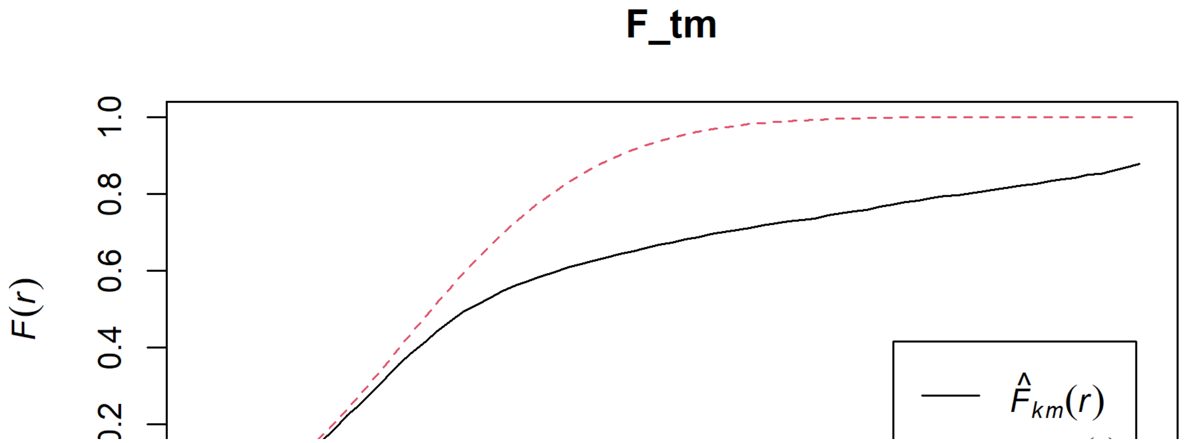


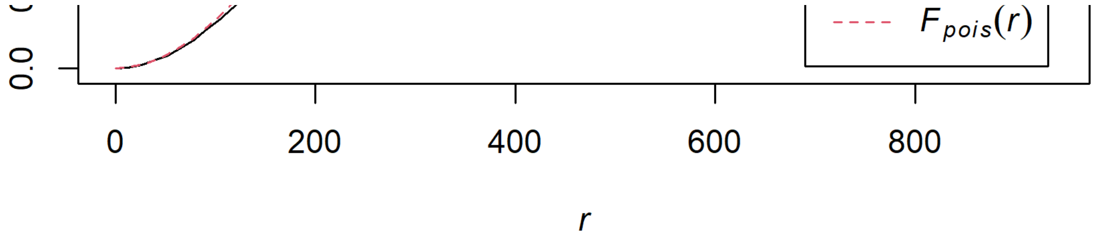
Tampines planning area

COMPUTING F-FUNCTION ESTIMATION

Monte Carlo test with F-fucntion

```
F_tm = Fest(childcare_tm_ppp, correction = "best")
plot(F_tm)
```





PERFORMING COMPLETE SPATIAL RANDOMNESS TEST

To confirm the observed spatial patterns above, a hypothesis test will be conducted. The hypothesis and test are as follows:

Ho = The distribution of childcare services at Tampines are randomly distributed.

H1= The distribution of childcare services at Tampines are not randomly distributed.

The null hypothesis will be rejected is p-value is smaller than alpha value of 0.001.

The code chunk below is used to perform the hypothesis testing.

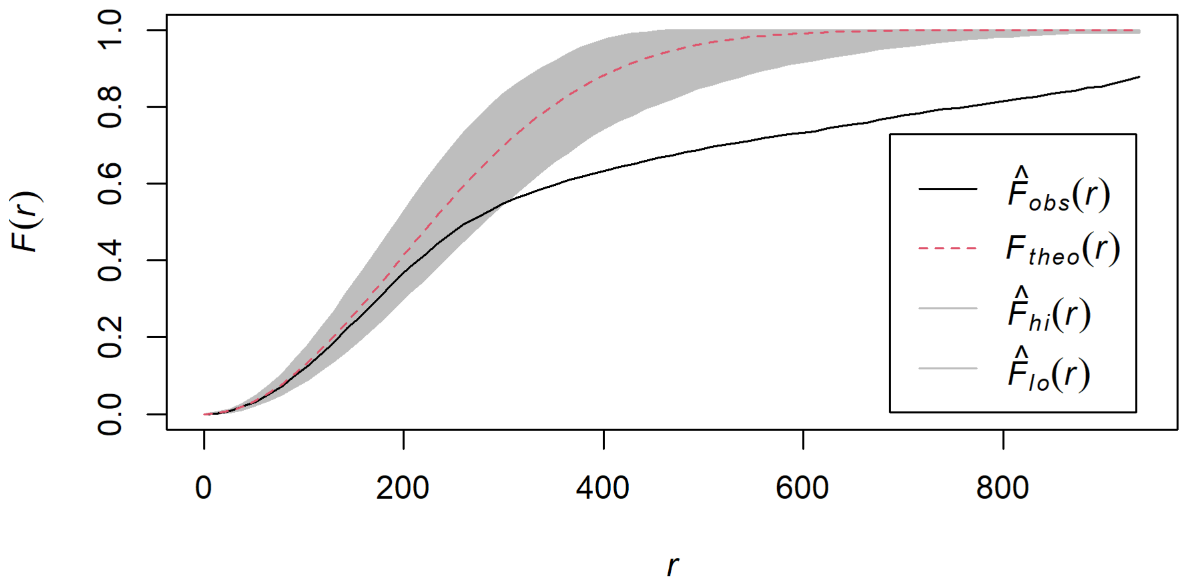
```
F_tm.csr <- envelope(childcare_tm_ppp, Fest, correction = "all"
, nsim = 999)
```

```
Generating 999 simulations of CSR ...
1, 2, 3, ......10.........20.........30.........40.........50.........60
.........70.........80.........90.........100.........110.........120
.........130.........140.........150.........160.........170.........180
.........190.........200.........210.........220.........230.........240
.........250.........260.........270.........280.........290.........300
.........310.........320.........330.........340.........350.........360
.........370.........380.........390.........400.........410.........420
.........430.........440.........450.........460.........470.........480
.........490.........500.........510.........520.........530.........540
.........550.........560.........570.........580.........590.........600
.........610.........620.........630.........640.........650.........660
.........670.........680.........690.........700.........710.........720
.........730.........740.........750.........760.........770.........780
.........790.........800.........810.........820.........830.........840
.........850.........860.........870.........880.........890.........900
.........910.........920.........930.........940.........950.........960
.........970.........980.........990........ 999.

Done.
```

```
plot(F_tm.csr)
```

F_tm.csr



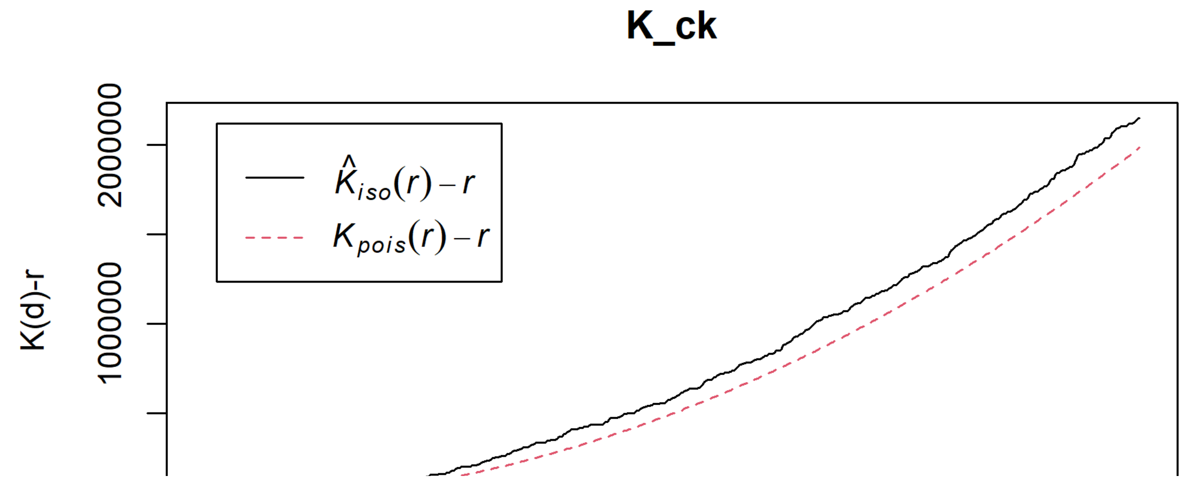
Analysing Spatial Point Process Using K-Function

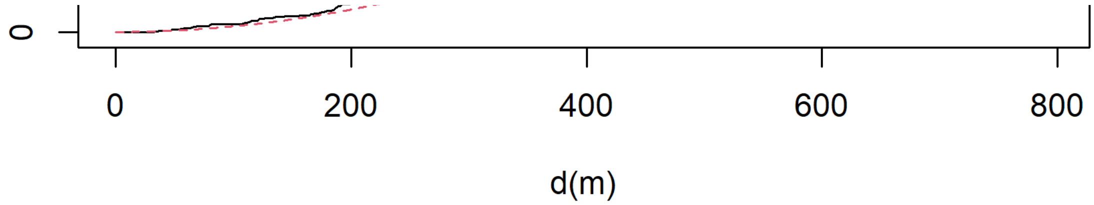
K-function measures the number of events found up to a given distance of any particular event. In this section, you will learn how to compute K-function estimates by using *Kest()* of **spatstat** package. You will also learn how to perform monta carlo simulation test using *envelope()* of spatstat package.

Choa Chu Kang planning area

COMPUTING K-FUCNTION ESTIMATE

```
K_ck = Kest(childcare_ck_ppp, correction = "Ripley")
plot(K_ck, . - r ~ r, ylab= "K(d)-r", xlab = "d(m)")
```





PERFORMING COMPLETE SPATIAL RANDOMNESS TEST

To confirm the observed spatial patterns above, a hypothesis test will be conducted. The hypothesis and test are as follows:

Ho = The distribution of childcare services at Choa Chu Kang are randomly distributed.

H1= The distribution of childcare services at Choa Chu Kang are not randomly distributed.

The null hypothesis will be rejected if p-value is smaller than alpha value of 0.001.

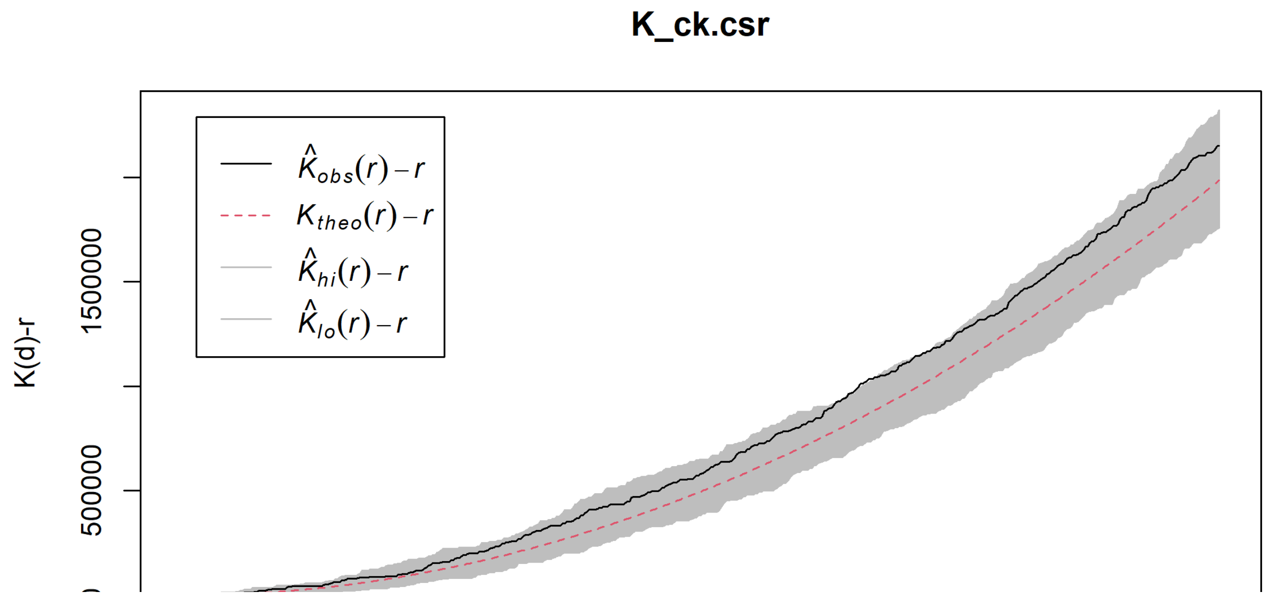
The code chunk below is used to perform the hypothesis testing.

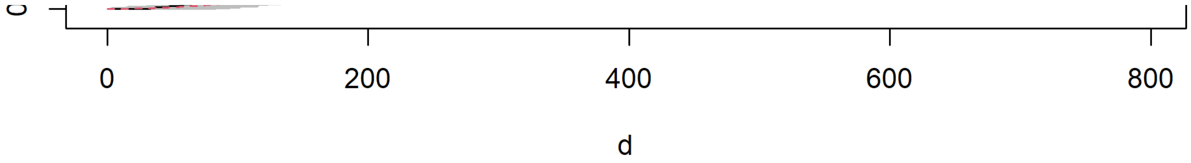
```
K_ck.csr <- envelope(childcare_ck_ppp, Kest, nsim = 99, rank = 1, glocal=TRUE)
```

```
Generating 99 simulations of CSR  ...
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, 2
36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 
71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 

Done.
```

```
plot(K_ck.csr, . - r ~ r, xlab="d", ylab="K(d)-r")
```

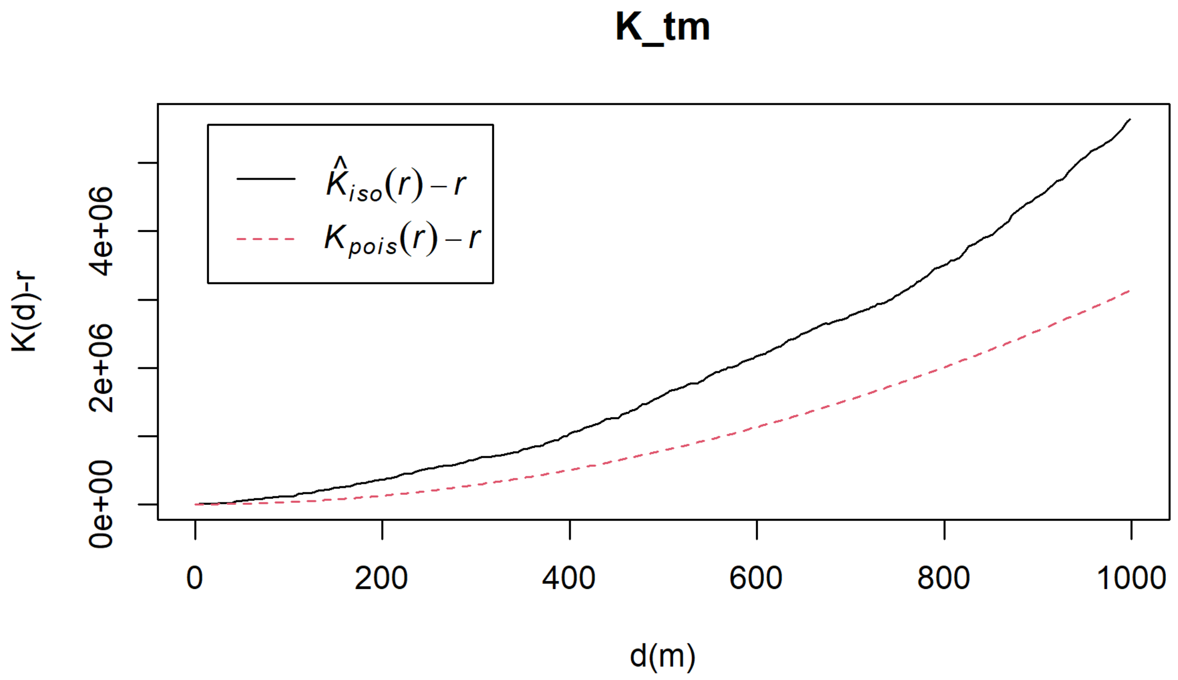




Tampines planning area

COMPUTING K-FUCNTION ESTIMATION

```
K_tm = Kest(childcare_tm_ppp, correction = "Ripley")
plot(K_tm, . -r ~ r,
     ylab= "K(d)-r", xlab = "d(m)",
     xlim=c(0,1000))
```



PERFORMING COMPLETE SPATIAL RANDOMNESS TEST

To confirm the observed spatial patterns above, a hypothesis test will be conducted. The hypothesis and test are as follows:

Ho = The distribution of childcare services at Tampines are randomly distributed.

H1= The distribution of childcare services at Tampines are not randomly distributed.

The null hypothesis will be rejected if p-value is smaller than alpha value of 0.001.

The code chunk below is used to perform the hypothesis testing.

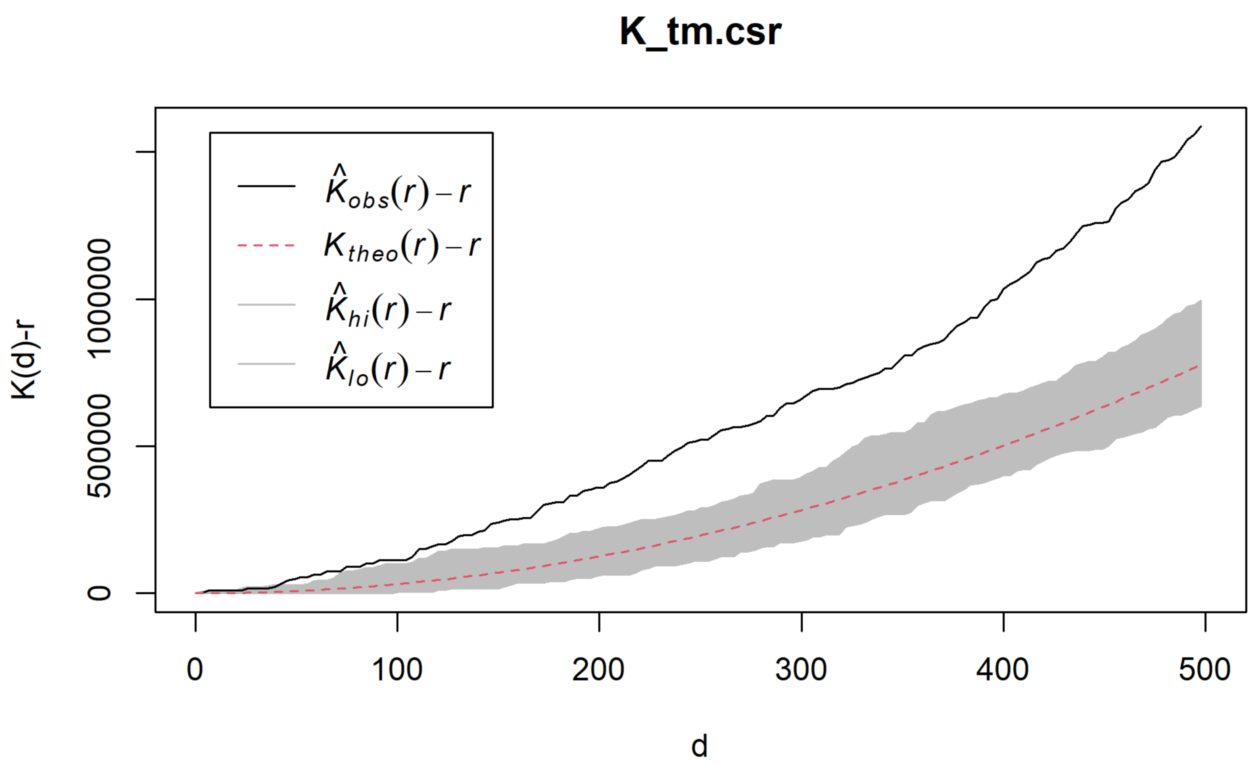
The code chunk below is used to perform the hypothesis testing.

```
K_tm.csr <- envelope(childcare_tm_ppp, Kest, nsim = 99, rank = 1, glocal=TRUE)
```

```
Generating 99 simulations of CSR ...
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
36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 6
71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 9

Done.
```

```
plot(K_tm.csr, . - r ~ r,
     xlab="d", ylab="K(d)-r", xlim=c(
0,500))
```



Analysing Spatial Point Process Using L-Function

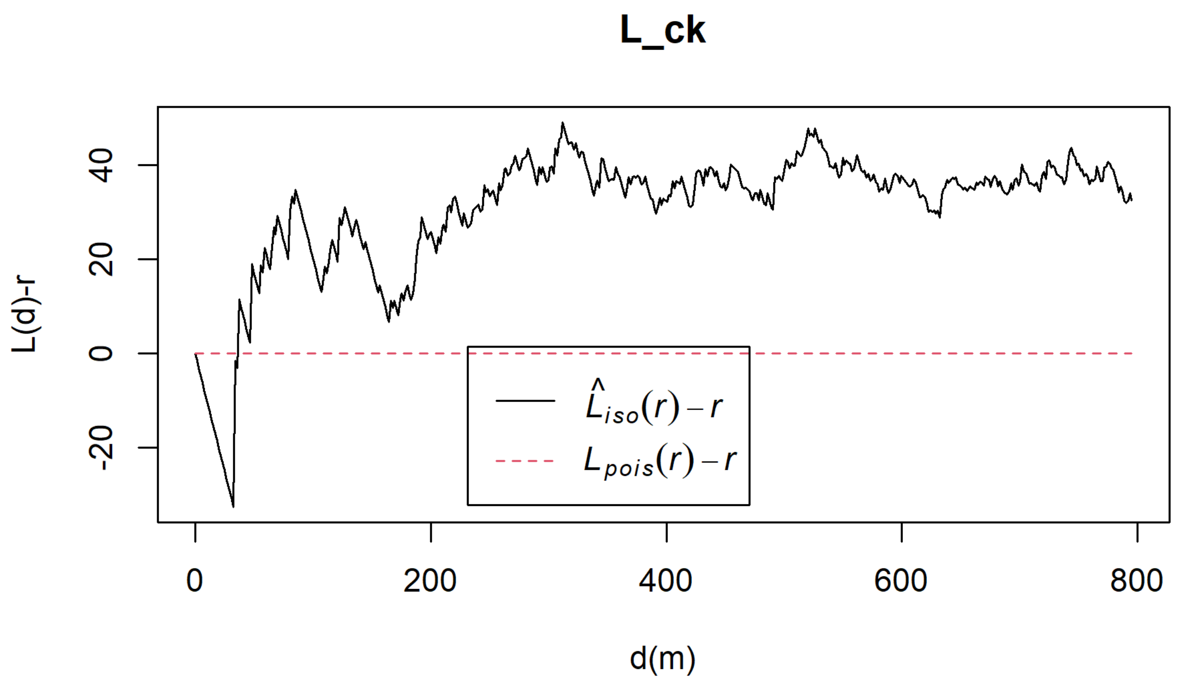
In this section, you will learn how to compute L-function estimation by using *Lest()* of **spatstat** package. You will also learn how to perform monta carlo simulation test using *envelope()* of spatstat package.

Choa Chu Kang planning area

COMPUTING L-FUCNTION ESTIMATION

COMPUTING L FUCNTION ESTIMATION

```
L_ck = Lest ( childcare_ck_ppp, correction = "Ripley" )
plot ( L_ck , . - r ~ r ,
    ylab= "L(d)-r" , xlab = "d(m)" )
```



PERFORMING COMPLETE SPATIAL RANDOMNESS TEST

To confirm the observed spatial patterns above, a hypothesis test will be conducted. The hypothesis and test are as follows:

Ho = The distribution of childcare services at Choa Chu Kang are randomly distributed.

H1= The distribution of childcare services at Choa Chu Kang are not randomly distributed.

The null hypothesis will be rejected if p-value if smaller than alpha value of 0.001.

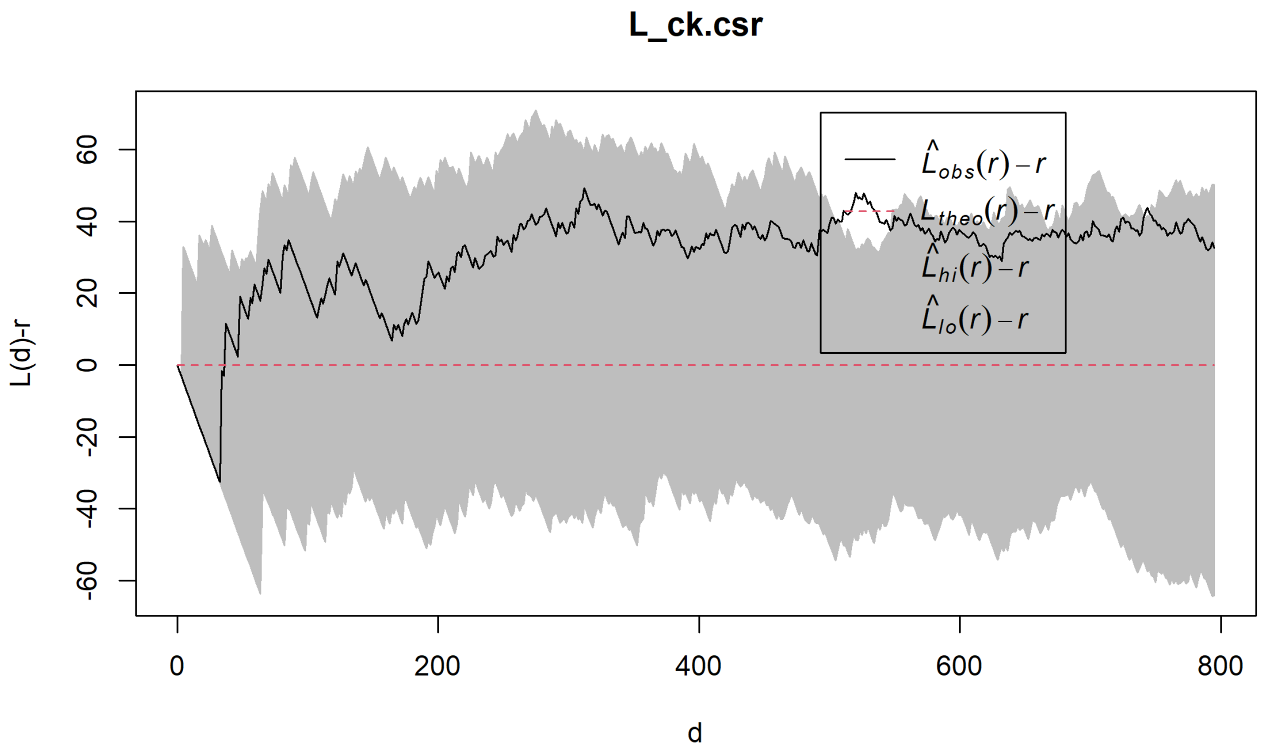
The code chunk below is used to perform the hypothesis testing.

```
L_ck.csr <- envelope ( childcare_ck_ppp, Lest , nsim = 99 , rank
= 1 , glocal= TRUE )
```

```
Generating 99 simulations of CSR ...
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
36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 6
71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 9

Done.
```

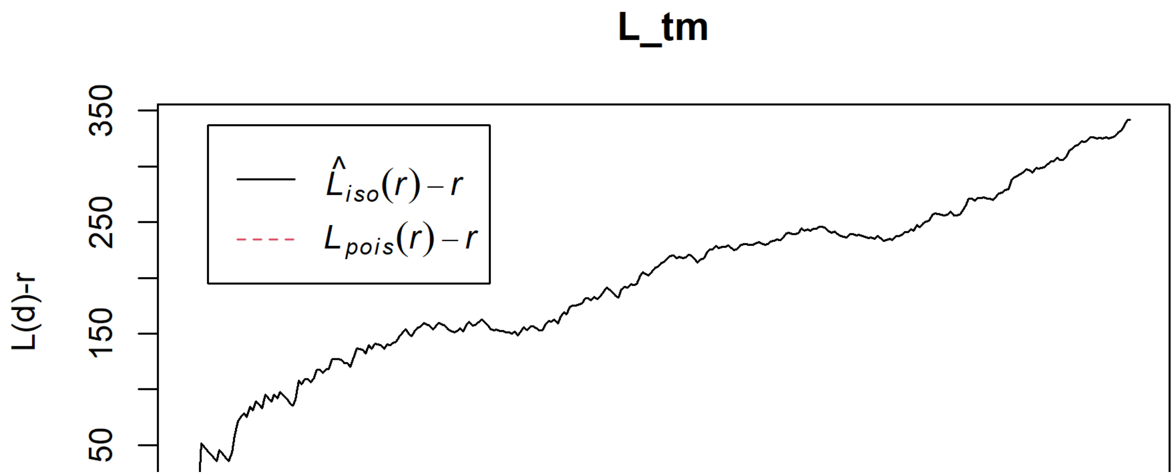
```
plot(L_ck.csr, . - r ~ r, xlab="d", ylab="L(d)-r")
```

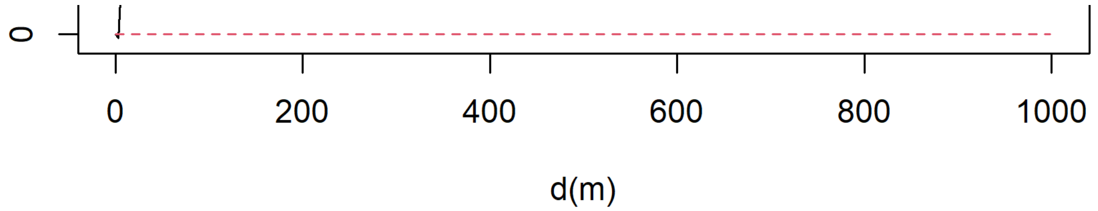


Tampines planning area

COMPUTING L-FUCNTION ESTIMATE

```
L_tm = Lest(childcare_tm_ppp, correction = "Ripley")
plot(L_tm, . - r ~ r,
     ylab= "L(d)-r", xlab = "d(m)",
     xlim=c(0,1000))
```





PERFORMING COMPLETE SPATIAL RANDOMNESS TEST

To confirm the observed spatial patterns above, a hypothesis test will be conducted. The hypothesis and test are as follows:

Ho = The distribution of childcare services at Tampines are randomly distributed.

H1= The distribution of childcare services at Tampines are not randomly distributed.

The null hypothesis will be rejected if p-value is smaller than alpha value of 0.001.

The code chunk below will be used to perform the hypothesis testing.

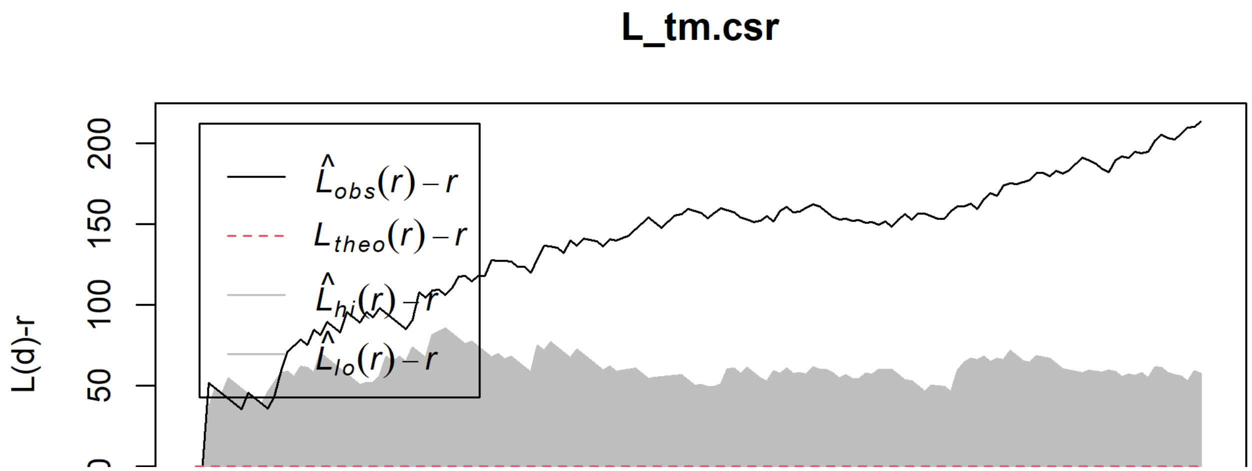
```
L_tm.csr <- envelope(childcare_tm_ppp, Lest, nsim = 99, rank = 1, glocal=TRUE)
```

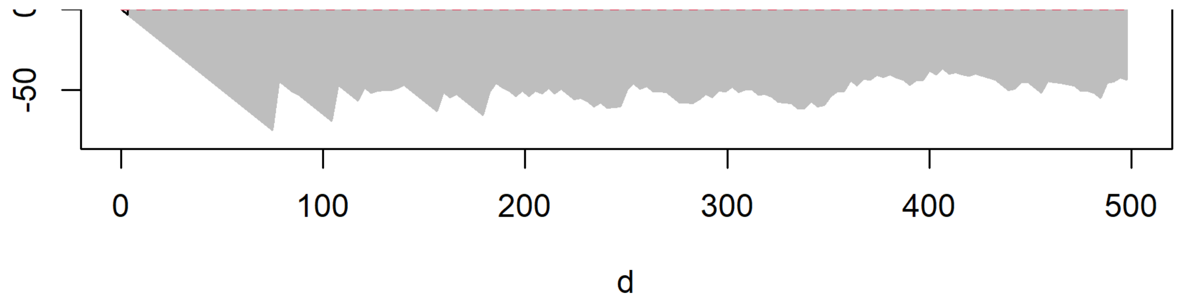
```
Generating 99 simulations of CSR ...
1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28
36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 6
71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 9

Done.
```

Then, plot the model output by using the code chun below.

```
plot(L_tm.csr, . - r ~ r,
    xlab="d", ylab="L(d)-r", xlim=c(
0,500))
```




-50
0
100
200
300
400
500
d