Triangulation of irregular spaced data using the sweep hull algorithm

Albrecht Gebhardt
University Klagenfurt

Roger Bivand
Norwegian School of Economics

Abstract

This vignette presents the R package **interp** and focuses on triangulation of irregular spaced data.

This is the second of planned three vignettes for this package (not yet finished).

Keywords: triangulation, Voronoi mosaic, R software.

1. Note

Notice: This is a preliminary and not yet complete version of this vignette. Finally three vignettes will be available for this package:

1. a first one related to partial derivatives estimation,
2. a next one describing interpolation related stuff
3. and this one dealing with triangulations and Voronoi mosaics.

2. Introduction

The functions described here where formerly (and still are) available in the R package **tripack** which is based on algorithms described in (Renka 1996). This code was also used by Akima in (Akima 1996) for his improved spline interpolator. Both these algorithms are under ACM licene and so the need to reimplement all related functions under a free license arose.

This package now re-implements the functions from the package **tripack** with a different but free triangulation algorithm operating in the background. This algorithm is a sweep hull algorithm introduced in (Sinclair 2016).

3. Delaunay Triangulation

In the next section we will use the notion of Delaunay triangulations, so lets start with this definition.

Definition 3.1. *Given a set of points* $P = \{p_i | p_i = (x_i, y_i)^\intercal, x_i \in \mathbb{R}, y_i \in \mathbb{R}, i = 1, \ldots, n\}$ *the set of all triangles with vertices in* P *which fulfill the condition that none of the points*

from P is contained in the interior of the circumcircle of any such triangle is called Delaunay triangulation.

Algorithms to determine Delaunay triangulations can be split into two steps:

1. An initial step to generate a triangulation which itself is a disjoint partition of the convex hull of P built with non-overlapping triangles out of the given vertices.
2. In a second step pairs of neighbouring triangles (p_1, p_2, p_3) and (p_3, p_2, p_4) which share a common edge (p_2, p_3) and do not fulfill the circumcircle condition in definition 3.1 are selected. Now these triangles are swapped, the new triangles beeing (p_1, p_2, p_4) and (p_4, p_2, p_3). They will now fulfil the condition.

Step 2 is repeated until no such pair of triangles to swap can be found anymore.
Sinclairs sweep hull algorithm (Sinclair 2016) specifies step 1 as follows:

1. Take a random triangle which contains none of the remaining points. This forms a initial triangulation with a known convex hull (the triangle itself).
2. Sort the remaining points in ascending distance to this triangle (its center).
3. Repeat until all points are exhausted:
   (a) Take the next nearest point p_{next}.
   (b) Determine that part of the convex hull of the current triangulation which is "visible" from p_{next}.
   (c) Form all non overlapping triangles with p_{next} and the "visible" part of the current convex hull.
   (d) Add the new triangles to the current triangulation, correct the convex hull to the new state.

The function `tri.mesh` is now applied to a simple artificial example data set:

```
> data(tritest)
> tr <- tri.mesh(tritest)
> tr

Delaunay triangulation, node and triangle indices:
triangle: nodes (a,b,c), neighbour triangles [i,j,k]
1: (1,9,4), [3,9,2]
2: (3,9,1), [1,15,5]
3: (12,4,9), [1,4,10]
4: (9,10,12), [6,3,5]
5: (9,3,10), [14,4,2]
6: (11,12,10), [4,12,8]
7: (12,6,7), [16,10,8]
8: (6,12,11), [6,17,7]
9: (7,1,4), [1,10,0]
10: (12,7,4), [9,3,7]
11: (11,5,8), [18,17,12]
12: (5,11,10), [6,13,11]
13: (10,2,5), [18,12,14]
```

```
14: (10,3,2), [15,13,5]
15: (2,3,1), [2,0,14]
16: (8,7,6), [7,17,0]
17: (11,8,6), [16,8,11]
18: (8,5,2), [13,0,11]
boundary nodes:  7 1 2 8
```

In return the triangles and the indices of their neighbour triangles will be printed. With `interp::triangles()` more detailed information can be accessed:

```
> triangles(tr)

      node1 node2 node3 tr1 tr2 tr3 arc1 arc2 arc3
 [1,]     1     9     4   3   9   2    1    2    3
 [2,]     3     9     1   1  15   5    3    4    5
 [3,]    12     4     9   1   4  10    1    6    7
 [4,]     9    10    12   6   3   5    8    6    9
 [5,]     9     3    10  14   4   2   10    9    5
 [6,]    11    12    10   4  12   8    8   11   12
 [7,]    12     6     7  16  10   8   13   14   15
 [8,]     6    12    11   6  17   7   12   16   15
 [9,]     7     1     4   1  10   0    2   17   18
[10,]    12     7     4   9   3   7   17    7   14
[11,]    11     5     8  18  17  12   19   20   21
[12,]     5    11    10   6  13  11   11   22   21
[13,]    10     2     5  18  12  14   23   22   24
[14,]    10     3     2  15  13   5   25   24   10
[15,]     2     3     1   2   0  14    4   26   25
[16,]     8     7     6   7  17   0   13   27   28
[17,]    11     8     6  16   8  11   27   16   20
[18,]     8     5     2  13   0  11   23   29   19
```

The first three columns contain the indices of the triangle vertices, the next three columns carry the indices of the neighbour triangles (0 means it is neigbour to the plane outside the convex hull). The last three columns are filled with indices to the arcs of the triangulation.

While plotting the triangulation, we also plot the circumcircles to check the condition of empty circumcircles:

```
> MASS::eqscplot(tritest)
> plot(tr, do.circumcircles=TRUE, add=TRUE)
```

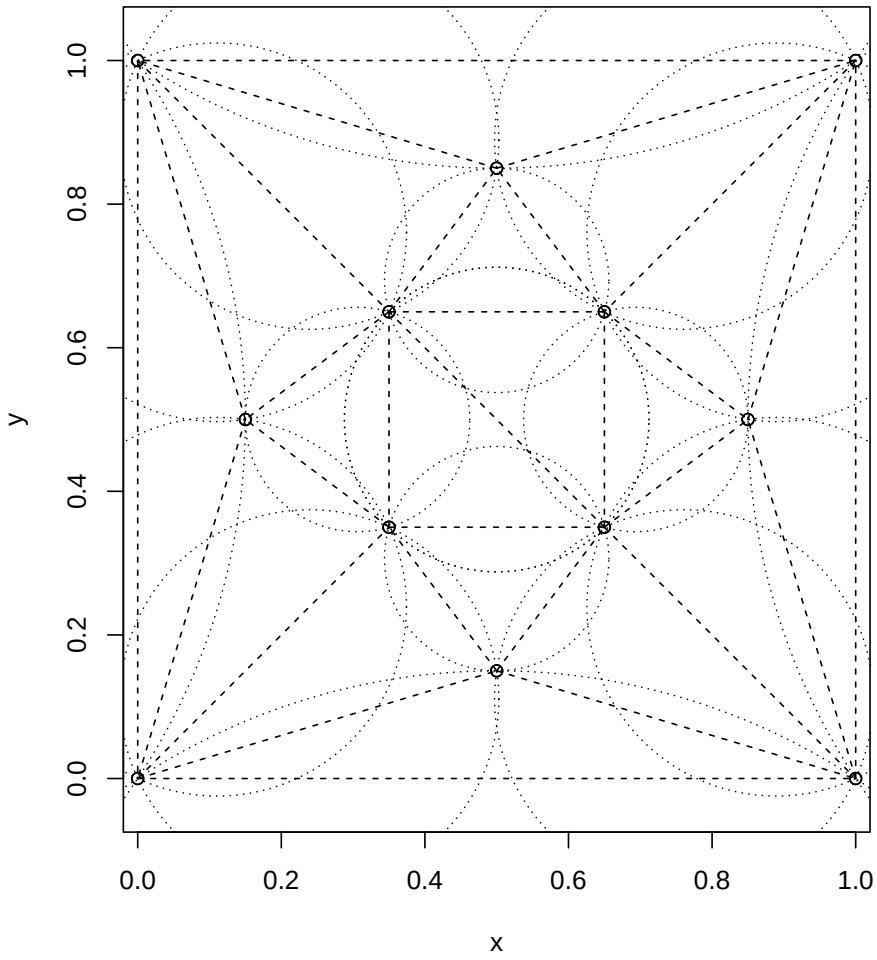

Figure 1: Delaunay triangulation with added circumcircles

4. Voronoi Mosaics

Definition 4.1. *Given a set of points* $P = \{p_i | p_i = (x_i, y_i)^\intercal, i = 1, \ldots, n\}$ *the associated Voronoi mosaic is a disjoint partition of the plane, where each set of this partition (the Thiessen polygon) is created by one of the points* p_i *in a way that this set is the geometric location of all points of* $\mathbb{R}^2$ *which have* p_i *as its nearest neighbour out of the set* P*.*

There is some sort of duality between Delaunay triangulations and Voronoi mosaics:

The circumcircle centers of the triangles of the triangulation are the vertices of the Voronoi mosaic. The edges of the Voronoi mosaic are the perpendicular bisectors of the edges of the triangles of the triangulation.

Using this duality it is easy to construct a Voronoi mosaic given a Delaunay triangulation. This is done completely in R, no `Rcpp` is used.

Continuing with the previous data we get the following mosaic:

```
> vm <- voronoi.mosaic(tr)
> vm

voronoi mosaic:
nodes: (x,y): neighbours (<0: dummy node)
1: (0.1107143,0.2392857): 3 9 2
2: (0.2392857,0.1107143): 1 15 5
3: (0.30625,0.5): 1 4 10
4: (0.5,0.5): 6 3 5
5: (0.5,0.30625): 14 4 2
6: (0.5,0.5): 4 12 8
7: (0.2392857,0.8892857): 16 10 8
8: (0.5,0.69375): 6 17 7
9: (-0.7583333,0.5): 1 10 -1
10: (0.1107143,0.7607143): 9 3 7
11: (0.8892857,0.7607143): 18 17 12
12: (0.69375,0.5): 6 13 11
13: (0.8892857,0.2392857): 18 12 14
14: (0.7607143,0.1107143): 15 13 5
15: (0.5,-0.7583333): 2 -2 14
16: (0.5,1.758333): 7 17 -3
17: (0.7607143,0.8892857): 16 8 11
18: (1.758333,0.5): 13 -4 11
dummy nodes: (x,y)
1: (-3.275,0.5)
2: (0.5,-3.275)
3: (0.5,4.275)
4: (4.275,0.5)
```

Dummy nodes have to be created to build the unbounded Voronoi cells on the border of the mosaic.

Again while plotting it we overlay it with the triangulation to show the above mentioned duality:

```
> MASS::eqscplot(tritest)
> plot(vm, add=TRUE)
> plot(tr, add=TRUE)
```

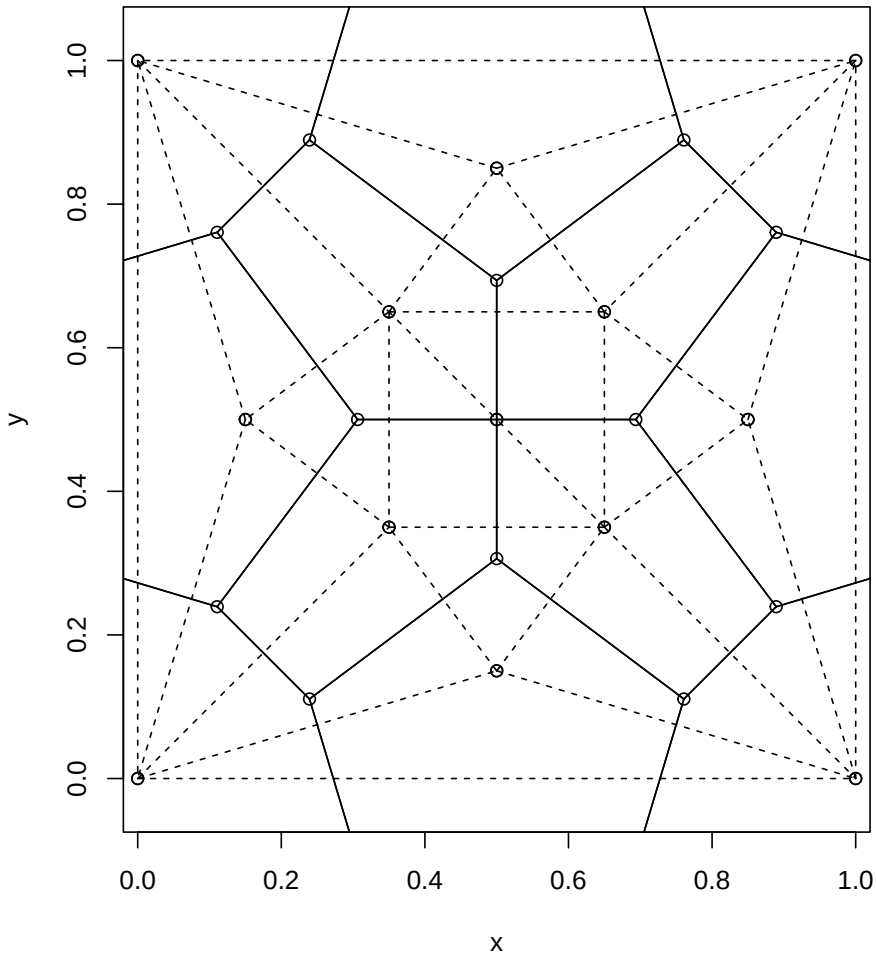

Figure 2: Voronoi mosaic with Delaunay triangulation as overlay

5. Implementation details

This is the call to `tri.mesh`:

```
tri.mesh(x, y = NULL, duplicate = "error", jitter = FALSE)
```

The argument `duplicate` offers three options to deal with duplicates:

- `"error"`: Stop with an error, this is the default.
- `"strip"`: Completely remove points with duplicates, or
- `"remove"`: Leave one of the duplicates and remove the remaining.

The two vectors `x` and `y` of equal length contain the coordinates of the given data points. Omitting `y` implicates that `x` consist of a two column matrix or dataframe containing x and y entries.

In case of errors with a specific data set the option `jitter=TRUE` can be tried. It adds some small random error to the x, y location. In some cases (e.g. collinear points) this can help to succeed with the triangulation. Under some circumstances the algorithm internally decides to restart with jitter. In this case a warning is issued.

The return value of `interp::tri.mesh()` is of the class `triSht`. This is in contrast to the return value of `tripack::tri.mesh()` which returns an object of class `tri`.

That means that it is not possible to use objects created by `tripack::tri.mesh()` as arguments to functions in **interp** which operate on triangulations returned by `interp::tri.mesh()`.

The call to `voronoi.mosaic()` uses the same arguments:

```
voronoi.mosaic(x, y = NULL, duplicate = "error")
```

`x` and `y` are treated as in `tri.mesh()`, but `x` can also be a triangulation object of class `triSht` returned by `tri.mesh()`.

All functions from **tripack** which generate triangulation or Voronoi mosaic objects are also available in **interp** with matching calls. The only restriction is that restricted triangulations as possible in **tripack** are not implemented in **interp**.

References

Akima H (1996). "Algorithm 761: scattered-data surface fitting that has the accuracy of a cubic polynomial." *ACM Transactions on Mathematical Software*, **22**, 362–371.

Renka RJ (1996). "Algorithm 751: TRIPACK: A Constrained Two-Dimensional Delaunay Triangulation Package." *ACM Transactions on Mathematical Software*, **22**, 1–8.

Sinclair D (2016). "S-hull: a fast radial sweep-hull routine for Delaunay triangulation." URL https://archive.org/details/arxiv-1604.01428.

List of Figures

Affiliation:

Albrecht Gebhardt Institut für Statistik
Universität Klagenfurt 9020 Klagenfurt, Austria
E-mail: albrecht.gebhardt@aau.at