Chapter 2

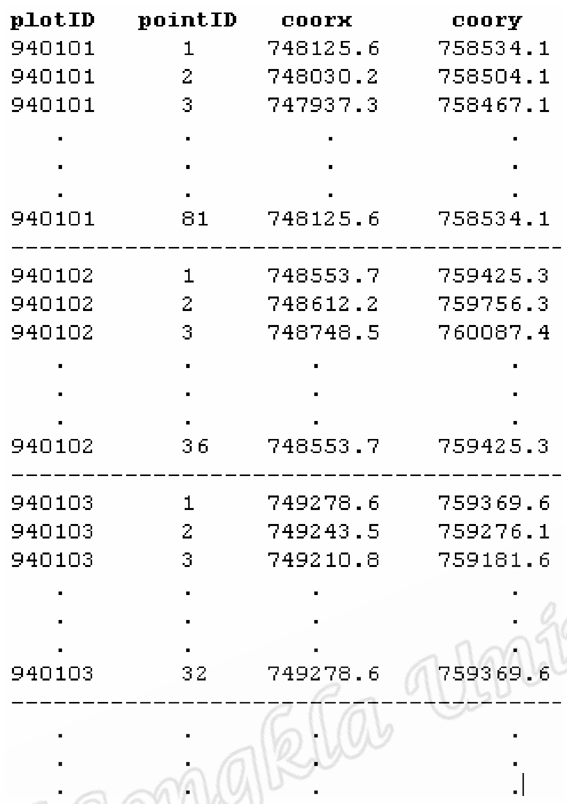
Methodology

In this chapter we will describe in three sections. In the first section, we describe the format of input file that is needed for using functions. The second, we describe the theory used in programming. The last section, we describe the methodology of each function, which include objective of each function, function syntax, algorithm and example of the result from function.

2.1 The format of input file

The functions must have at least two files to create the map and display the information, namely a spatial data file and an attribute data file. Each file must contain a primary key, which must be composite, to uniquely identify each record. The spatial data file and the attribute data file are described as below:

Spatial data file contains the Cartesian coordinates system. Figure 2.1 shows an example of a spatial data file. It contains two columns *plotID* and *pointID*, representing the primary key, and *coorx* and *coory*, representing the *x*- and *y*-coordinates, respectively. The column called *plotID* represents the region code while the *pointID* represents a sequential index of coordinates in each region. The fields must be ordered by *plotID*, *pointID*, *coorx* and *coory*. However the names of fields depend on the user.



```
plotID    pointID    coorx        coory
940101       1      748125.6    758534.1
940101       2      748030.2    758504.1
940101       3      747937.3    758467.1
   .         .          .           .
   .         .          .           .
   .         .          .           .
940101      81      748125.6    758534.1
----------------------------------------
940102       1      748553.7    759425.3
940102       2      748612.2    759756.3
940102       3      748748.5    760087.4
   .         .         .            .
   .         .         .            .
   .         .         .            .
940102      36      748553.7    759425.3
----------------------------------------
940103       1      749278.6    759369.6
940103       2      749243.5    759276.1
940103       3      749210.8    759181.6
   .         .         .            .
   .         .         .            .
   .         .         .            .
940103      32      749278.6    759369.6
----------------------------------------
   .         .         .            .
   .         .         .            .
   .         .         .            .
```

Figure 2.1: An example of a spatial data file

The spatial data file has two types that are simple region and complex region. *Simple region* is area that doesn't have a hole. For example the figure 2.2 has five simple regions, which we will call A, B, C, D and E.

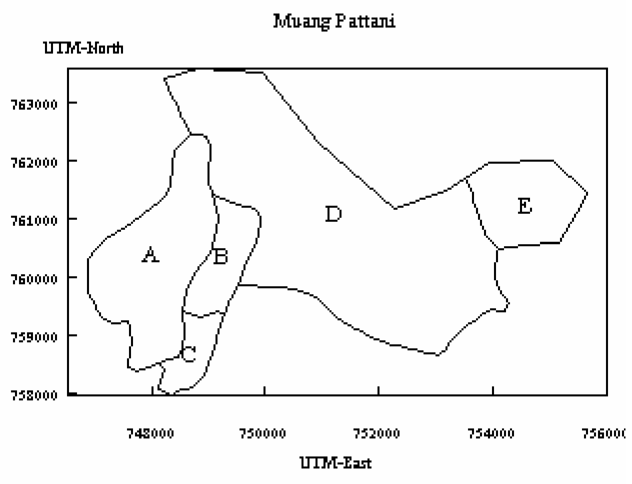
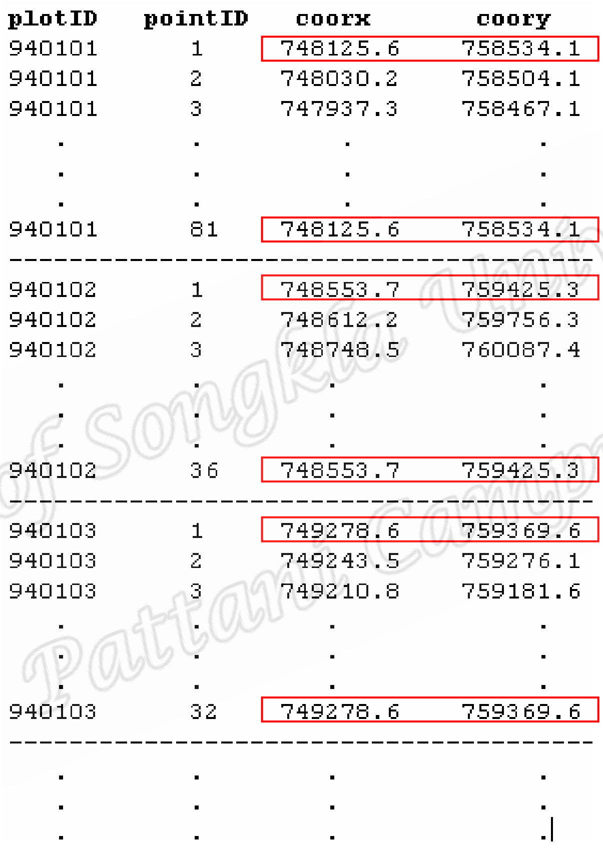


Figure 2.2: Simple regions

If we look in the example text file (figure 2.3) it can be seen that each region has one duplicated pair of *x*- and *y*- coordinates that is a polygon. The region of "*940101*" start and end with *x* = *748125.6* and *y* = *758534.1*, the region "*940102*" start and end with *x* = *748553.7* and *y* = *759425.3*, and the region "*940103*" start and end with *x* = *749278.6* and *y* = *759369.6*.



```
plotID    pointID    coorx        coory
940101       1      748125.6     758534.1
940101       2      748030.2     758504.1
940101       3      747937.3     758467.1
   .         .         .            .
   .         .         .            .
   .         .         .            .
940101      81      748125.6     758534.1
---------------------------------------
940102       1      748553.7     759425.3
940102       2      748612.2     759756.3
940102       3      748748.5     760087.4
   .         .         .            .
   .         .         .            .
   .         .         .            .
940102      36      748553.7     759425.3
---------------------------------------
940103       1      749278.6     759369.6
940103       2      749243.5     759276.1
940103       3      749210.8     759181.6
   .         .         .            .
   .         .         .            .
   .         .         .            .
940103      32      749278.6     759369.6
---------------------------------------
   .         .         .            .
   .         .         .            .
   .         .         .            .
```

Figure 2.3: Text file containing *x*- and *y*- coordinates of simple regions

Complex region is area that has one or more holes, or an area is interspersed with smaller polygons. As an example, figure 2.4 has two regions in a place, which we call A and B.

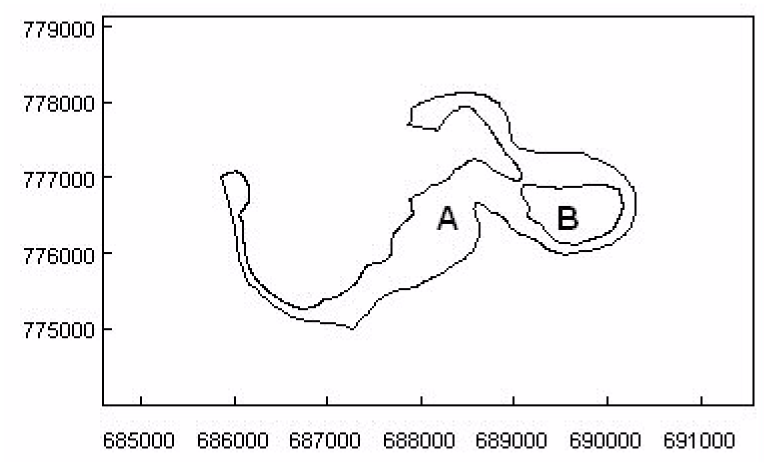
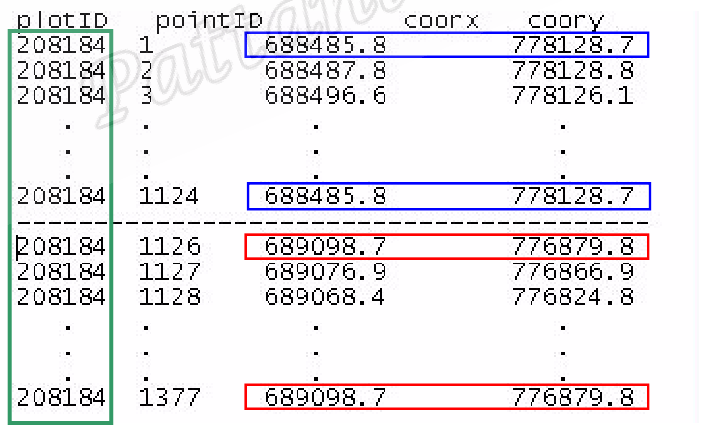


Figure 2.4: Complex regions

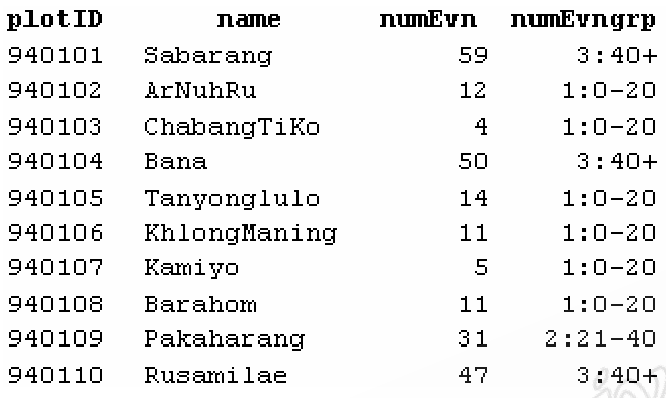
If we look in file format, it can be seen that for one region there are two pairs or more of duplicated *x*- and *y*- coordinates,as show n in figure 2.5. The region of "*208184*" has two polygons, the first of polygon start and end with *x = 688485.8* and *y = 778128.7* and the second of polygon start and end with *x = 689098.7* and *y = 776879.8*



```
plotID    pointID          coorx      coory
208184    1        688485.8           778128.7
208184    2        688487.8           778128.8
208184    3        688496.6           778126.1
   .      .           .                  .
   .      .           .                  .
   .      .           .                  .
208184    1124     688485.8           778128.7
-------------------------------------------------
208184    1126     689098.7           776879.8
208184    1127     689076.9           776866.9
208184    1128     689068.4           776824.8
   .      .           .                  .
   .      .           .                  .
   .      .           .                  .
208184    1377     689098.7           776879.8
```

Figure 2.5: Text file containing *x*- and *y*- coordinates of complex regions

Attribute data files contain statistical data. Figure 2.6 shows an example of an attribute data file. In this file *plotID* is the primary key, representing the region code. The column called *name* is the name of the region. The *numEvn* and *numEvngrp* columns are the variables to display on the map. In this example, *numEvn* is the number of terrorist events in each region, while *numEvngrp* is the same data categorized into groups.



plotID	name	numEvn	numEvngrp
940101	Sabarang	59	3:40+
940102	ArNuhRu	12	1:0-20
940103	ChabangTiKo	4	1:0-20
940104	Bana	50	3:40+
940105	Tanyonglulo	14	1:0-20
940106	KhlongManing	11	1:0-20
940107	Kamiyo	5	1:0-20
940108	Barahom	11	1:0-20
940109	Pakaharang	31	2:21-40
940110	Rusamilae	47	3:40+

Figure 2.6: Example of an attribute data file

2.2 Theory related

Location reference systems for spatial features on the Earth's surface use a coordinates system. Some maps are measured in longitude and latitude values, called a geographic grid. Other maps are measured in two dimensional coordinates, which are called a plane coordinate system or Cartesian coordinate system. Longitude values are similar to *x* values in a coordinate system and latitude value are similar to *y* values. The geographic grid consists of meridians and parallels. The meridians are lines of longitude for the East-West direction. The parallels are lines of latitude for the North-South direction. The plane coordinate systems are designed for detailed calculations and positioning. Scales of measurement can vary, depending on the level of detail and precision required. Four coordinates systems are commonly used in the United States, with measurements varying between meters, feet and miles. In this study a spatial data file that contains the Cartesian coordinates system was used (Chang, 2002).

The theory that the researcher used to develop functions is described as below:
To calculate perimeter, suppose we are given the two points that are (x_1, y_1) and (x_2, y_2), as shown in figure 2.7.

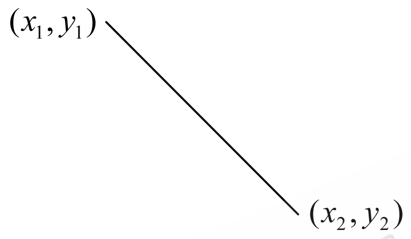


Figure 2.7: A line between two points

From the standard Pythagorean theorem for relating the parts of a triangle, we can compute the distance between these points, which is given by the formula in figure 2.8.

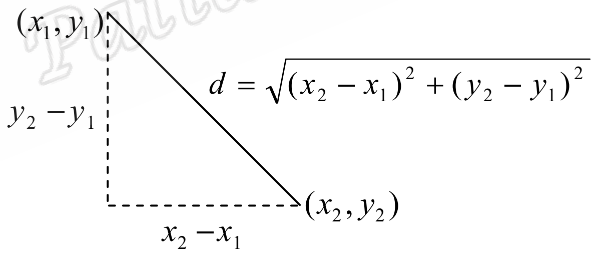
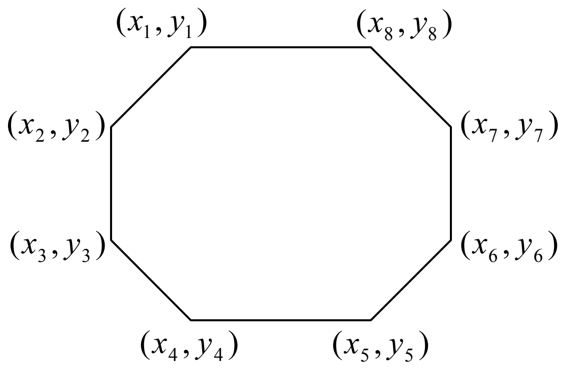


Figure 2.8: The Pythagorean theorem

where d is the distance, x_2-x_1 is difference in the x direction, y_2-y_1 is difference in the y direction. We can compute the perimeter by aggregating the distances between every pair of points. This formula isused for a coordinate system based on a projection, such as the Universal Transverse Mercator (UTM), State Plane or United Kingdom National Grid. It will not work for latitude and longitude.

For computing the area of a polygon the formula in figure 2.9 was used.



$$A = \frac{1}{2}\sum_{i=0}^{N-1}(x_i y_{i+1} - x_{i+1} y_i)$$

Figure 2.9: A polygon

where *A* is the area, *i* is a index for every Cartesian coordinate, and *N* is the total number of coordinates, for the center of a polygon the formula below was used:

$$C_x = \frac{1}{6A}\sum_{i=0}^{N-1}(x_i + x_{i+1})(x_i y_{i+1} - x_{i+1} y_i),\ C_y = \frac{1}{6A}\sum_{i=0}^{N-1}(y_i + y_{i+1})(x_i y_{i+1} - x_{i+1} y_i)$$

Where C_x is the center of *x* and C_y is the center of *y*, *A* is the area, *i* is an index for every Cartesian coordinate and *N* is the total number of coordinates (Bourke, 1988). These formulas also can apply to computing for triangle (figure 2.10) and square (figure 2.11).

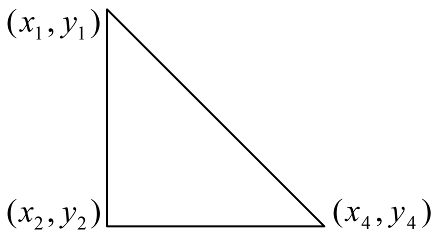


Figure 2.10: A triangle

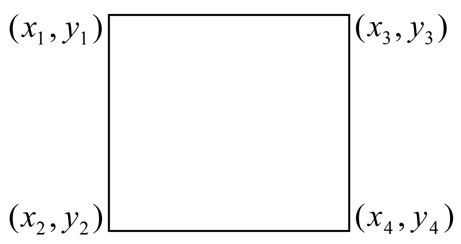


Figure 2.11: A square

For managing the integer variable into groups by using Frequency Distribution of Grouped Data, as the first step we calculated a range of data. The range of data is the difference in value between a maximum value and a minimum value.

$$r = Max\text{-}Min$$

Where r is a range value, Max is a maximum value and Min is a minimum value. For the second step, we computed a number of class from this formula:

$$k = 1+3.3logN$$

Where k is a number of class and N is a number of data. The third step calculates a class interval value.

$$i = r/k$$

Where i is class interval value, r is a range value and k is a number of class. The class interval must be the integer. If the value of class interval has a decimal, it must be rounded up in value, whether it is less or more than 0.5. If the value of class interval is an integer, it must be that integer plus one. The last step counts the number in each class. The lower bound of the first class must cover the minimum value. The upper bound of the last class must cover the maximum value (Hanmongkolpipat, 2003).

2.3 Methodology of each function

In this study we created 10 functions. They can be divided to three groups. The first group being functions to manage regions, there are five functions, namely *create.map*(), *setcol.map*(), *setcol.cmap*(), *setnme.map*() and *combine.map*(). The second group contains functions to show statistics data, there are two functions which are *colstat.map*() and *piestat.map*(). The third group containsfunctions to compute area, perimeter and center of region, and these are *area.map*(), *perimeter.map*() and *center.map*(). Figure 2.12 shows chart of functions that are created.

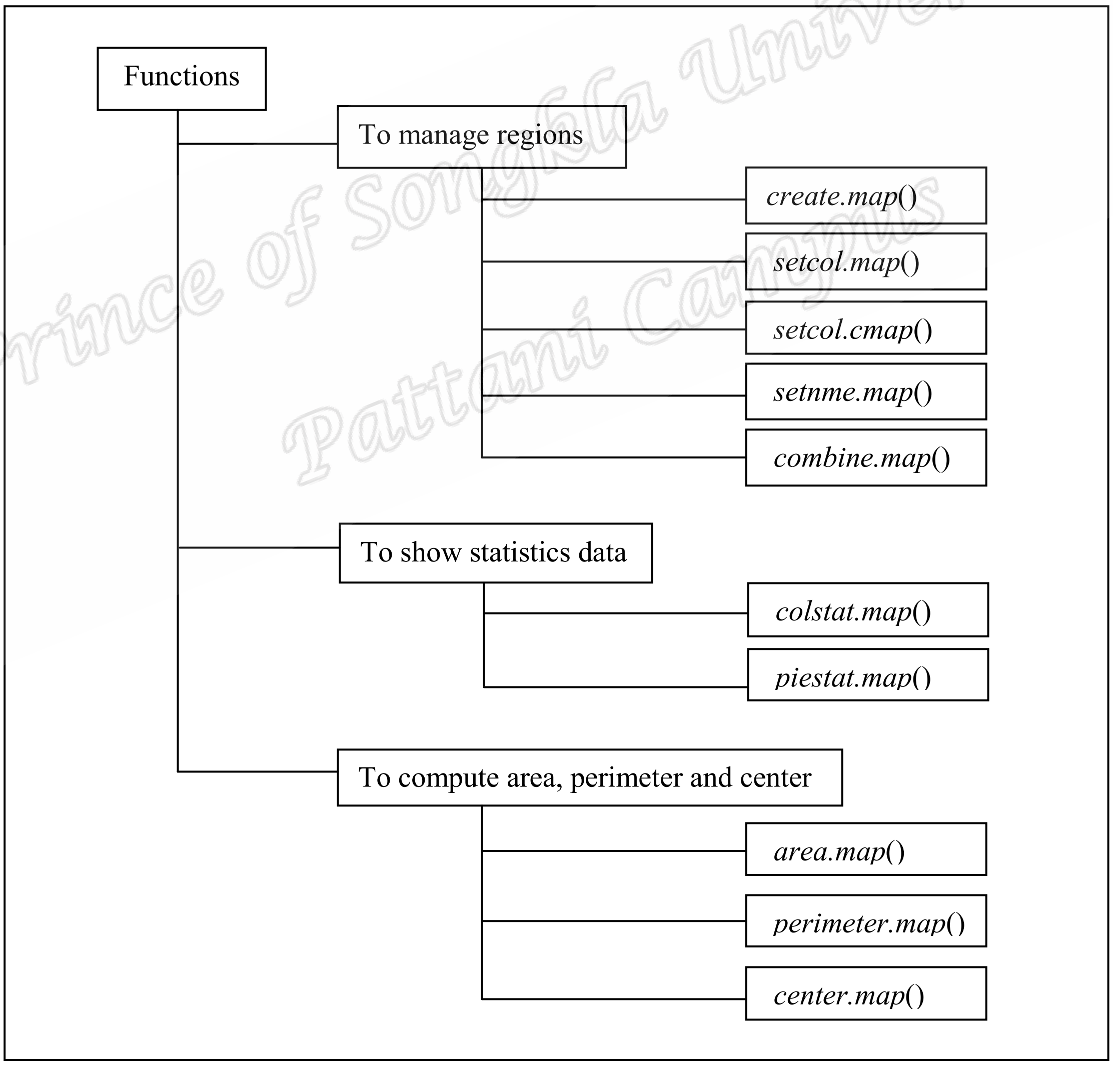


Figure 2.12: The functions are created

__The *create.map*() function__

Objective: To create a map.

Function syntax: *create.map* (*flexy, xscl, yscl, scl.size, scl.col, wh, ww, header.text, header.size, header.col, map.col, xylabel, xylabel.size, xylabel.col, xyline, xyline.col, xyline.type*)

The arguments are described in table 2.1.

Argument	Description
flexy	Spatial data file
xscl	The minimum and maximum value of *x* axis
yscl	The minimum and maximum value of *y* axis
scl.size	Size of *x* axis and *y* axis
scl.col	Color of *x* axis and *y* axis
wh	Window height
ww	Window width
header.text	The main title name
header.size	Size of title name
header.col	Color of title name
map.col	Color of map
xylabel	Show "*UTM-North*" on *y* label and "*UTM-East*" on *x* label
xylabel.size	Size of *x* label and *y* label
xylabel.col	Color of *x* label and *y* label
xyline	Show grid line
xyline.col	Color of grid line
xyline.type	Type of grid line

Table 2.1: The data input for *create.map*() function

The algorithm for creating map is described in algorithm 2.1. There are seven steps.

Algorithm 2.1: *create.map*()

1. Read the data from syntax.

2. Check complex regions, if one region has two pairs or more of duplicated *x*- and *y*-coordinates, put the *NA* value between each region of each complex region.

3. Create a window using *windows*() function.

4. Create a map using *polygon*() function.

5. If *header.text* is not null, display the title name on the top of a map using *mtext*() function.

6. If *xyline* = *T*, display the grid line with *abline*() function.

7. Display the place code or the primary key of simple regions and complex regions on R Console.

Example: Figure 2.13 shows the result from *create.map*() function. It is 13 sub-districts of Mueang Pattani district, Pattani province. There are Sabarang, Anoru, Chabang Tiko, Bana, Tanyong Lulo, Khlong Maning, Kamiyo, Barahom, Paka Harang, Rusa Milae, Talubo, Baraho and Puyut.

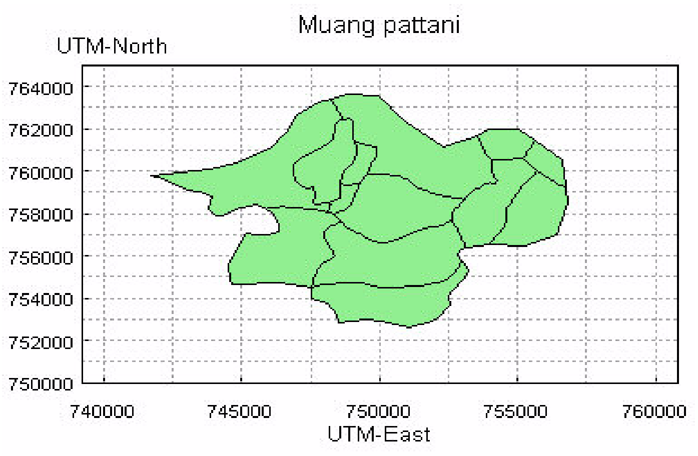


Figure 2.13: The result from *create.map*() function

The *setcol.map*() function

Objective: To specify color of each region.

Function syntax: *setcol.map* (*flexy, plcid, mcol*)

The arguments are described in table 2.2.

Argument	Description
flexy	Spatial data file
plcid	The primary key of region to display color
mcol	Color of each region

Table 2.2: The data input for *setcol.map*() function

The algorithm for specifying color of each region is described in algorithm 2.2. There are six steps.

Algorithm 2.2: *setcol.map*()

1. Read the data from syntax.

2. Check complex regions, if one region has two pairs or more of duplicated *x*- and *y*-coordinates, put the*NA* value between each region of each complex region.

3. If *plcid*s null, get the primary key in *flexy*.

4. If *mcol* is null, generate the color for *mcol*.

5. Display the color of each region using *polygon*() function. For complex regions, all regions of each complex region willhave the same color.

6. Display the place code or the primary key and the color of each region on R Console.

Example: Figure 2.14 shows the result from *setcol.map*() function of Ko Mak sub-district, Pak Phayun district, Phatthalung Province.

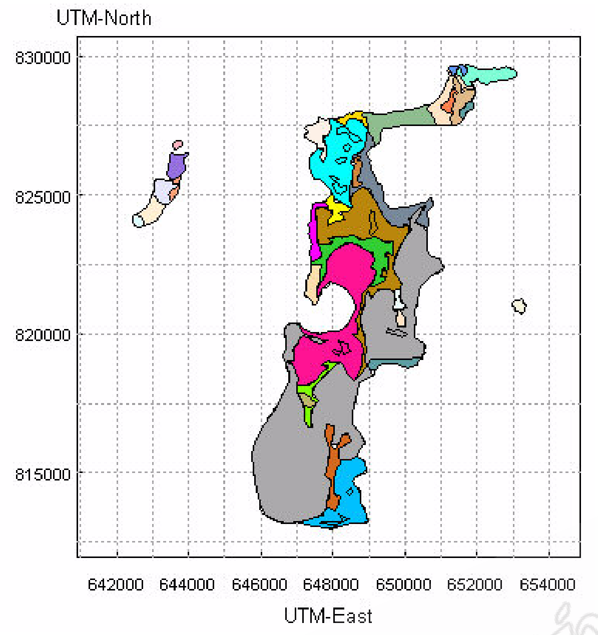


Figure 2.14: The result from *setcol.map*() function

The *setcol.cmap*() function

Objective: To specify color of complex region.

Function syntax: *setcol.cmap* (*flexy, plcid, reg, mcol*)

The arguments are described in table 2.3.

Argument	Description
flexy	Spatial data file
plcid	The primary key of a complex region to display color
reg	Specify regions are numeric (1, 2, 3, …, *n*) from the largest region to the smallest region
mcol	Color of each region

Table 2.3: The data input of *setcol.cmap*() function

The algorithm for specifying color of complex region is described in algorithm 2.3. There are five steps.

Algorithm 2.3: *setcol.cmap*()

1. Read the data from syntax.

2. Count the number of region in *plcid* variable.

3. Compute the area of each region and sort them from the largest region to the smallest region.

4. Display the color of each region with *polygon*() function.

5. Display the place code or the primary key and the color of each region on R Console.

Example: Figure 2.15 shows the result from *setcol.cmap*() function. It is a complex region of one place in figure 2.14.

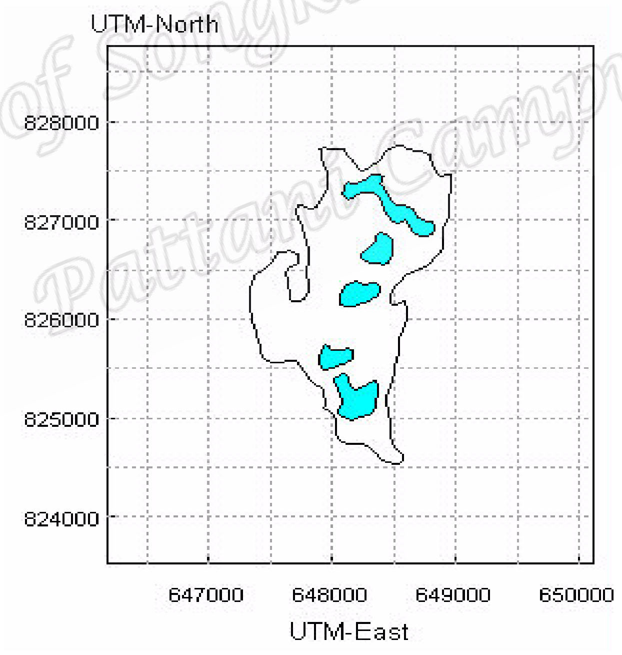


Figure 2.15: The result from *setcol.cmap*() function

The *setnme.map*() function

Objective: To display a name on each region.

Function syntax: *setnme.map* (*x, y, flexy, plcid, nme, frm, sfrm, wfrm, colfrm, sfont, wfont, nmecmap*)

The arguments are described in table 2.4.

Argument	Description
x	Position of *x* axis
y	Position of *y* axis
flexy	Spatial data file
plcid	The primary key of region to display name
nme	The name to display
frm	Frame of name
sfrm	Size of frame
wfrm	Width of frame
colfrm	Color of frame
sfont	Font size of name
wfont	Font width of name
nmecmap	If do not display name of complex regions giving value to "*F*"

Table 2.4: The data input of *setnme.map*() function

The algorithm for displaying a name on each region is described in algorithm 2.4. There are six steps.

Algorithm 2.4: *setnme.map*()

```
1. Read the data from syntax.
2. Check complex regions, if one region has two pairs or more of duplicated x- and y-
coordinates, put theNA value between each region of each complex region.
3. If x, y and nme is not null.
        3.1 If frm is not null, display the frame using points() function.
        3.2 Display the name on each region with text() function.
4. If plcidand nme is not null.
.       4.1 If frm is not null, display the frame using points() function.
        4.2 For simple regions, compute the center of each region and display the
name using text() function on the center of each region.
        4.3 For each complex region:
                - Count the region number of a complex region.
                - Compute the area of each region.
                - Find the maximum area from all regions and compute the center.
                - Display the name using text() function.
5. If x, y and plcid is null.
        5.1 Find the place code or the primary key in flexy.
        5.2 If frm is not null, display the frame using points() function.
        5.3 For simple regions, compute the center of each region and display the
name using text() function on the center of each region.
        5.4 For each complex region:
                - Count the region number of a complex region.
                - Compute the area of each region.
```

- Find the maximum area from all regions and compute the center.

- Display the name using *text*() function.

5.5 Display the name of each region, that is numeric from 1 to *n* (*n* is a number of region) with *text*() function on the center of region.

6. Display the place code or the primary key and the name of each region on R Console.

Example: Figure 2.16 shows the result from *setnme.map*() function.

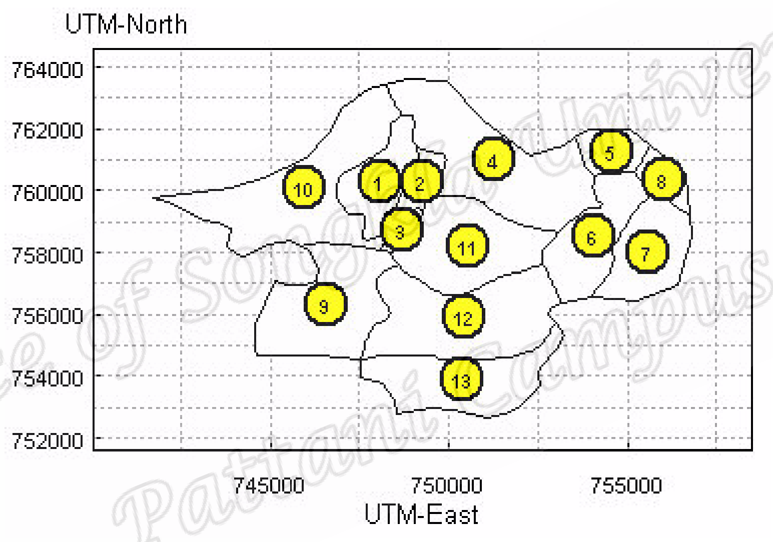


Figure 2.16: The result from *setnme.map*() function

The *combine.map*() function

Objective: To combine different regions into one region.

Function syntax: *combine.map* (*flexy, plcid, mcol, mline*)

The arguments are described in table 2.5.

Argument	Description
flexy	Spatial data file
plcid	The primary key of region to combining
mcol	Color of region after combining
mline	Line type of region after combining

Table 2.5: The data input of *combine.map*() function

The algorithm for combining regions is described in algorithm 2.5. There are five steps.

Algorithm 2.5: *combine.map*()

1. Read the data from syntax.

2. Check complex regions, if one region has two pairs or more of duplicated *x*- and *y*- coordinates, put the*NA* value between each region of each complex region.

3. If *mcol* is not null, display the color of each region using *polygon*() function.

4. Merge place which is specified in *plcid*variable with *merge*() function.

5. Draw new line, especially the *x*- and *y*- coordinates are matched by using *line*() function.

Example: Figure 2.17 shows the result from *combine.map*() function.

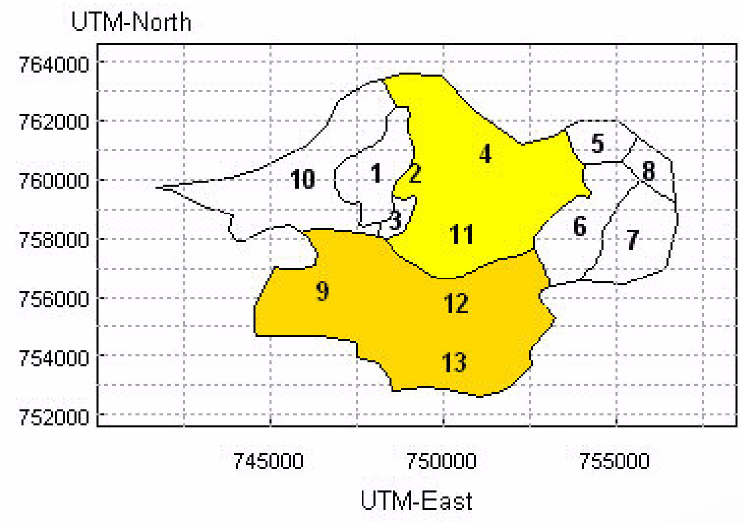


Figure 2.17: The result from *combine.map*() function

The *colstat.map*() function

Objective: To display statistical data on the map using color shade.

Function syntax: *colstat.map* (*flexy, plcid, dat, mcol, grp, mline, xlg, ylg, lg, pslg, slg, ncollg*)

The arguments are described in table 2.6.

Argument	Description
flexy	Spatial data file
plcid	The primary key of region
dat	The statistical data or the information
mcol	Color of each group
grp	Group number
mline	Type of line.
xlg	Position of *x* axis to show legend
ylg	Position of y axis to show legend
lg	Show legend defaults to "*TRUE*"

Argument	Description
pslg	Position of legend
slg	Size of legend
ncollg	Column number of legend

Table 2.6: The data input of *colstat.map*() function

The algorithm for displaying color shade is described in algorithm 2.6. There are six steps.

Algorithm 2.6: *colstat.map*()

1. Read the data from syntax.

2. Check complex regions, if one region has two pairs or more of duplicated *x*- and *y*-coordinates, put the*NA* value between each region of each complex region.

3. If *dat* is a categorical variable, display the color of each group using *polygon*() function and display the type of line using method like *combine.map*() function.

4. If *dat* is a continuous variable.

    4.1 Manage *dat* variable to be categorical variable by using Frequency Distribution of Grouped Data including:

        - Compute range of data using formula: $r = Max\text{-}Min$

        - If *grp* is not null, give $k = grp$. If *grp* is null, compute number of group using formula: $k = 1+3.3logN$

        - Compute class interval using formula: $i = r/k$

        - Give the value for each group, which are *k* groups.

    4.2 Define value, which is the categorical data, to *dat* variable.

4.3 Display the color of each group with *polygon*() function and display the type of line using method like *combine.map*() function.

5. If *lg* = *T*, display the legend of data with *legend*() function using position of *xlg*, *ylg* or *pslg*.

6. Display the name and the color of each group on R Console.

Example: Figure 2.18 shows the result from *colstat.map*() function.

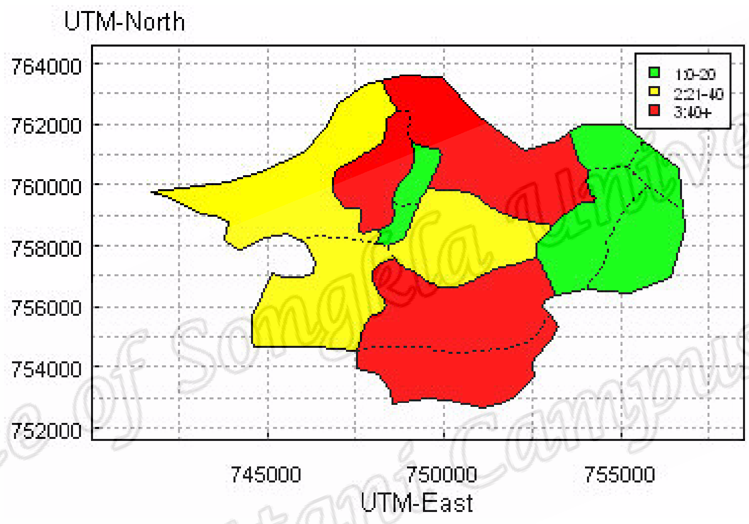


Figure 2.18: The result from *colstat.map*() function

The *piestat.map*() function

Objective: To display statistical data on the map using circle.

Function syntax: *piestat.map* (*flexy, plcid, dat, mcol, grp, xlg, ylg, lg, pslg, slg, ncollg, strpie*)

The arguments are described in table 2.7.

Argument	Description
flexy	Spatial data file
plcid	The primary key of region
dat	The statistical data or the information
mcol	Color of circle
grp	Group number, when *dat* variable is integer
xlg	Position of *x* axis to show legend
ylg	Position of y axis to show legend
lg	Show legend defaults to "*TRUE*"
pslg	Position of legend
slg	Size of legend
ncollg	Column number of legend
strpie	The start size of circle

Table 2.7: The data input of *piestat.map*() function

The algorithm for displaying circle is described in algorithm 2.7. There are six steps.

Algorithm 2.7: *piestat.map*()

```
1. Read the data from syntax.
2. Check complex regions, if one region has two pairs or more of duplicated x- and y-coordinates, put theNA  value between each region of each complex region.
3. If dat is categorical variable.
        3.1 For simple regions, display the circle on the center of each region using points() function. Size of the circle should be increased for each group.
        3.2 For each complex region:
                - Count the number of region of a complex region.
```

- Compute the area of each region.

- Find the maximum area from all regions and compute the center of that region.

- Display the circle using *points*() function.

4. If *dat* is continuous variable.

4.1 Manage *dat* variable to be categorical variable by using Frequency Distribution of Grouped Data including:

- Compute range of data using formula: $r = Max\text{-}Min$

- If *grp* is not null, give $k = grp$. If *grp* is null, compute the number of group using formula: $k = 1{+}3.3logN$

- Compute class interval using formula: $i = r/k$

- Give the value for each group, which are *k* groups.

4.2 Define value, which is the categorical data, to *dat* variable.

4.3 For simple regions, display the circle on the center of each region using *points*() function. Size of the circle should be increased for each group.

4.4 For each complex region:

- Count the number of region of a complex region.

- Compute the area of each region.

- Find the maximum area from all regions and compute the center of that region.

- Display the circle using *points*() function.

5. If $lg = T$, display the legend of data with *legend*() function using position of *xlg*, *ylg* or *pslg*.

6. Display the name and the circle size of each group on R Console.

Example: Figure 2.19 shows the result from *piestat.map*() function.

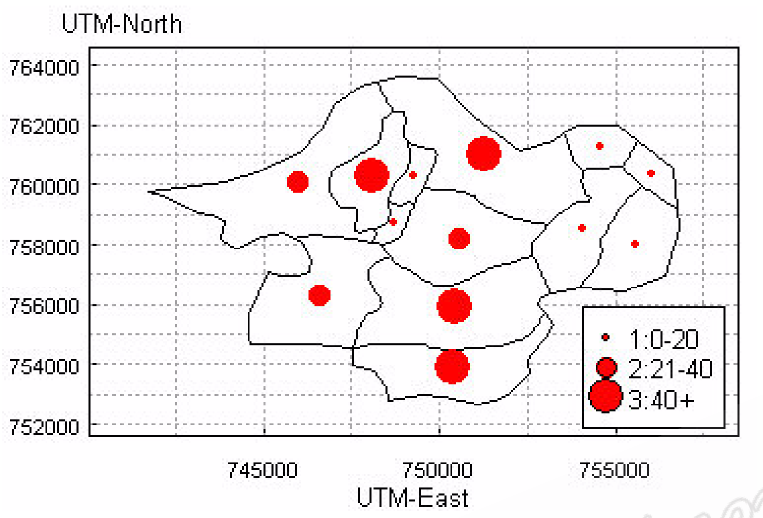


Figure 2.19: The result from *piestat.map*() function

The *area.map*() function

Objective: To compute the area of each region.

Function syntax: *area.map* (*flexy, plcid, mshow*)

The arguments are described in table 2.8.

Argument	Description
flexy	Spatial data file
plcid	The primary key of region
mshow	Show the area on map, defaults to "*FALSE*"

Table 2.8: The data input of *area.map*() function

The algorithm for computing area is described in algorithm 2.8. There are six steps.

Algorithm 2.8: *area.map*()

1. Read the data from syntax.

2. If *plcid*s null, get the primary key in *flexy*.

3. For complex regions, count the number of region of each complex region.

4. Compute the area of simple regions and each region of complex regions.

5. If *mshow* = *T*, display the area on the center of each region.

6. Display the area of each region on R Console.

Example: Figure 2.20 shows the result from *area.map*() function.

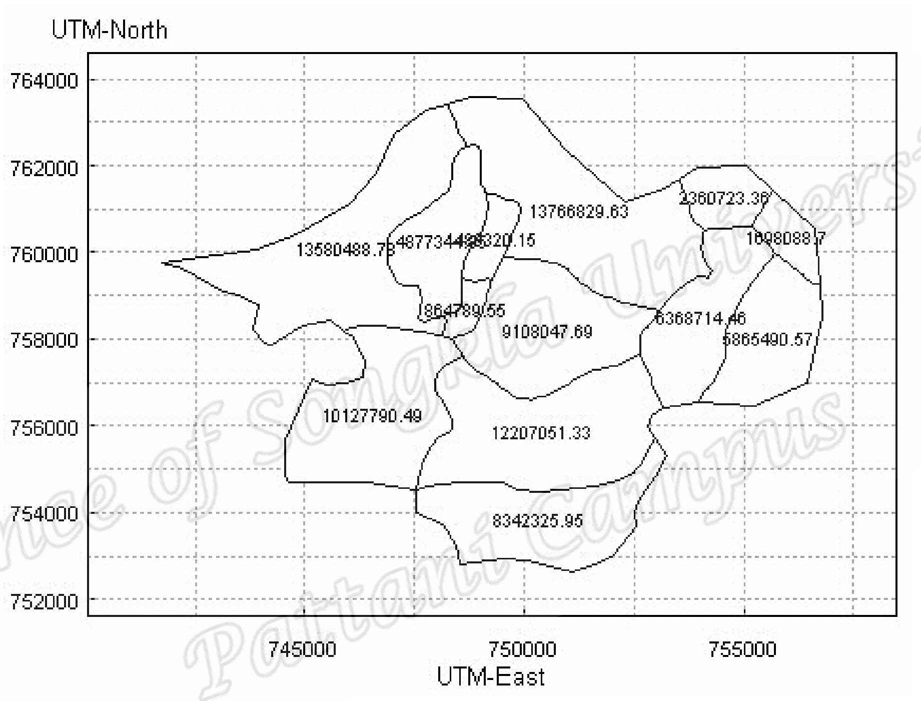


Figure 2.20: The result from *area.map*() function

<u>The *perimeter.map*() function</u>

Objective: To compute the perimeter of each region.

Function syntax: *perimeter.map* (*flexy, plcid, mshow*)

The arguments are described in table 2.9.

Argument	Description
flexy	Spatial data file
plcid	The primary key of region
mshow	Show the area on map, defaults to "*FALSE*"

Table 2.9: The data input of *perimeter.map*() function

The algorithm for computing perimeter is described in algorithm 2.9. There are six steps.

Algorithm 2.9: *perimeter.map*()

1. Read the data from syntax.
2. If *plcid*s null, get the primary key in *flexy*.
3. For complex regions, count the number of region of each complex region.
4. Compute the perimeter of simple regions and each region of complex regions.
5. If *mshow* = *T*, display the perimeter on the center of each region.
6. Display the perimeter of each region on R Console.

Example: Figure 2.21 shows the result from *perimeter.map*() function. It is the Na Thap canal in Na Thap sub-district, Chana district, Songkhla province.

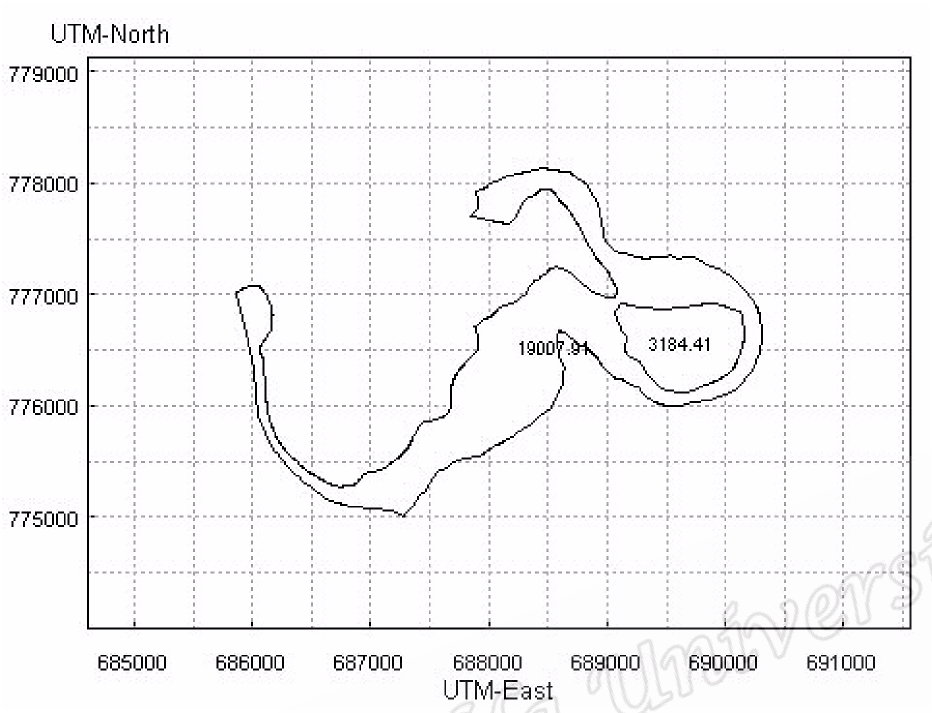


Figure 2.21: The result from *perimeter.map*() function

The *center.map*() function

Objective: To compute the center of each region.

Function syntax: *center.map* (*flexy, plcid*)

The arguments are described in table 2.10.

Argument	Description
flexy	Spatial data file
plcid	The primary key of region

Table 2.10: The data input of *center.map*() function

The algorithm for computing the center is described in algorithm 2.10. There are five steps.

Algorithm 2.10: *center.map*()

1. Read the data from syntax.

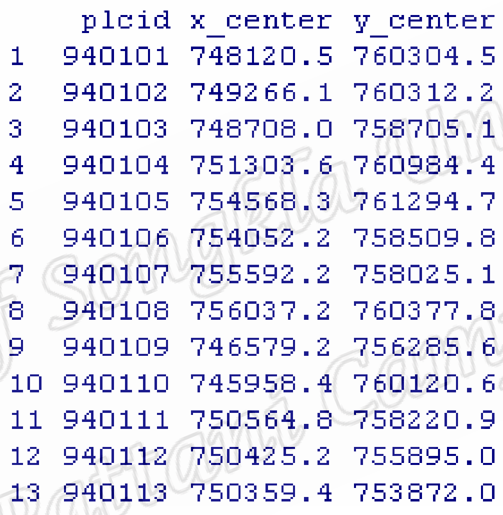
2. If *plcid*s null, get the primary key in *flexy*.

3. For complex regions, count the number of region of each complex region.

4. Compute the center of simple regions and each region of complex regions.

5. Display the center of each region on R Console.

Example: Figure 2.22 shows the result from *center.map*() function.



```
    plcid x_center y_center
1  940101 748120.5 760304.5
2  940102 749266.1 760312.2
3  940103 748708.0 758705.1
4  940104 751303.6 760984.4
5  940105 754568.3 761294.7
6  940106 754052.2 758509.8
7  940107 755592.2 758025.1
8  940108 756037.2 760377.8
9  940109 746579.2 756285.6
10 940110 745958.4 760120.6
11 940111 750564.8 758220.9
12 940112 750425.2 755895.0
13 940113 750359.4 753872.0
```

Figure 2.22: The result from *center.map*() function

In summary, the functions must have at least two files to create the map and display the information, there are a spatial data file and an attribute data file. The spatial data file has two types that are simple region and complex region. In this chapter also we described the methodology to create function of 10 functions, which include the objective, function syntax, algorithm and the example of result of each function.

For next chapter, we will describe the detail of result for each function. How is the ability of each function?