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

AUTHOR

Dr. Kam Tin Seong, Associate Professor of Information Systems (Practice)

AFFILIATION

School of Computing and Information Systems, Singapore Management University

PUBLISHED

May 25, 2021

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.

In this hands-on exercise, you will gain hands-on experience on using appropriate functions of spatstat to perform. The case study aims to discover the spatial point processes of childecare centres in Singapore.

The research questions

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 is downloaded from www.data.gov.sg and is in ESRI shapefile 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.
- 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:

-rgdal, which provides bindings to the 'Geospatial' Data Abstraction Library (GDAL) (>= 1.11.4) and access to projection/transformation operations from the PROJ library. In this exercise, rgdal will be used to import geospatial data in R and store as sp objects.

- **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('rgdal', 'maptools', '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, *readOGR()* of **rgdal** package will be used to import the three geospatial data in R's *spatialpolygonsdataframe*.

```
childcare <- readOGR(dsn = "data", layer="CHILDCARE")
```

```
OGR data source with driver: ESRI Shapefile
Source: "D:\tskam\GeoDSA\Hands-on_Ex\Hands-on_Ex04\data", layer: "CHILDCARE"
with 1312 features
It has 18 fields
```

```
sg <- readOGR(dsn = "data", layer="CostalOutline")
```

```
OGR data source with driver: ESRI Shapefile
Source: "D:\tskam\GeoDSA\Hands-on_Ex\Hands-on_Ex04\data", layer: "CostalOutline"
with 60 features
It has 4 fields
```

```
mpsz <- readOGR(dsn = "data", layer="MP14_SUBZONE_WEB_PL")
```

```
OGR data source with driver: ESRI Shapefile
Source: "D:\tskam\GeoDSA\Hands-on_Ex\Hands-on_Ex04\data", layer: "MP14_SUBZONE_WEB_PL"
with 323 features
It has 15 fields
```

Before we can use these data for analysis, it is important for us to ensure that they are projected in same projection system. We can retrieve the information of these geospatial data by using the code chunk below.

```
crs(childcare)
```

```
CRS arguments:
 +proj=tmerc +lat_0=1.36666666666667 +lon_0=103.833333333333
+k=1 +x_0=28001.642 +y_0=38744.572 +datum=WGS84 +units=m
+no_defs
```

```
crs(mpsz)
```

```
CRS arguments:
 +proj=tmerc +lat_0=1.36666666666667 +lon_0=103.833333333333
+k=1 +x_0=28001.642 +y_0=38744.572 +datum=WGS84 +units=m
+no_defs
```

```
crs(sg)
```

```
CRS arguments:
 +proj=tmerc +lat_0=1.36666666666667 +lon_0=103.833333333333
+k=1 +x_0=28001.642 +y_0=38744.572 +datum=WGS84 +units=m
+no_defs
```

Next, we can examine the imported geospatial data by using *plot()*.

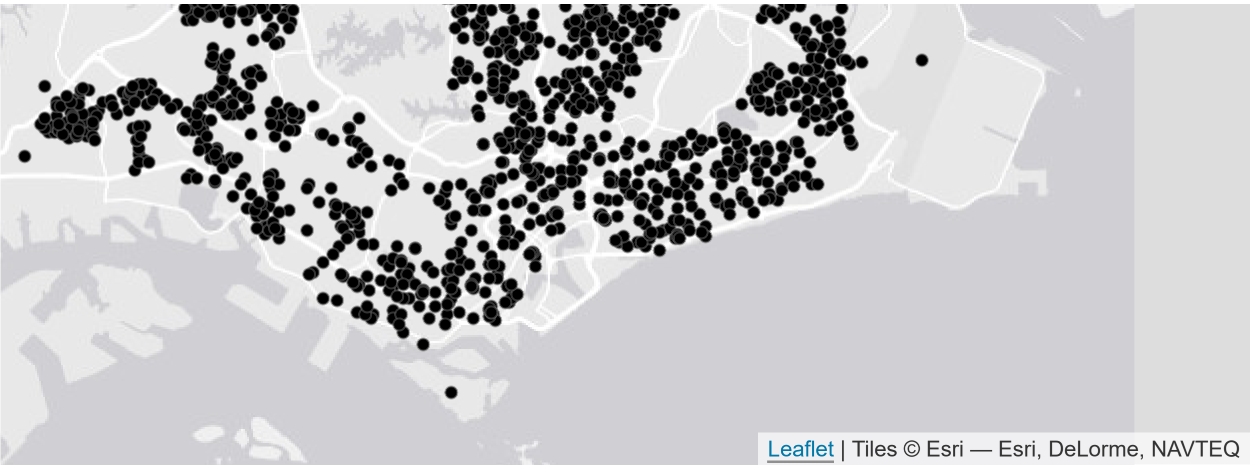
```
par(mfrow=c(1,3))
plot(childcare)
plot(mpsz)
plot(sg)
```

Alternatively, we can also plotting these three geospatial data in one plot by using code chunk below.

```
plot(sg, border="lightgrey")
plot(sg, add=TRUE)
plot(childcare, add=TRUE)
```

We can also prepare an interactive pin map by using the code chunk below.

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




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

Lastly, let us examine the childcare SpatialPointsDataFrame.

```
childcare
```

```
class       : SpatialPointsDataFrame
features    : 1312
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   : 18
names       : OBJECTID, ADDRESSBLO, ADDRESSBUI, ADDRESSPOS,
min values  :        1,         NA,         NA,     038983,
max values  :     1312,         NA,         NA,     829646, UPPER BASEMENT LEVEL WEST WING TERMINAL 1
```

Converting the spatial point data frame into generic sp format

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

The codes below will convert the SpatialPoint and SpatialPolygon data frame into generic spatialpoints and spatialpolygons objects.

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

Do you know what are the differences between SpatialPoints object and SpatialPointDataFrame object?

Let us plot the childcare_sp data by using the code chun below.

```
plot(childcare_sp)
```

Note that the output map look similar to the earlier plot.

How about we view the properties of childcare_sp data object by using the code chun below?

```
childcare_sp
```

```
class       : SpatialPoints
features    : 1312
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
```

Can you see the different now?

Converting the generic sp format into spatstat's ppp format

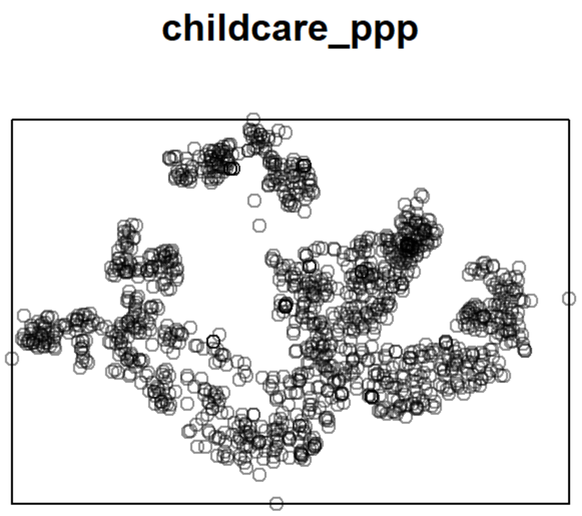
Now, we will use *as.ppp()* function of **spatstat** to convert the spatial data into **spatstat**'s ***ppp*** object format.

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

```
Planar point pattern: 1312 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:  1312 points
Average intensity 1.623186e-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 coindicence point, we will use the *multiplicity()* function as shown in the code chunk below.

```
multiplicity(        childcare_ppp)
```

```
  1   2   3   4   5   6   7   8   9  10  11  12  13  14 
  1   1   4   1   1   1   1   1   1   1   1   1   1   1 
 15  16  17  18  19  20  21  22  23  24  25  26  27  28 
  1   1   1   1   1   1   1   1   1   1   1   1   1   1 
 29  30  31  32  33  34  35  36  37  38  39  40  41  42 
  1   1   1   1   1   1   1   1   1   1   1   1   1   1 
 43  44  45  46  47  48  49  50  51  52  53  54  55  56 
  3   1   1   1   1   1   1   1   1   1   1   1   1   1 
 57  58  59  60  61  62  63  64  65  66  67  68  69  70 
  1   2   1   1   1   1   1   1   1   1   2   1   1   1 
 71  72  73  74  75  76  77  78  79  80  81  82  83  84 
  1   1   1   7   1   1   1   1   1   1   1   1   1   1 
 85  86  87  88  89  90  91  92  93  94  95  96  97  98 
  2   1   1   1   1   1   1   1   1   1   1   1   2   1 
 99 100 101 102 103 104 105 106 107 108 109 110 111 112 
  1   1   1   1   1   2   1   1   1   1   1   1   1   1 
113 114 115 116 117 118 119 120 121 122 123 124 125 126 
  1   1   1   1   1   1   2   1   1   1   1   5   1   1 
127 128 129 130 131 132 133 134 135 136 137 138 139 140 
  1   1   2   1   1   1   1   1   1   2   1   1   1   1 
141 142 143 144 145 146 147 148 149 150 151 152 153 154 
  1   1   1   1   1   1   1   1   1   1   1   1   1   2 
155 156 157 158 159 160 161 162 163 164 165 166 167 168 
  1   1   1   1   1   1   1   1   1   1   1   1   1   1 
169 170 171 172 173 174 175 176 177 178 179 180 181 182 
  1   1   1   1   1   1   1   1   1   1   1   1   1   1 
183 184 185 186 187 188 189 190 191 192 193 194 195 196 
  1   1   7   1   1   1   1   1   1   1   1   1   1   1 
197 198 199 200 201 202 203 204 205 206 207 208 209 210 
  5   1   1   1   1   1   1   2   1   1   1   1   1   1 
211 212 213 214 215 216 217 218 219 220 221 222 223 224 
  1   1   1   1   1   1   1   1   1   1   2   1   1   1 
225 226 227 228 229 230 231 232 233 234 235 236 237 238 
  1   1   2   1   1   1   1   1   1   1   1   1   1   1 
239 240 241 242 243 244 245 246 247 248 249 250 251 252 
  1   1   1   1   1   1   1   1   1   1   1   2   1   1 
253 254 255 256 257 258 259 260 261 262 263 264 265 266 
  1   1   2   2   1   2   1   1   3   1   1   1   1   1 
267 268 269 270 271 272 273 274 275 276 277 278 279 280 
  2   1   1   1   1   1   1   1   7   1   1   1   1   2 
281 282 283 284 285 286 287 288 289 290 291 292 293 294 
  1   2   1   1   2   1   1   1   1   1   1   1   1   1 
295 296 297 298 299 300 301 302 303 304 305 306 307 308 
  1   1   1   1   1   1   1   1   1   1   1   1   1   1 
309 310 311 312 313 314 315 316 317 318 319 320 321 322 
```

1	1	1	1	1	1	1	1	1	1	1	1	1	1
323	324	325	326	327	328	329	330	331	332	333	334	335	336
1	1	1	1	1	1	1	1	1	1	1	1	1	1
337	338	339	340	341	342	343	344	345	346	347	348	349	350
1	1	1	1	1	1	1	1	1	1	1	1	1	1
351	352	353	354	355	356	357	358	359	360	361	362	363	364
1	1	4	1	1	1	1	1	1	2	1	1	1	1
365	366	367	368	369	370	371	372	373	374	375	376	377	378
1	1	1	1	1	1	1	2	1	1	1	1	1	1
379	380	381	382	383	384	385	386	387	388	389	390	391	392
1	1	1	1	1	1	1	1	1	1	1	1	1	1
393	394	395	396	397	398	399	400	401	402	403	404	405	406
1	1	1	1	1	1	1	1	1	1	1	1	1	1
407	408	409	410	411	412	413	414	415	416	417	418	419	420
1	1	1	1	1	1	1	1	1	1	1	1	1	1
421	422	423	424	425	426	427	428	429	430	431	432	433	434
1	1	1	1	1	1	1	1	1	1	1	1	1	7
435	436	437	438	439	440	441	442	443	444	445	446	447	448
1	2	1	1	1	1	1	1	1	1	1	1	1	1
449	450	451	452	453	454	455	456	457	458	459	460	461	462
1	3	1	1	1	1	1	1	1	1	1	2	2	2
463	464	465	466	467	468	469	470	471	472	473	474	475	476
1	1	1	1	1	1	1	1	1	2	1	1	1	1
477	478	479	480	481	482	483	484	485	486	487	488	489	490
4	1	1	1	1	1	1	1	1	1	3	1	1	1
491	492	493	494	495	496	497	498	499	500	501	502	503	504
1	1	1	1	1	1	1	1	1	1	1	1	1	1
505	506	507	508	509	510	511	512	513	514	515	516	517	518
1	1	3	1	1	1	1	1	1	1	1	1	1	1
519	520	521	522	523	524	525	526	527	528	529	530	531	532
1	1	4	1	4	1	1	1	2	1	1	1	1	1
533	534	535	536	537	538	539	540	541	542	543	544	545	546
1	1	1	1	1	1	1	1	1	1	1	1	1	1
547	548	549	550	551	552	553	554	555	556	557	558	559	560
1	1	1	1	1	1	1	3	1	1	1	1	1	1
561	562	563	564	565	566	567	568	569	570	571	572	573	574
1	1	1	1	4	1	1	1	1	1	1	1	1	1
575	576	577	578	579	580	581	582	583	584	585	586	587	588
1	1	1	1	1	1	1	1	1	1	1	1	1	1
589	590	591	592	593	594	595	596	597	598	599	600	601	602
1	2	1	1	1	1	1	1	1	1	1	1	1	1
603	604	605	606	607	608	609	610	611	612	613	614	615	616
2	1	1	1	1	1	1	1	1	1	1	1	1	1
617	618	619	620	621	622	623	624	625	626	627	628	629	630
1	1	1	2	1	1	1	1	1	1	1	1	1	1
631	632	633	634	635	636	637	638	639	640	641	642	643	644
1	1	1	1	2	1	1	7	1	1	1	1	4	1
645	646	647	648	649	650	651	652	653	654	655	656	657	658
1	1	2	1	1	1	1	1	1	1	1	1	1	1
659	660	661	662	663	664	665	666	667	668	669	670	671	672
1	1	1	1	1	1	1	1	1	1	1	2	1	3

673	674	675	676	677	678	679	680	681	682	683	684	685	686
1	1	1	1	1	1	1	1	1	1	1	1	1	1
687	688	689	690	691	692	693	694	695	696	697	698	699	700
1	1	2	1	1	1	1	1	1	1	1	1	1	1
701	702	703	704	705	706	707	708	709	710	711	712	713	714
1	1	1	1	1	1	1	1	1	1	1	1	1	1
715	716	717	718	719	720	721	722	723	724	725	726	727	728
1	1	1	1	1	1	1	1	1	1	1	1	1	1
729	730	731	732	733	734	735	736	737	738	739	740	741	742
1	1	1	1	1	1	1	1	4	1	1	1	1	1
743	744	745	746	747	748	749	750	751	752	753	754	755	756
1	7	1	1	1	1	1	1	1	1	1	1	1	1
757	758	759	760	761	762	763	764	765	766	767	768	769	770
1	1	1	1	1	4	1	2	2	1	1	1	1	1
771	772	773	774	775	776	777	778	779	780	781	782	783	784
2	1	1	1	2	1	1	1	1	1	1	1	2	1
785	786	787	788	789	790	791	792	793	794	795	796	797	798
1	1	1	1	1	1	1	1	1	1	1	1	1	1
799	800	801	802	803	804	805	806	807	808	809	810	811	812
1	1	1	1	1	1	5	1	1	1	1	1	1	1
813	814	815	816	817	818	819	820	821	822	823	824	825	826
1	1	1	1	1	1	1	1	1	1	1	1	1	1
827	828	829	830	831	832	833	834	835	836	837	838	839	840
1	1	1	1	1	1	1	1	2	1	1	1	1	1
841	842	843	844	845	846	847	848	849	850	851	852	853	854
1	1	1	1	1	1	1	1	3	1	1	1	1	1
855	856	857	858	859	860	861	862	863	864	865	866	867	868
1	1	1	1	2	1	1	1	1	1	1	1	1	1
869	870	871	872	873	874	875	876	877	878	879	880	881	882
1	1	1	1	1	1	1	1	1	1	1	1	1	1
883	884	885	886	887	888	889	890	891	892	893	894	895	896
1	1	1	1	1	1	1	1	1	1	1	1	1	1
897	898	899	900	901	902	903	904	905	906	907	908	909	910
1	1	1	1	1	1	1	1	1	1	1	1	1	1
911	912	913	914	915	916	917	918	919	920	921	922	923	924
1	1	1	1	1	1	1	1	1	1	1	1	1	1
925	926	927	928	929	930	931	932	933	934	935	936	937	938
1	1	1	1	1	1	1	1	1	1	1	1	1	1
939	940	941	942	943	944	945	946	947	948	949	950	951	952
1	1	1	1	1	3	1	1	1	1	1	2	1	1
953	954	955	956	957	958	959	960	961	962	963	964	965	966
1	1	1	1	1	1	1	1	1	1	1	1	1	1
967	968	969	970	971	972	973	974	975	976	977	978	979	980
1	1	1	1	1	1	2	1	1	1	1	1	1	1
981	982	983	984	985	986	987	988	989	990	991	992	993	994
1	1	1	1	1	1	1	1	1	1	1	1	1	1
995	996	997	998	999	1000	1001	1002	1003	1004	1005	1006	1007	1008
1	5	1	1	1	1	1	1	1	1	1	1	1	1
1009	1010	1011	1012	1013	1014	1015	1016	1017	1018	1019	1020	1021	1022
1	1	1	1	1	1	1	1	1	1	1	1	1	1
1023	1024	1025	1026	1027	1028	1029	1030	1031	1032	1033	1034	1035	1036

```
   1    1    1    1    1    1    1    1    1    1    1    1    1    1 
1037 1038 1039 1040 1041 1042 1043 1044 1045 1046 1047 1048 1049 1050 
   1    1    1    1    1    1    1    1    1    1    1    1    1    1 
1051 1052 1053 1054 1055 1056 1057 1058 1059 1060 1061 1062 1063 1064 
   1    1    5    1    1    1    1    1    1    1    1    1    1    1 
1065 1066 1067 1068 1069 1070 1071 1072 1073 1074 1075 1076 1077 1078 
   1    1    1    1    4    1    1    1    1    1    1    1    1    1 
1079 1080 1081 1082 1083 1084 1085 1086 1087 1088 1089 1090 1091 1092 
   1    1    1    1    1    1    1    1    1    1    1    1    1    1 
1093 1094 1095 1096 1097 1098 1099 1100 1101 1102 1103 1104 1105 1106 
   1    1    1    1    1    1    1    1    1    1    1    1    1    1 
1107 1108 1109 1110 1111 1112 1113 1114 1115 1116 1117 1118 1119 1120 
   1    1    1    1    1    1    1    1    1    1    1    1    1    1 
1121 1122 1123 1124 1125 1126 1127 1128 1129 1130 1131 1132 1133 1134 
   1    1    1    1    1    1    1    1    1    1    1    1    1    1 
1135 1136 1137 1138 1139 1140 1141 1142 1143 1144 1145 1146 1147 1148 
   1    1    1    1    1    1    1    1    1    1    1    1    1    1 
1149 1150 1151 1152 1153 1154 1155 1156 1157 1158 1159 1160 1161 1162 
   1    1    1    1    2    1    1    1    1    1    1    1    1    1 
1163 1164 1165 1166 1167 1168 1169 1170 1171 1172 1173 1174 1175 1176 
   1    1    1    1    1    1    1    1    1    1    1    1    1    1 
1177 1178 1179 1180 1181 1182 1183 1184 1185 1186 1187 1188 1189 1190 
   1    1    1    1    1    1    1    1    1    1    4    1    1    1 
1191 1192 1193 1194 1195 1196 1197 1198 1199 1200 1201 1202 1203 1204 
   1    1    1    1    2    1    1    1    1    1    1    1    1    1 
1205 1206 1207 1208 1209 1210 1211 1212 1213 1214 1215 1216 1217 1218 
   1    1    1    1    1    1    1    1    1    1    1    1    1    1 
1219 1220 1221 1222 1223 1224 1225 1226 1227 1228 1229 1230 1231 1232 
   1    1    1    1    1    1    1    1    1    1    1    1    1    1 
1233 1234 1235 1236 1237 1238 1239 1240 1241 1242 1243 1244 1245 1246 
   1    1    1    1    1    1    1    1    1    1    1    1    1    1 
1247 1248 1249 1250 1251 1252 1253 1254 1255 1256 1257 1258 1259 1260 
   1    1    2    1    1    1    1    4    2    1    1    1    1    1 
1261 1262 1263 1264 1265 1266 1267 1268 1269 1270 1271 1272 1273 1274 
   1    1    1    1    1    1    2    1    1    1    1    1    1    1 
1275 1276 1277 1278 1279 1280 1281 1282 1283 1284 1285 1286 1287 1288 
   1    1    1    1    1    1    1    1    1    1    1    1    1    1 
1289 1290 1291 1292 1293 1294 1295 1296 1297 1298 1299 1300 1301 1302 
   1    2    1    1    1    1    1    1    1    1    1    1    1    2 
1303 1304 1305 1306 1307 1308 1309 1310 1311 1312 
   1    1    1    1    1    1    1    1    7    2 
```

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

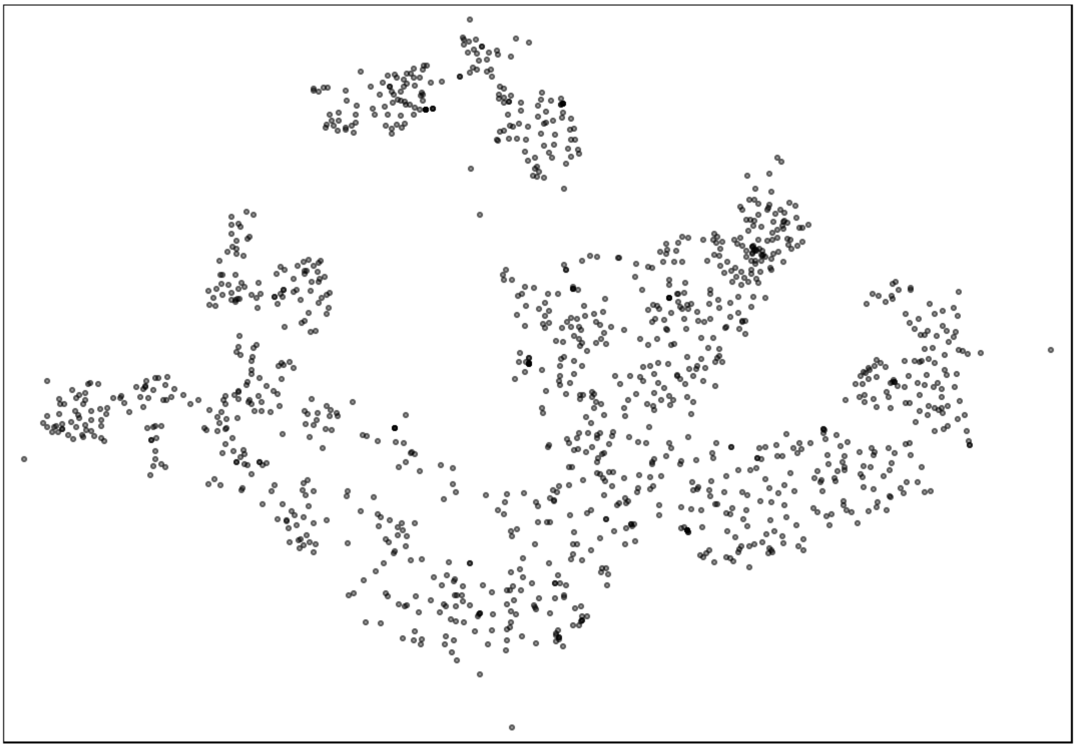
```
sum(multiplicity(childcare_ppp) > 1)
```

```
[1] 85
```

The output shows that there are 85 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("plot")
tm_shape(childcare) +
  tm_dots(alpha=0.4, size=0.05)
```



```
tmap_mode("plot")
```

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.

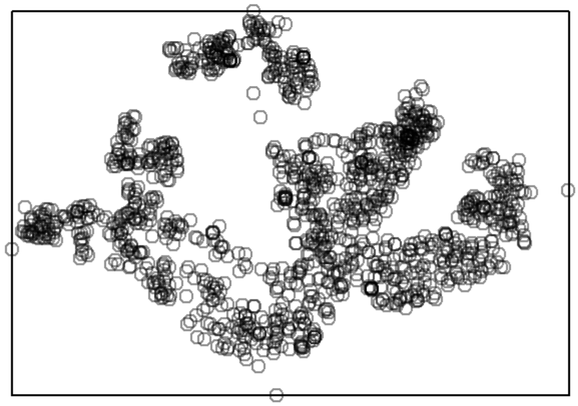
The third solution is to make each point "unique" and then attach the duplicates of the points to the patterns as **marks**, as attributes of the points. Then you would need analytical techniques that take into account these marks.

The code chunk below implements the jittering approach.

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

plot(childcare_ppp_jit)
```

childcare_ppp_jit



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

```
[1] FALSE
```

Notice the difference with the original plot. Can you see how the circumference do not overlap perfectly now?

Creating *owin*

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.

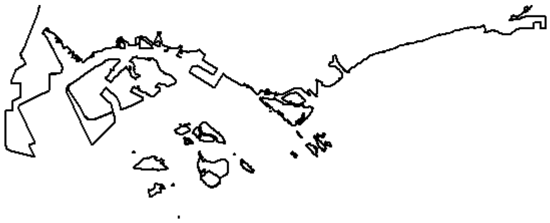
The code chunk below is used to covert *sg* SpatialPolygon object into owin object of **spatstat**.

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

The ouput object can be displayed by using plot() and summary() functions.

```
plot(sg_owin)
```

sg_owin



```r
summary(sg_owin)
```

```
Window: polygonal boundary
60 separate polygons (no holes)
            vertices        area relative.area
polygon 1         38 1.56140e+04      2.09e-05
polygon 2        735 4.69093e+06      6.27e-03
polygon 3         49 1.66986e+04      2.23e-05
polygon 4         76 3.12332e+05      4.17e-04
polygon 5       5141 6.36179e+08      8.50e-01
polygon 6         42 5.58317e+04      7.46e-05
polygon 7         67 1.31354e+06      1.75e-03
polygon 8         15 4.46420e+03      5.96e-06
polygon 9         14 5.46674e+03      7.30e-06
polygon 10        37 5.26194e+03      7.03e-06
polygon 11        53 3.44003e+04      4.59e-05
polygon 12        74 5.82234e+04      7.78e-05
polygon 13        69 5.63134e+04      7.52e-05
polygon 14       143 1.45139e+05      1.94e-04
polygon 15       165 3.38736e+05      4.52e-04
polygon 16       130 9.40465e+04      1.26e-04
polygon 17        19 1.80977e+03      2.42e-06
polygon 18        16 2.01046e+03      2.69e-06
polygon 19        93 4.30642e+05      5.75e-04
polygon 20        90 4.15092e+05      5.54e-04
polygon 21       721 1.92795e+06      2.57e-03
polygon 22       330 1.11896e+06      1.49e-03
polygon 23       115 9.28394e+05      1.24e-03
polygon 24        37 1.01705e+04      1.36e-05
polygon 25        25 1.66227e+04      2.22e-05
polygon 26        10 2.14507e+03      2.86e-06
polygon 27       190 2.02489e+05      2.70e-04
polygon 28       175 9.25904e+05      1.24e-03
polygon 29      1993 9.99217e+06      1.33e-02
polygon 30        38 2.42492e+04      3.24e-05
polygon 31        24 6.35239e+03      8.48e-06
polygon 32        53 6.35791e+05      8.49e-04
polygon 33        41 1.60161e+04      2.14e-05
polygon 34        22 2.54368e+03      3.40e-06
```

```
polygon 35        30 1.08382e+04      1.45e-05
polygon 36       327 2.16921e+06      2.90e-03
polygon 37       111 6.62927e+05      8.85e-04
polygon 38        90 1.15991e+05      1.55e-04
polygon 39        98 6.26829e+04      8.37e-05
polygon 40       415 3.25384e+06      4.35e-03
polygon 41       222 1.51142e+06      2.02e-03
polygon 42       107 6.33039e+05      8.45e-04
polygon 43         7 2.48299e+03      3.32e-06
polygon 44        17 3.28303e+04      4.38e-05
polygon 45        26 8.34758e+03      1.11e-05
polygon 46       177 4.67446e+05      6.24e-04
polygon 47        16 3.19460e+03      4.27e-06
polygon 48        15 4.87296e+03      6.51e-06
polygon 49        66 1.61841e+04      2.16e-05
polygon 50       149 5.63430e+06      7.53e-03
polygon 51       609 2.62570e+07      3.51e-02
polygon 52         8 7.82256e+03      1.04e-05
polygon 53       976 2.33447e+07      3.12e-02
polygon 54        55 8.25379e+04      1.10e-04
polygon 55       976 2.33447e+07      3.12e-02
polygon 56        61 3.33449e+05      4.45e-04
polygon 57         6 1.68410e+04      2.25e-05
polygon 58         4 9.45963e+03      1.26e-05
polygon 59        46 6.99702e+05      9.35e-04
polygon 60        13 7.00873e+04      9.36e-05
enclosing rectangle: [2663.93, 56047.79] x [16357.98, 50244.03] units
                     (53380 x 33890 units)
Window area = 748741000 square units
Fraction of frame area: 0.414
```

Combining childcare points and the study area

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

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

Here we plot the combined childcare point and Punggol region to prove that it works

```
plot     (         childcareSG_ppp)
```

```
summary  (        childcareSG_ppp)
```

```
Planar point pattern:  1312 points
Average intensity 1.752274e-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: polygonal boundary
60 separate polygons (no holes)
            vertices        area relative.area
polygon 1         38 1.56140e+04      2.09e-05
polygon 2        735 4.69093e+06      6.27e-03
polygon 3         49 1.66986e+04      2.23e-05
polygon 4         76 3.12332e+05      4.17e-04
polygon 5       5141 6.36179e+08      8.50e-01
polygon 6         42 5.58317e+04      7.46e-05
polygon 7         67 1.31354e+06      1.75e-03
polygon 8         15 4.46420e+03      5.96e-06
polygon 9         14 5.46674e+03      7.30e-06
polygon 10        37 5.26194e+03      7.03e-06
polygon 11        53 3.44003e+04      4.59e-05
polygon 12        74 5.82234e+04      7.78e-05
polygon 13        69 5.63134e+04      7.52e-05
polygon 14       143 1.45139e+05      1.94e-04
polygon 15       165 3.38736e+05      4.52e-04
polygon 16       130 9.40465e+04      1.26e-04
polygon 17        19 1.80977e+03      2.42e-06
polygon 18        16 2.01046e+03      2.69e-06
polygon 19        93 4.30642e+05      5.75e-04
polygon 20        90 4.15092e+05      5.54e-04
polygon 21       721 1.92795e+06      2.57e-03
polygon 22       330 1.11896e+06      1.49e-03
polygon 23       115 9.28394e+05      1.24e-03
```

```
polygon 24        37 1.01705e+04      1.36e-05
polygon 25        25 1.66227e+04      2.22e-05
polygon 26        10 2.14507e+03      2.86e-06
polygon 27       190 2.02489e+05      2.70e-04
polygon 28       175 9.25904e+05      1.24e-03
polygon 29      1993 9.99217e+06      1.33e-02
polygon 30        38 2.42492e+04      3.24e-05
polygon 31        24 6.35239e+03      8.48e-06
polygon 32        53 6.35791e+05      8.49e-04
polygon 33        41 1.60161e+04      2.14e-05
polygon 34        22 2.54368e+03      3.40e-06
polygon 35        30 1.08382e+04      1.45e-05
polygon 36       327 2.16921e+06      2.90e-03
polygon 37       111 6.62927e+05      8.85e-04
polygon 38        90 1.15991e+05      1.55e-04
polygon 39        98 6.26829e+04      8.37e-05
polygon 40       415 3.25384e+06      4.35e-03
polygon 41       222 1.51142e+06      2.02e-03
polygon 42       107 6.33039e+05      8.45e-04
polygon 43         7 2.48299e+03      3.32e-06
polygon 44        17 3.28303e+04      4.38e-05
polygon 45        26 8.34758e+03      1.11e-05
polygon 46       177 4.67446e+05      6.24e-04
polygon 47        16 3.19460e+03      4.27e-06
polygon 48        15 4.87296e+03      6.51e-06
polygon 49        66 1.61841e+04      2.16e-05
polygon 50       149 5.63430e+06      7.53e-03
polygon 51       609 2.62570e+07      3.51e-02
polygon 52         8 7.82256e+03      1.04e-05
polygon 53       976 2.33447e+07      3.12e-02
polygon 54        55 8.25379e+04      1.10e-04
polygon 55       976 2.33447e+07      3.12e-02
polygon 56        61 3.33449e+05      4.45e-04
polygon 57         6 1.68410e+04      2.25e-05
polygon 58         4 9.45963e+03      1.26e-05
polygon 59        46 6.99702e+05      9.35e-04
polygon 60        13 7.00873e+04      9.36e-05
enclosing rectangle: [2663.93, 56047.79] x [16357.98, 50244.03] units
                     (53380 x 33890 units)
Window area = 748741000 square units
Fraction of frame area: 0.414
```

First-order SPPA

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

- 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 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*.

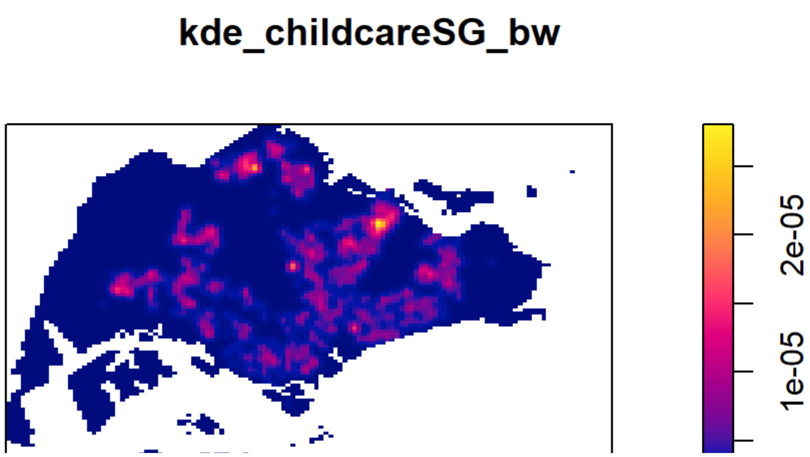
```
kde_childcareSG_bw <- density(childcareSG_ppp,
                              sigma=bw.diggle,
                              edge=TRUE,
                            kernel="gaussian")
```

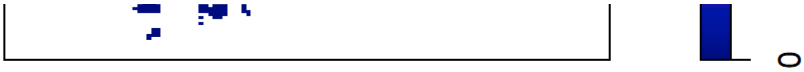
```
bw <- bw.diggle(childcareSG_ppp)
bw
```

```
   sigma
298.4095
```

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

```
plot(kde_childcareSG_bw)
```





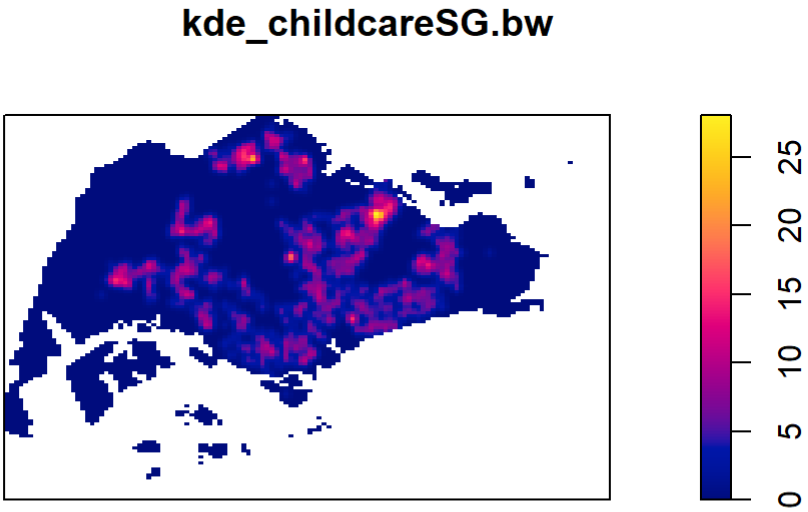
The density values of the output range from 0 to 0.000035 which is way too small to comprehend. This is because the default unit of measurement of svy21 is in meter. As a result, the density values computed is in “number of points per square meter”.

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
3.080455
```

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

```
 sigma.x  sigma.y
2.303178 1.492997
```

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

```
    sigma
0.3310477
```

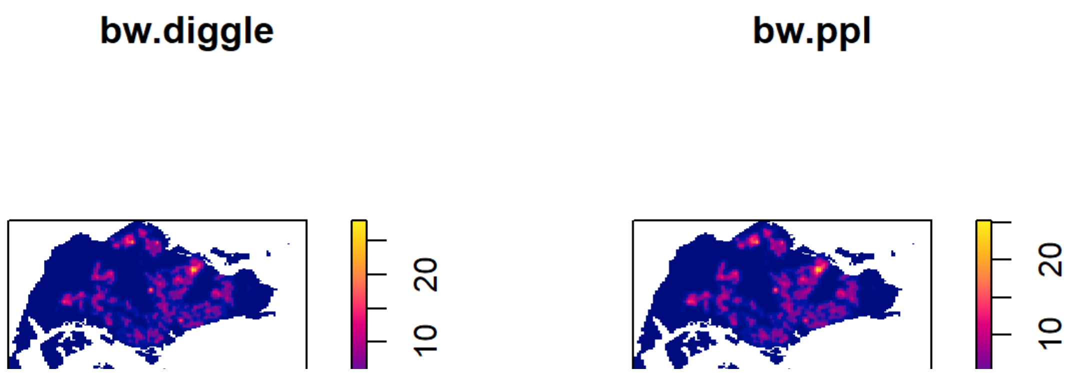
```
bw.diggle(childcareSG_ppp.km)
```

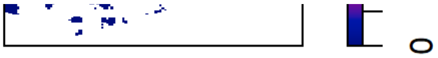
```
    sigma
0.2984095
```

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")
```



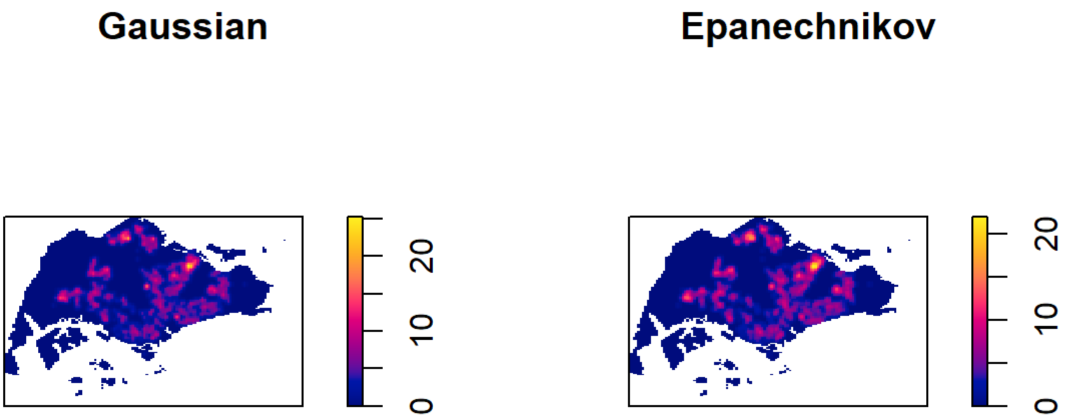


Working with different kernel methods

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

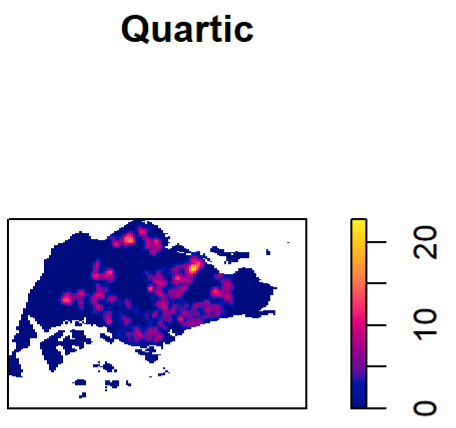
The code chunk below will be used to compute three more kernel density estimations by using these three kernel function.

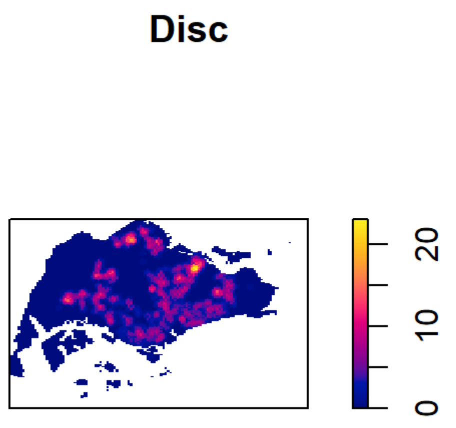
```
par(mfrow=c(1,2))
plot(density(childcareSG_ppp.km, sigma=bw.ppl, edge=TRUE, kernel="gaussian"), main="Gaussian")
plot(density(childcareSG_ppp.km, sigma=bw.ppl, edge=TRUE, kernel="epanechnikov"), main="Epanechnikov")
```




```
par(mfrow=c(1,2))
plot(density(childcareSG_ppp.km, sigma=bw.ppl, edge=TRUE, kernel="quartic"), main="Quartic")
plot(density(childcareSG_ppp.km, sigma=bw.ppl, edge=
```

```
plot ( density ( childcareSG_ppp.km, sigma= bw.ppl , edge=
TRUE , kernel= "disc" ) , main= "Disc" )
```



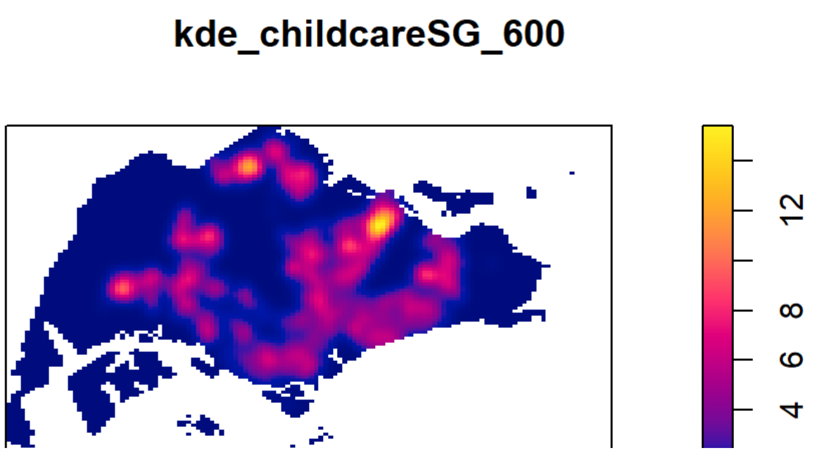


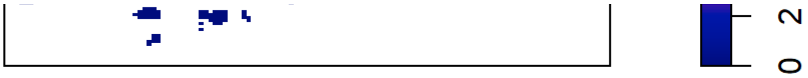
Fixed and Adaptive KDE

COMPUTING KDE BY USING FIXED BANDWIDTH

Next, you will compute a density map 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.

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



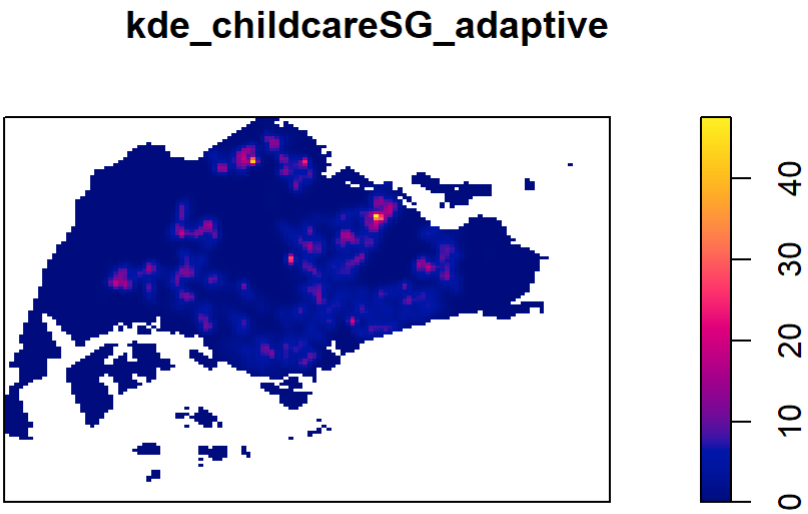


COMPUTING KDE BY USING ADAPTIVE BANDWIDTH

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.

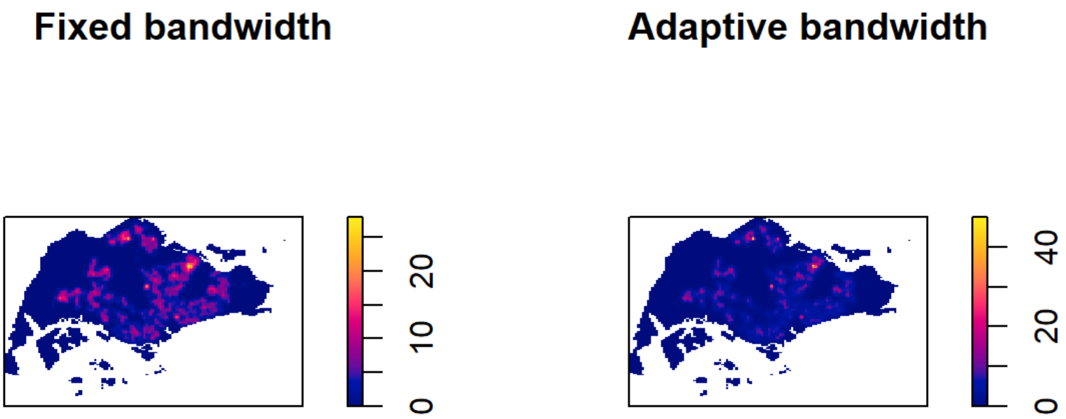
In this section, you will learn how to derive adaptive kernel density estimation by using *density.adaptive()* of **spatstat**.

```
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.

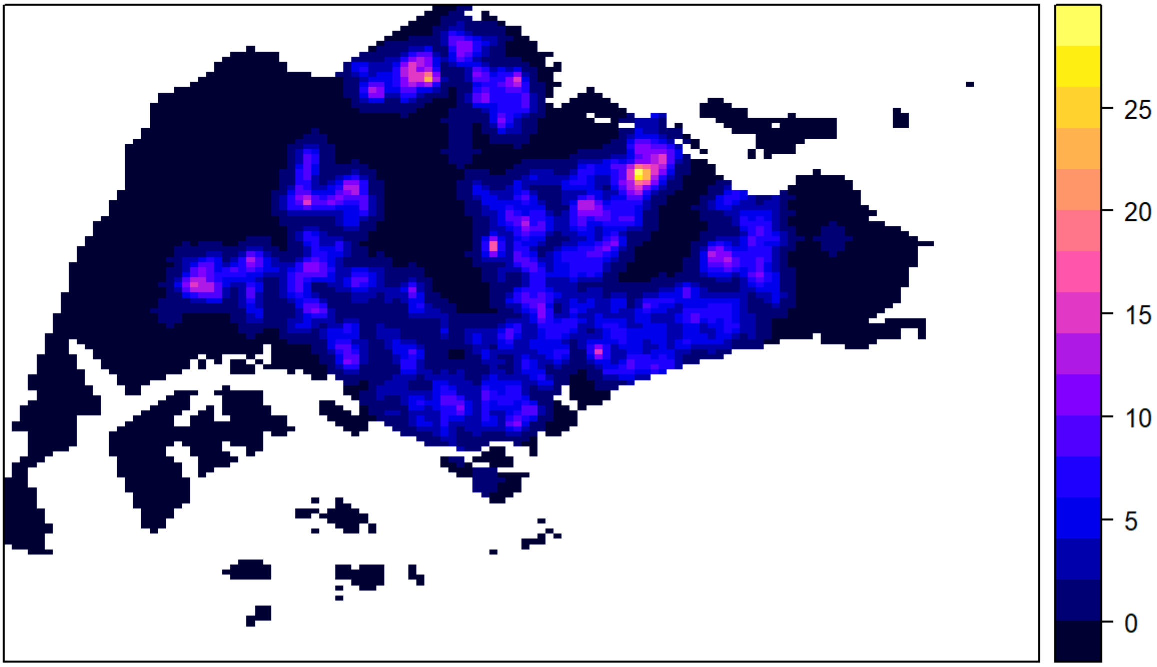
```
par(mfrow=c(1,2))
plot(kde_childcareSG.bw, main = "Fixed bandwidth")
plot(kde_childcareSG_adaptive, main = "Adaptive bandwidth")
```



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      : -6.052971e-15, 28.01036  (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=387
source      : memory
names       : v
values      : -6.052971e-15, 28.01036  (min, max)
```

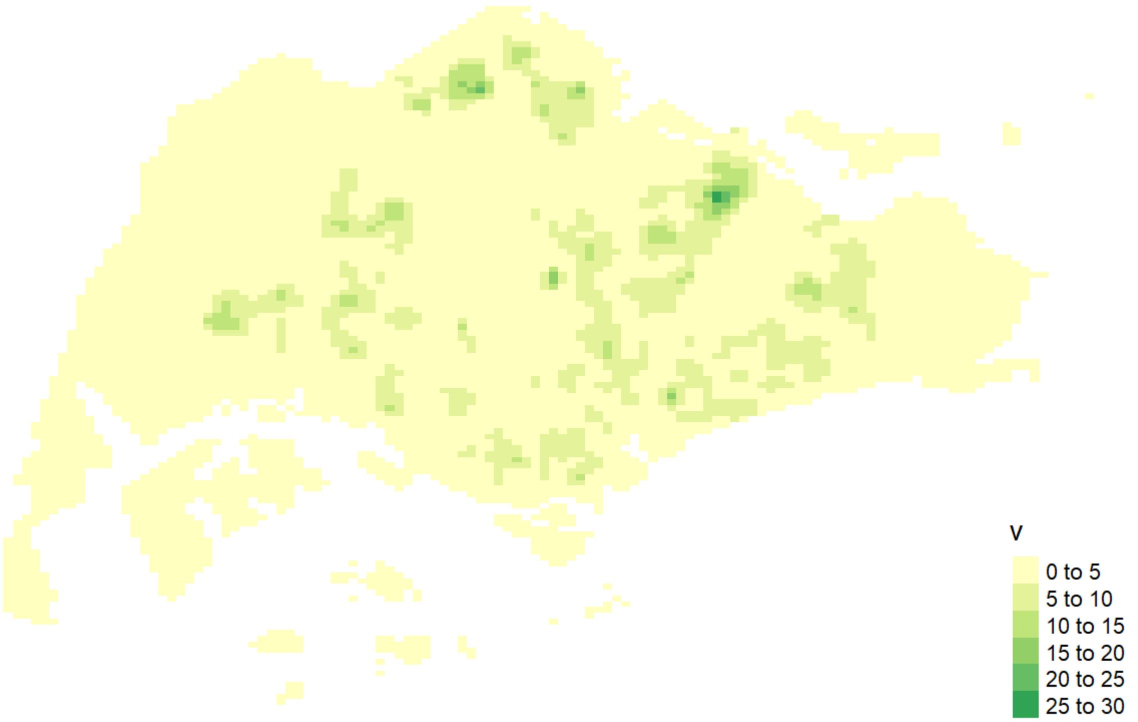
Notice that the crs property is completed.

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.

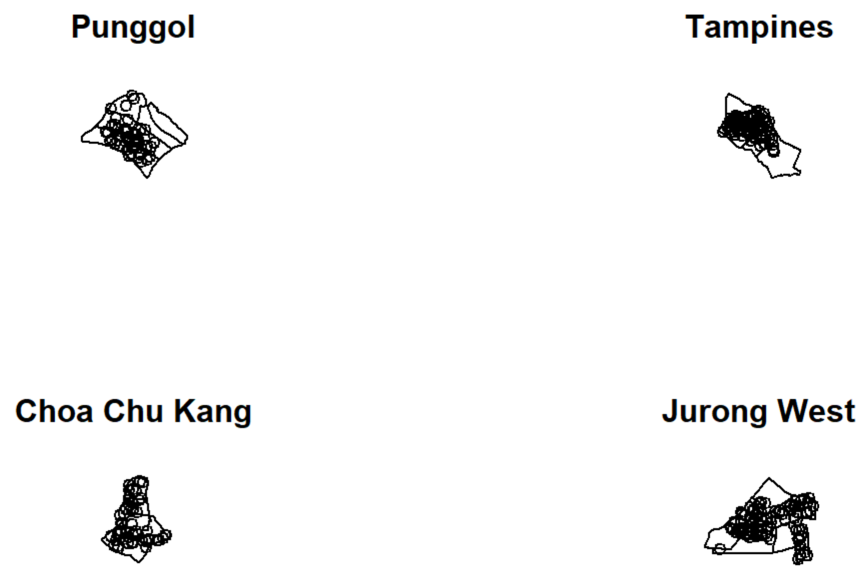
```
childcare_pg_ppp = childcare_ppp_jit[pg_owin]
childcare_tm_ppp = childcare_ppp_jit[tm_owin]
childcare_ck_ppp = childcare_ppp_jit[ck_owin]
childcare_jw_ppp = childcare_ppp_jit[jw_owin]
```

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")
```

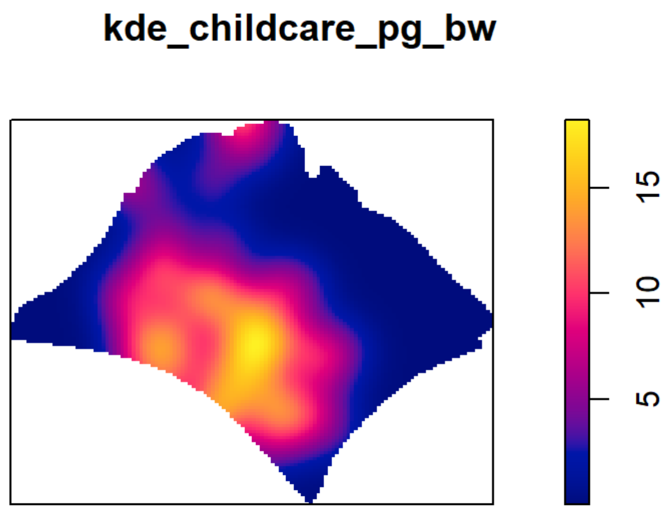


COMPUTING KDE

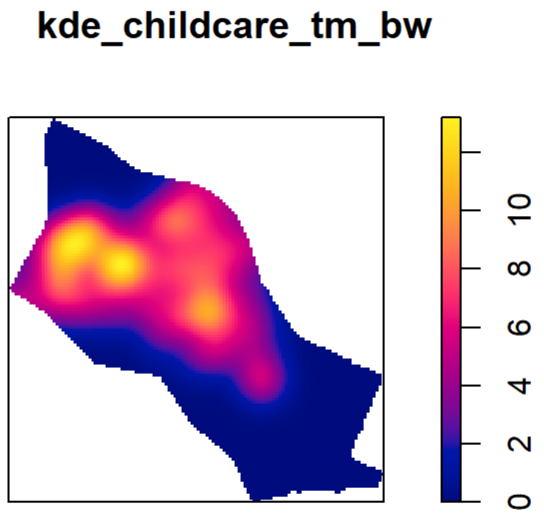
The code chunk below will be used to compute the KDE of these four planning area. ***bw.diggle*** method is used to derive the bandwidth of each

```
kde_childcare_pg_bw <- density(childcare_pg_ppp.km, sigma=bw.diggle, edge
= TRUE, kernel="gaussian")
```

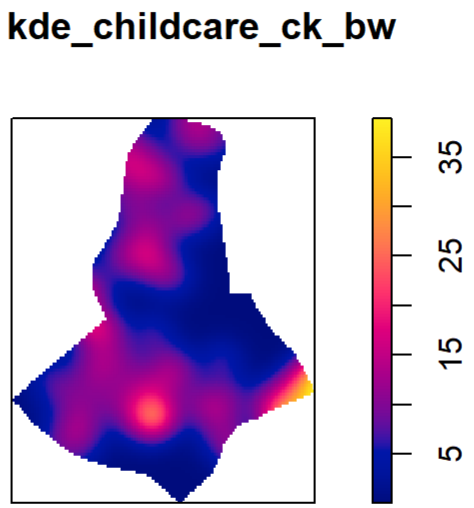
```
= TRUE , kernel= "gaussian")
plot ( kde_childcare_pg_bw)
```



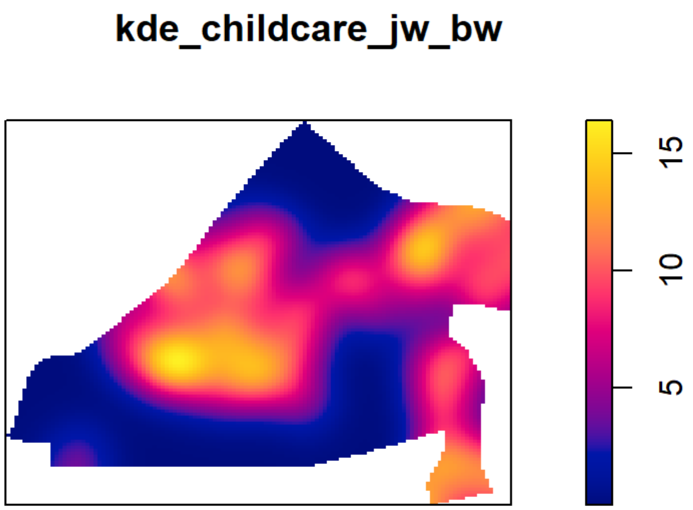

```
kde_childcare_tm_bw <- density ( childcare_tm_ppp.km, sigma= bw.diggle, edge
= TRUE , kernel= "gaussian")
plot ( kde_childcare_tm_bw)
```




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



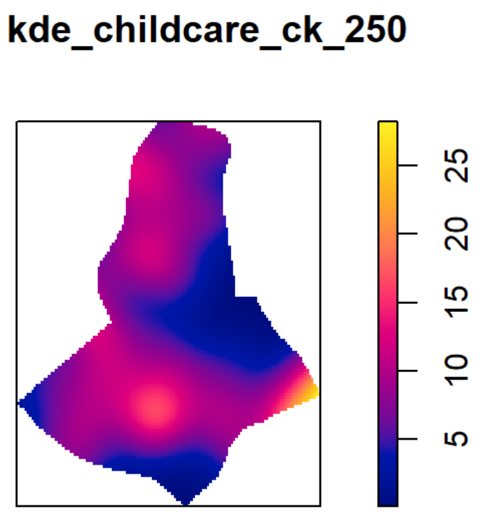

```
kde_childcare_jw_bw <- density(childcare_jw_ppp.km, sigma=bw.diggle, edge=TRUE, kernel="gaussian")
plot(kde_childcare_jw_bw)
```



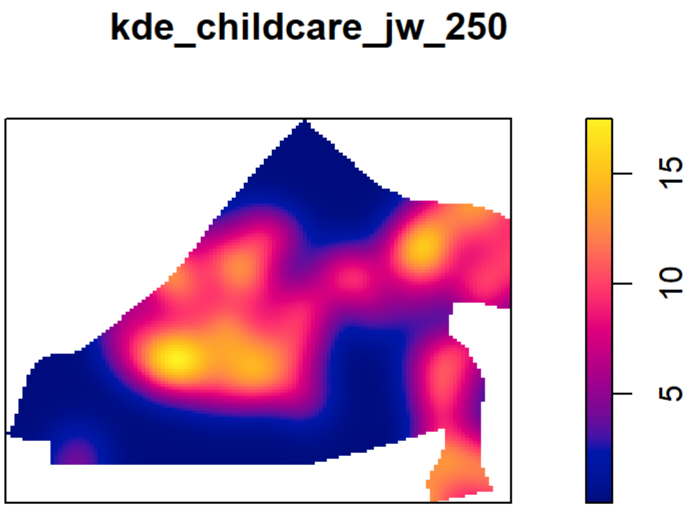
COMPUTING FIXED BANDWIDTH KDE

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

```
kde_childcare_ck_250 <- density(childcare_ck_ppp.km, sigma=0.25, edge=TRUE, kernel="gaussian")
plot(kde_childcare_ck_250)
```

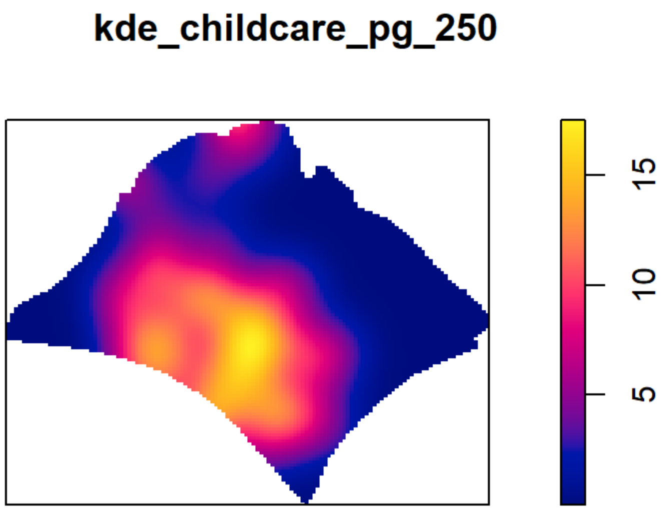



```
kde_childcare_jw_250 <- density(childcare_jw_ppp.km, sigma=0.25, edge=TRUE, kernel="gaussian")
plot(kde_childcare_jw_250)
```

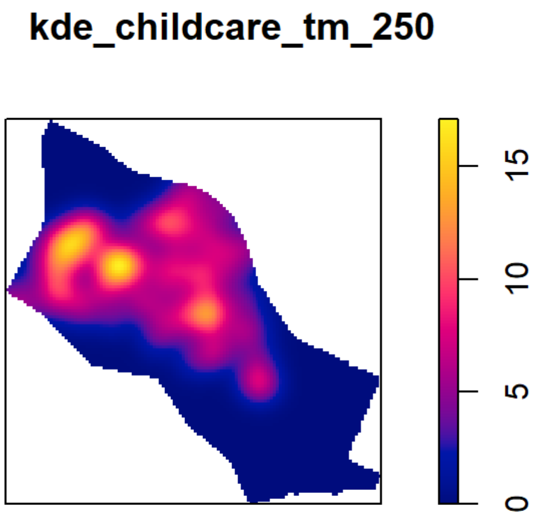



```
kde_childcare_pg_250 <- density(childcare_pg_ppp.km, sigma=0.25,
```

```
edge=TRUE, kernel="gaussian")
plot(kde_childcare_pg_250)
```




```
kde_childcare_tm_250 <- density(childcare_tm_ppp.km, sigma=0.25,
edge=TRUE, kernel="gaussian")
plot(kde_childcare_tm_250)
```



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 = 0.55696, 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
```

```
data:  childcare_ck_ppp
R = 0.9811, p-value = 0.3
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 = 0.7767, 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.

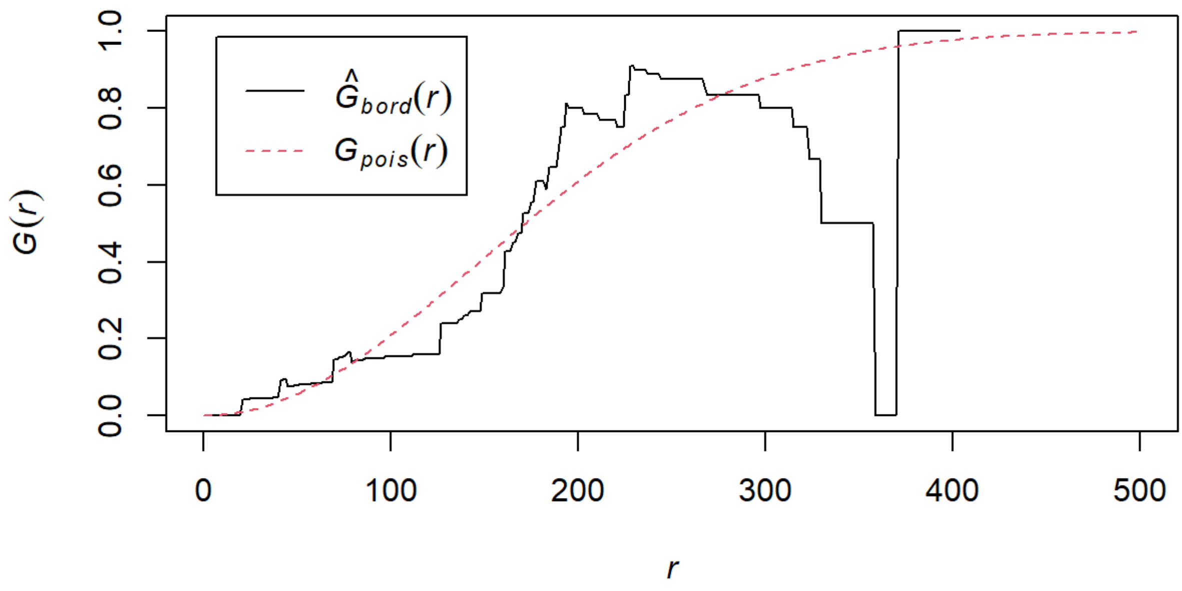
Choa Chu Kang planning area

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))
```

G_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 G-fucntion

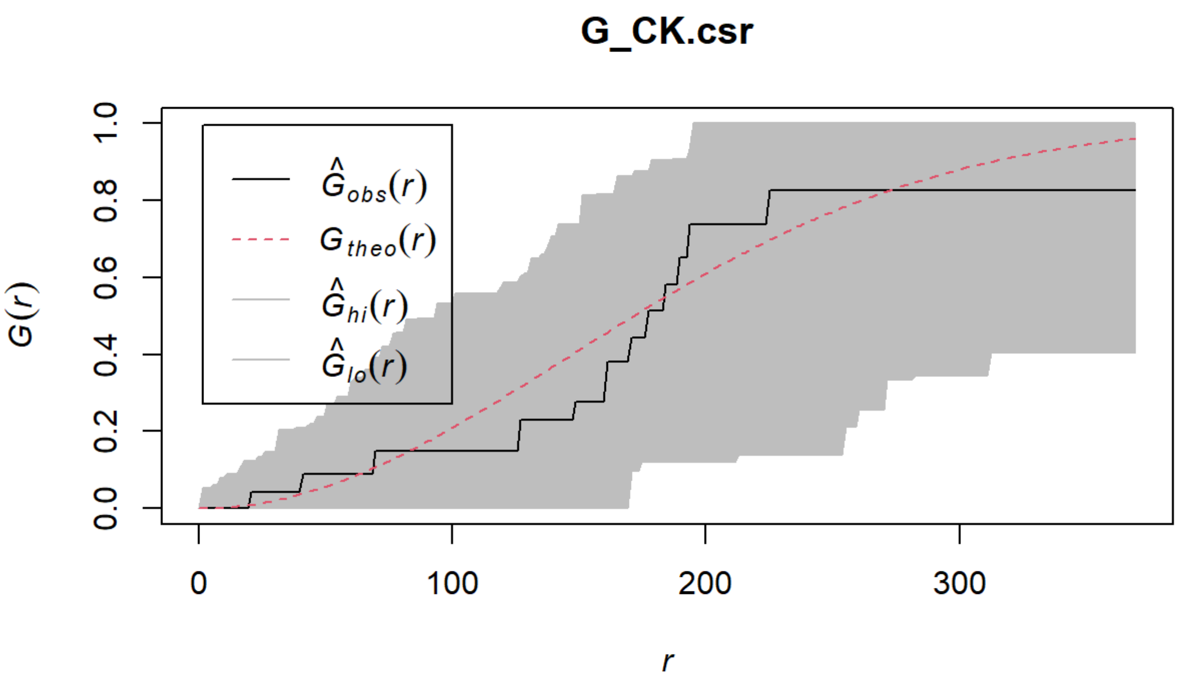
```
G_CK.csr  <-        envelope (        childcare_ck_ppp, Gest      , nsim =          999
)
```

```
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.
```

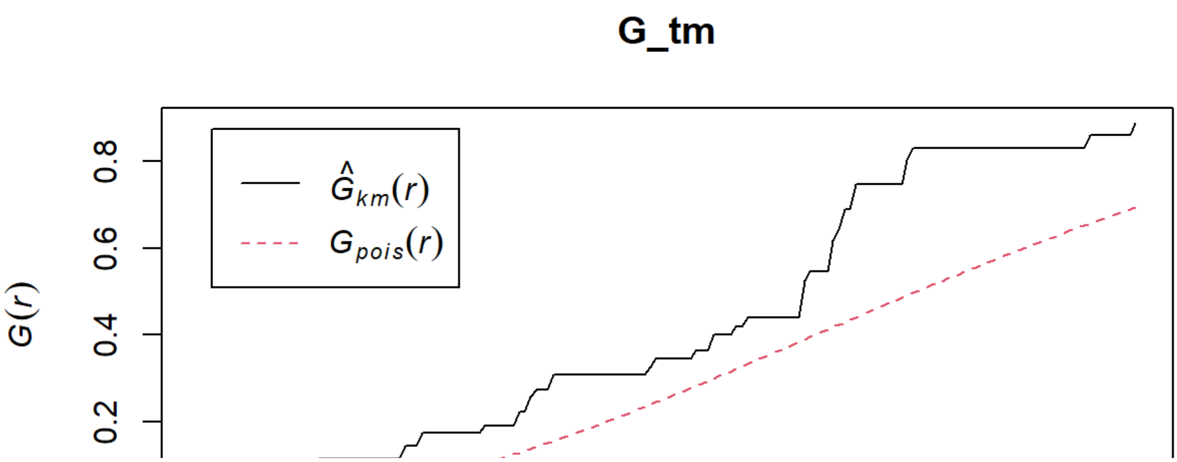
```
plot    (     G_CK.csr )
```

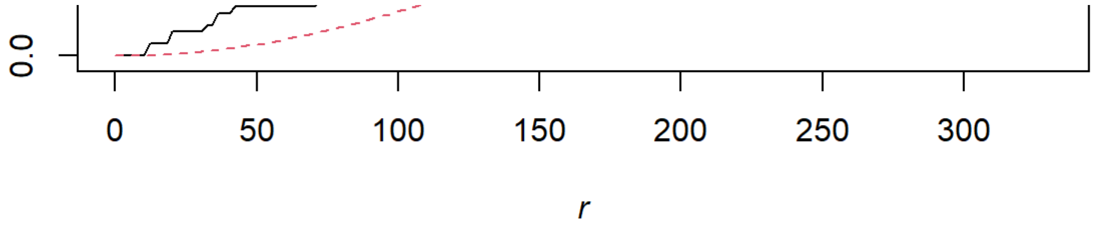


Tampines planning area

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.

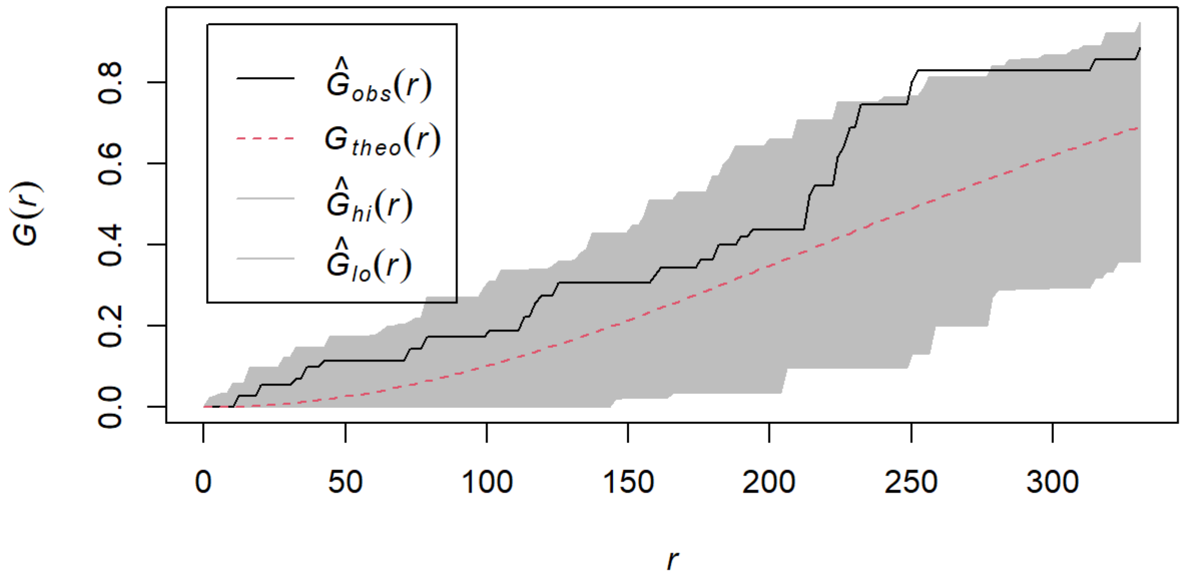
```
G_tm.csr <- envelope(childcare_tm_ppp, Gest, correction = "all", nsim = 999)
```

```
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)
```

G_tm.csr



Analysing Spatial Point Process Using F-Function

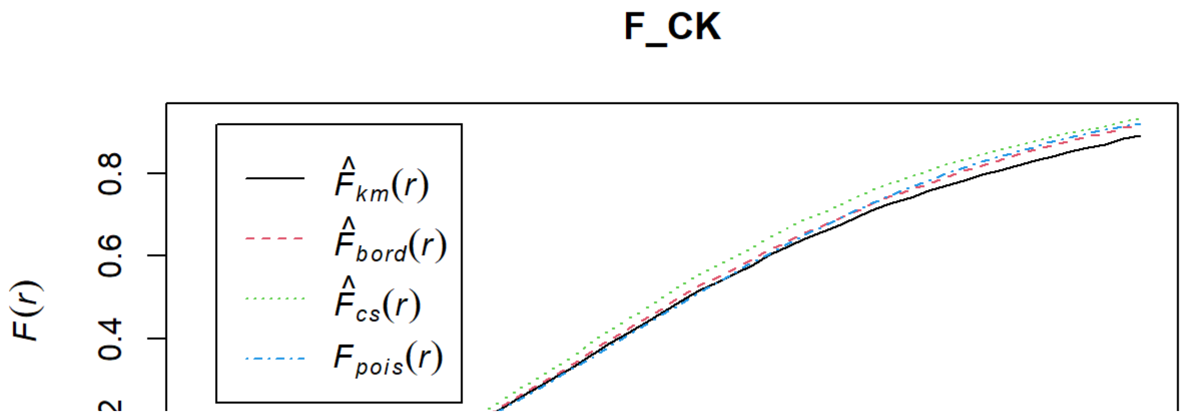
The F function estimates the empty space function F(r) or its hazard rate h(r) from a point pattern in a window of arbitrary shape. In this section, you will learn how to compute F-function estimation by using *Fest()* of **spatstat** package. You will also learn how to perform monta carlo simulation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

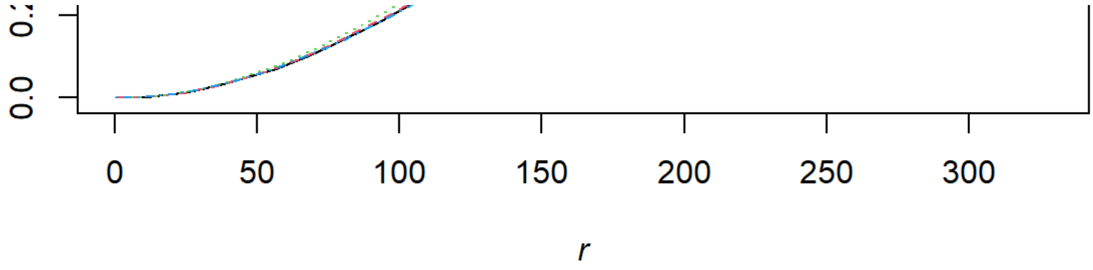
Choa Chu Kang planning area

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.

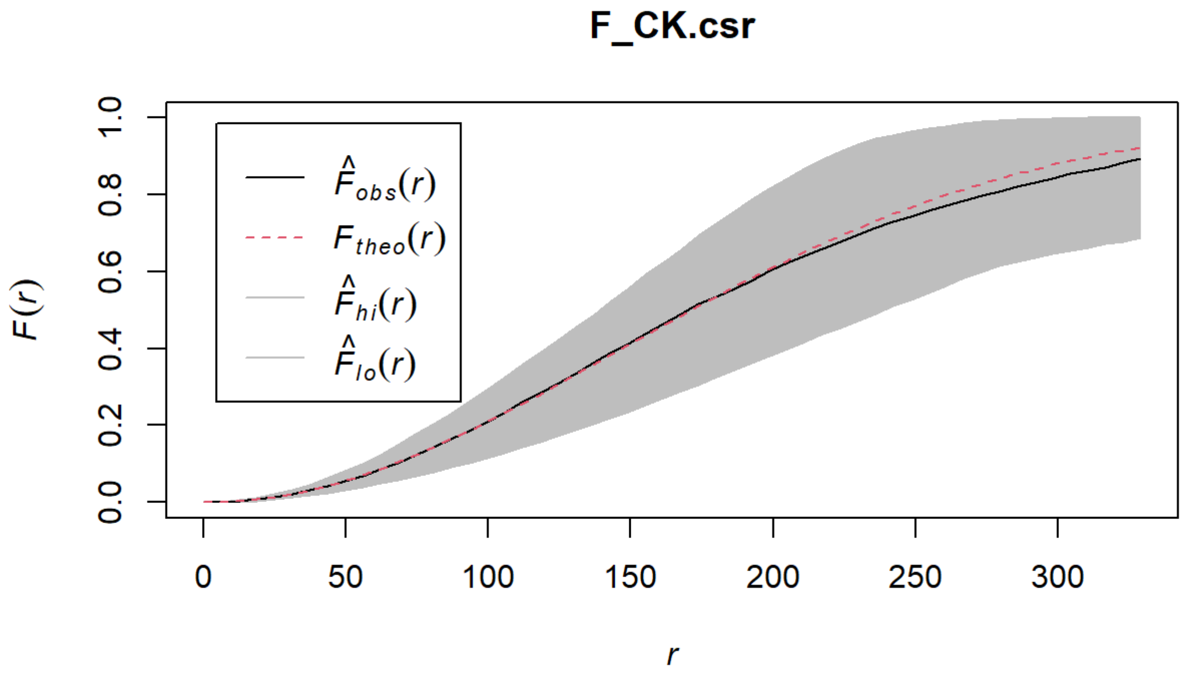
Monte Carlo test with F-fucntion

```
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)
```

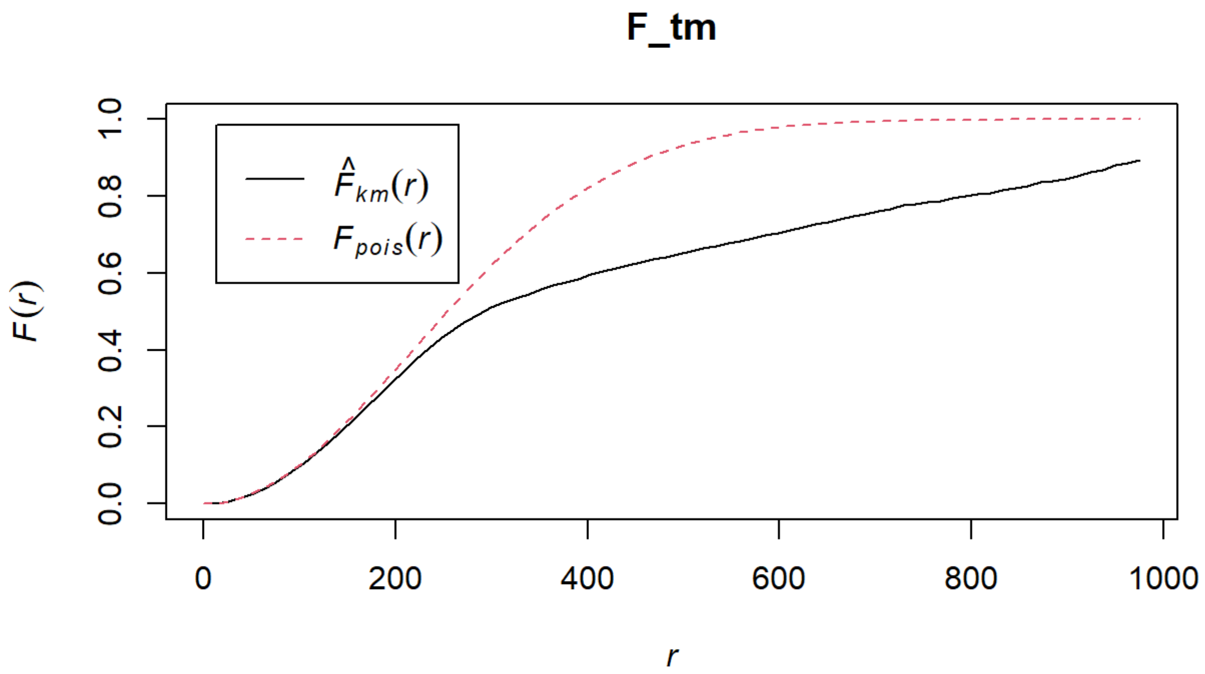


Tampines planning area

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.

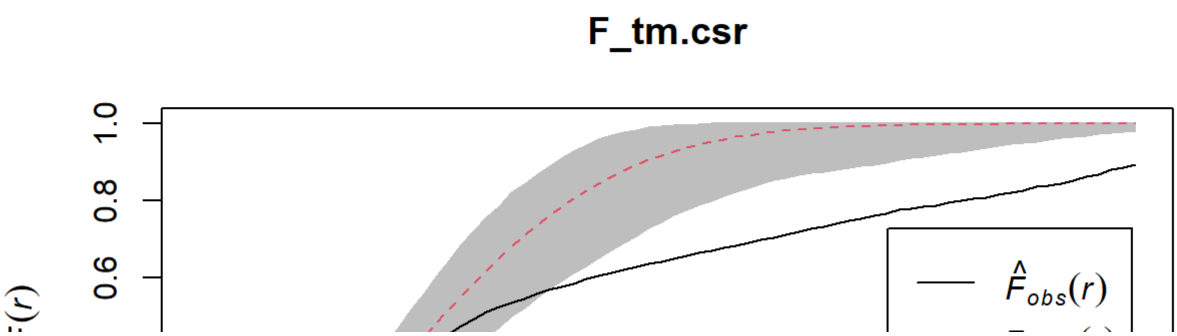
The code chunk below is used to perform the hypothesis testing.

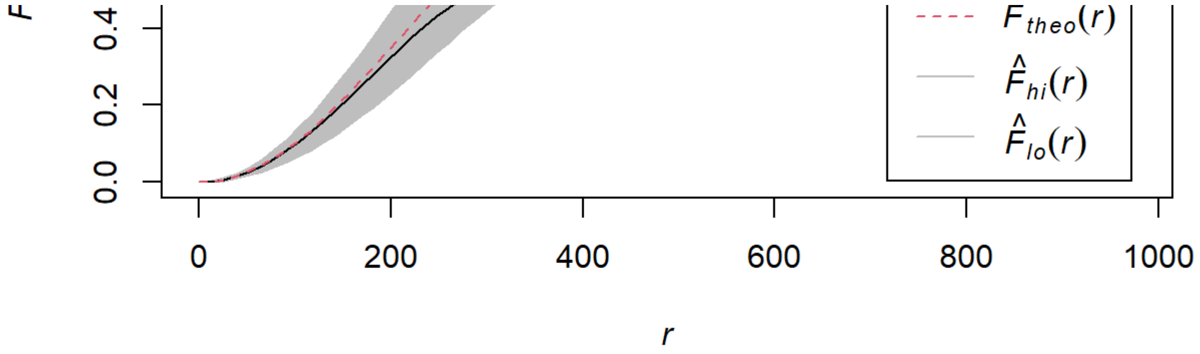
```
F_tm.csr <- envelope(childcare_tm_ppp, Fest, correction = "all", nsim = 999)
```

```
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)
```





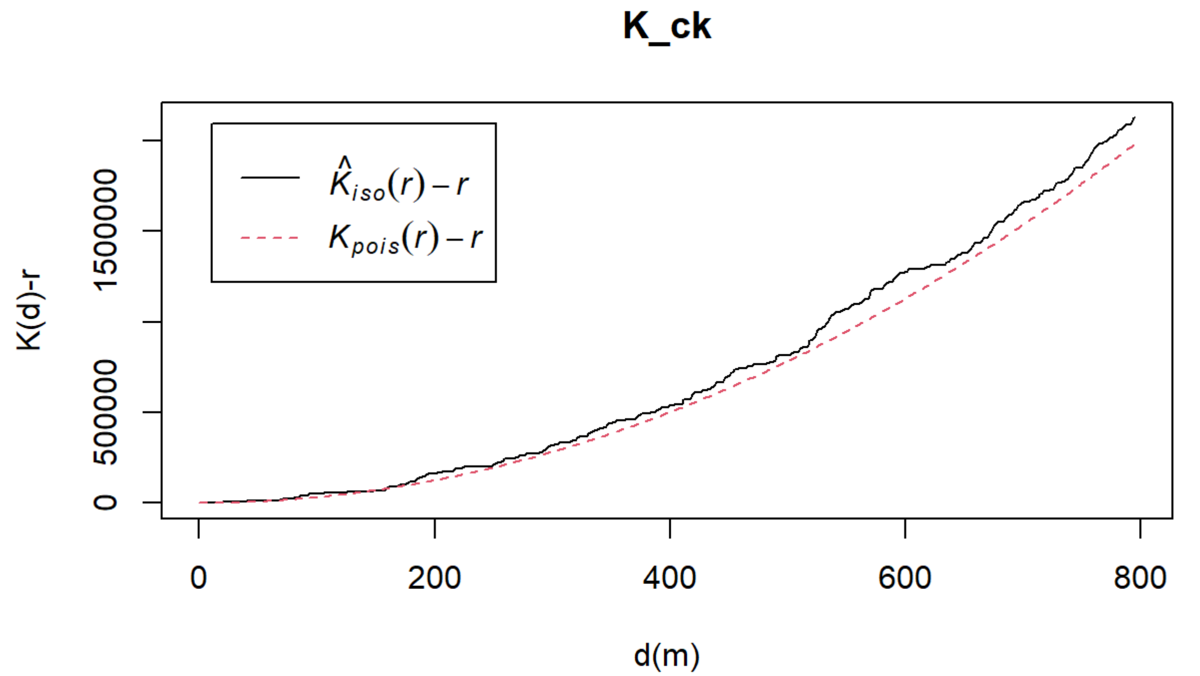
Analysing Spatial Point Process Using K-Function

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.

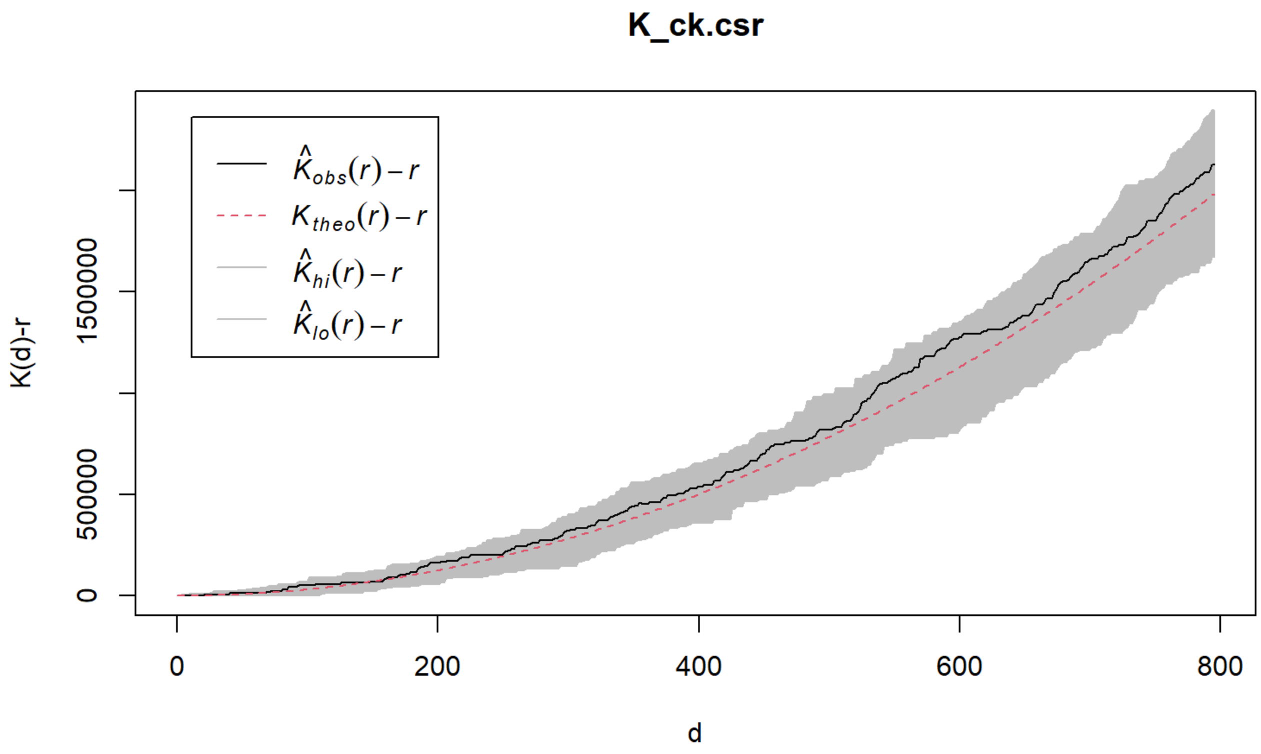
The code chunk below is used to perform the hypothesis testing.

```
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, 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.
```

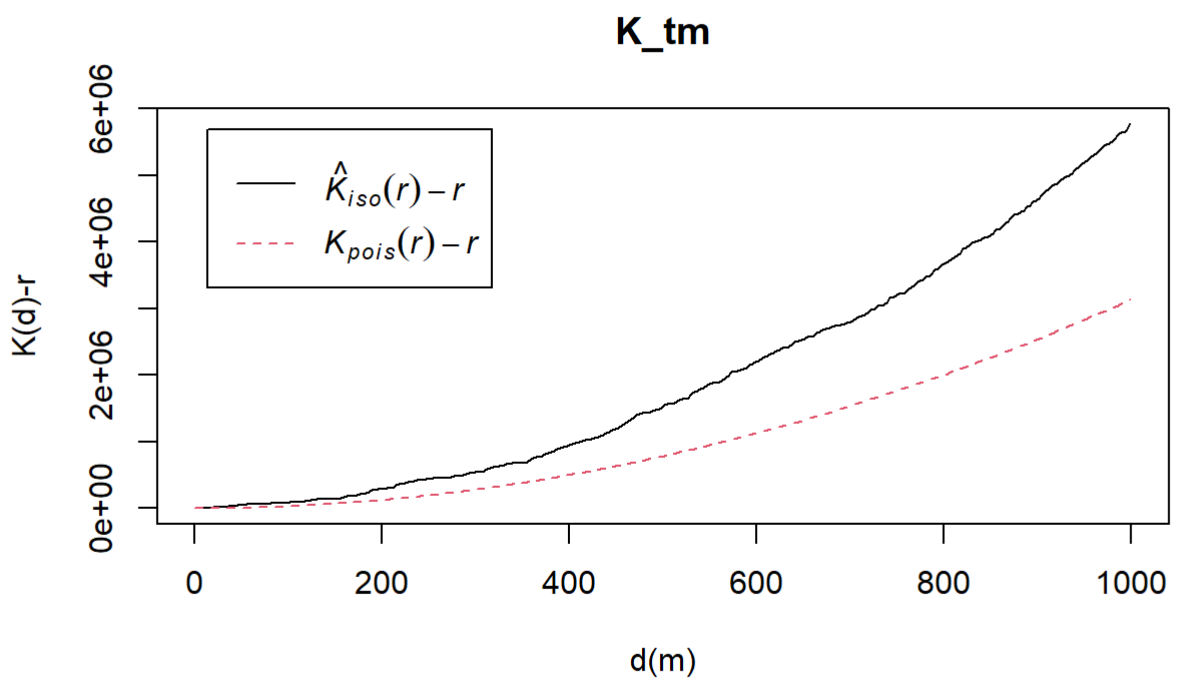
```
plot(K_ck.csr, . - r ~ r, xlab="d", ylab="K(d)-r")
```



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.

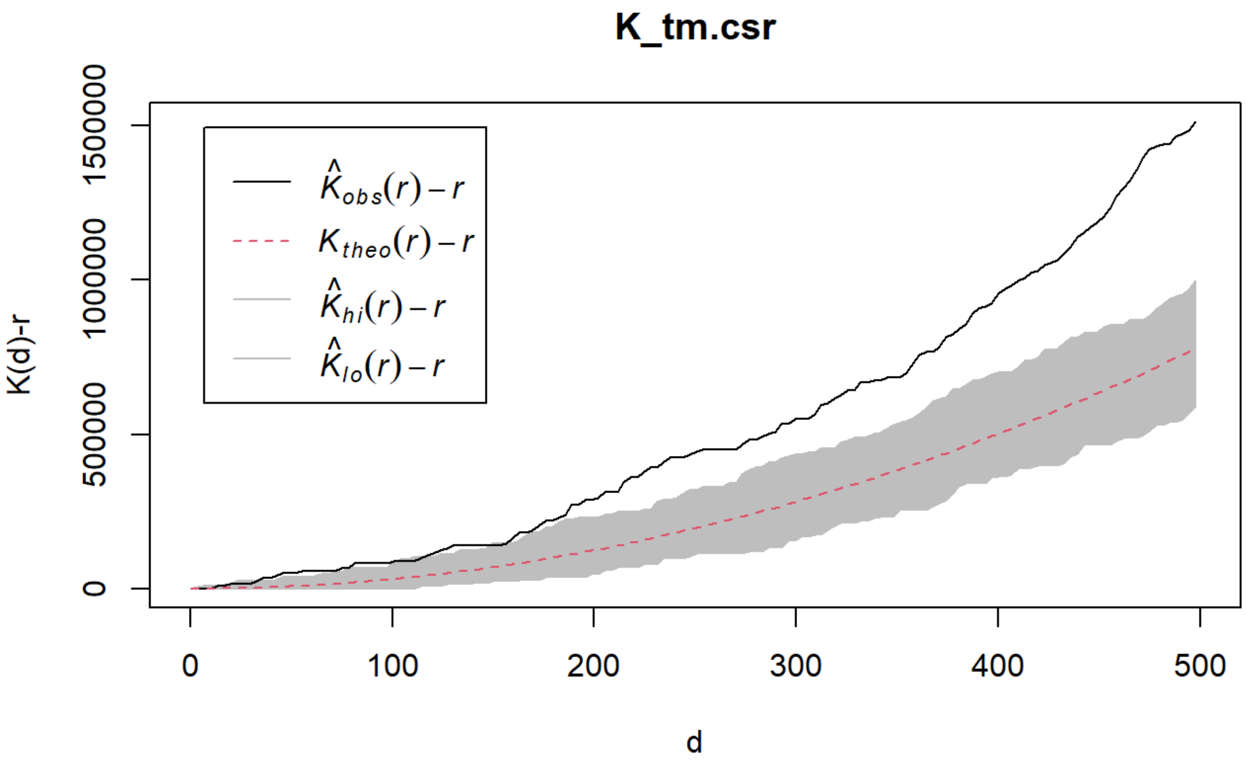
```
K_tm.csr <- envelope ( childcare_tm_ppp, Kest , nsim = 99 , rank
= 1 , glocal= TRUE )
```

```
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.
```

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



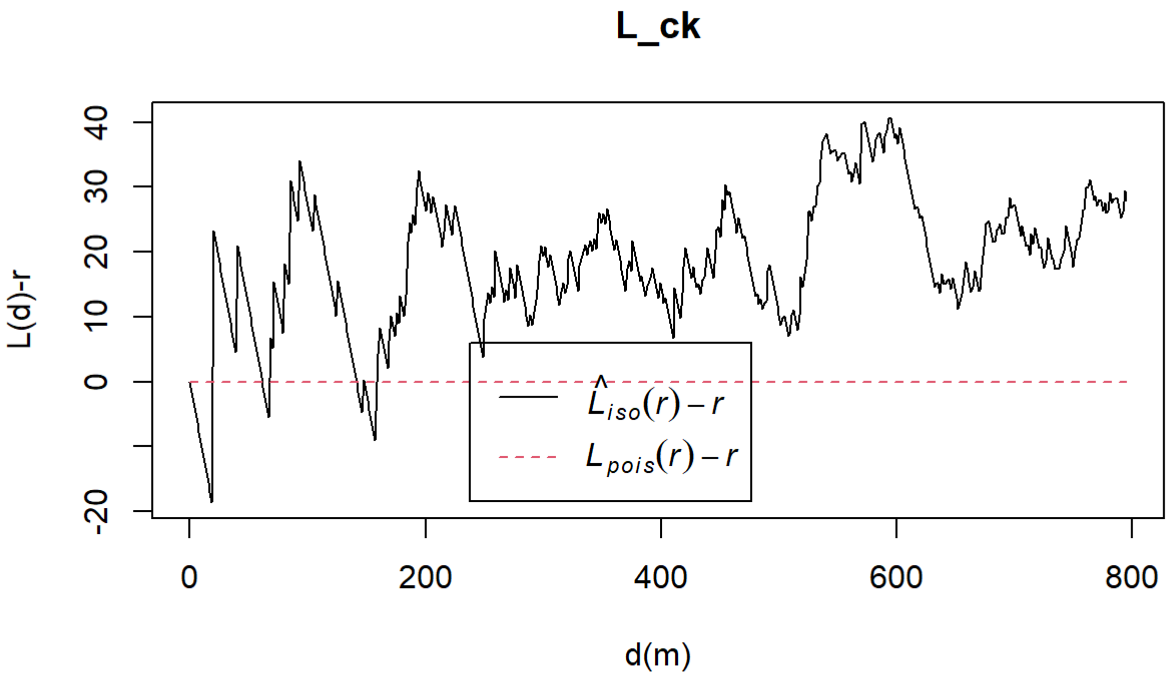
Analysing Spatial Point Process Using L-Function

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

```r
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.

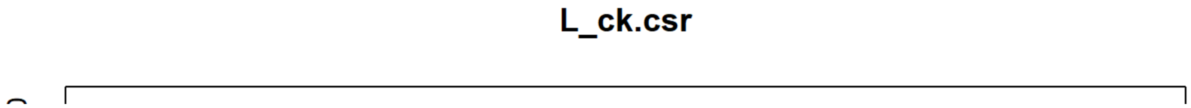
The code chunk below is used to perform the hypothesis testing.

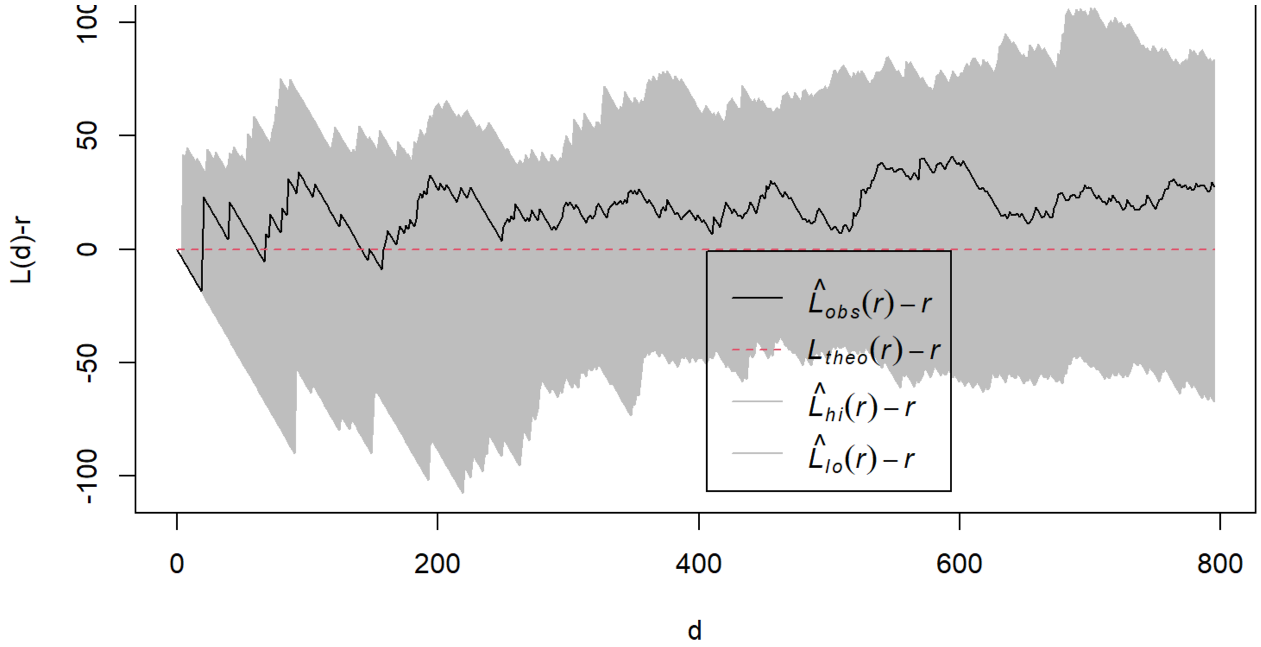
```
L_ck.csr <- envelope(childcare_ck_ppp, Lest, nsim = 99, rank = 1, glocal=TRUE)
```

```
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")
```

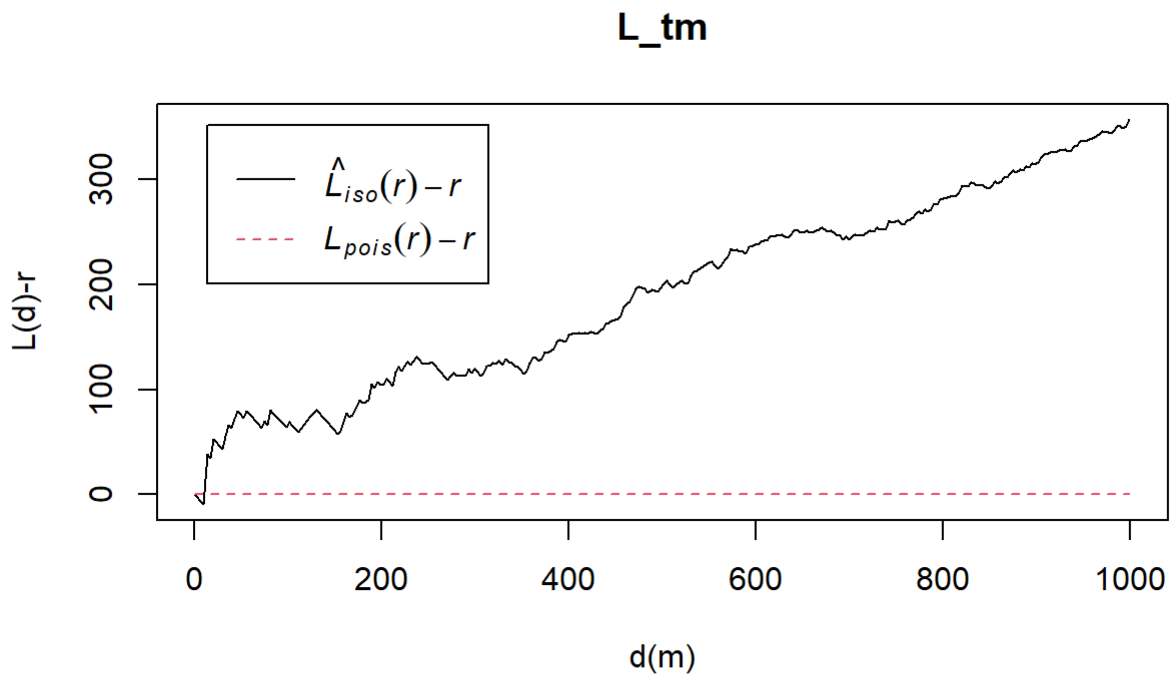




Tampines planning area

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.

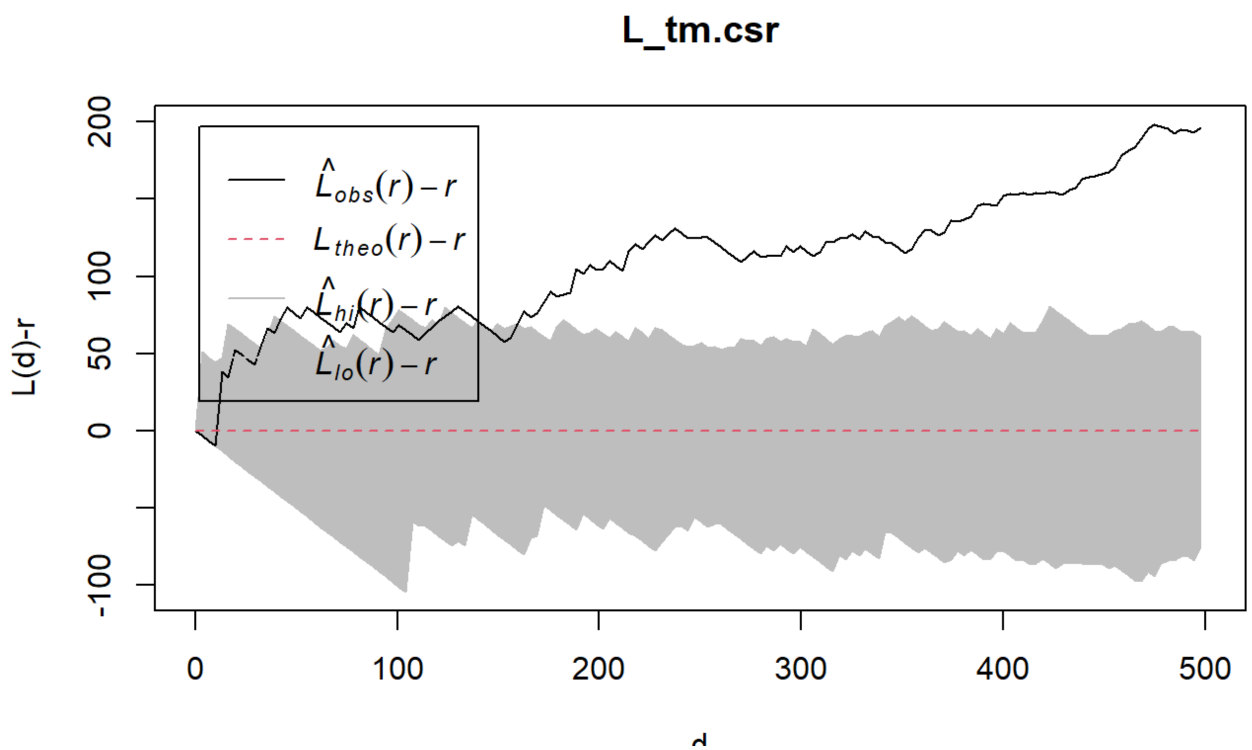
```
L_tm.csr <- envelope(childcare_tm_ppp, Lest, nsim = 99, rank = 1, glocal=TRUE)
```

```
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))
```



```r
library(pagedown)
pagedown::chrome_print("Hands-on_Ex04_SPPA_spatstat.html")
```