# Implementation of Three-dimensional Scagnostics

by

Lijie Fu

A research paper presented to the
University of Waterloo
In partial fulfillment of the requirements for the degree of
Master of Mathematics
in
Computational Mathematics

supervisor: Wayne Oldford

Waterloo, Ontario, Canada, 2009

## Abstract

The concept of scagnostics was first proposed by Tukey and Tukey (at an IMA workshop in 1989). It is an combination of two terms: scatterplot and cognostics (**co**mputer **g**uiding diag**nostic**), which means diagnosing interesting features of scatterplots guided by computer. It is an approach to assess the patterns of scattered points and distinguish the most interesting patterns existed.

Graph-theoretic scagnostics were developed by Wilkinson et al. (2005) [1]. There, they thoroughly described how to relate convex hull, alpha hull and Euclidean minimum spanning tree with the nine measures of the data points. The nine measures are derived from geometric features and out of the interests on three aspects of the scattered points, which are density, shape and association. `scagnostics` in 2D was implemented.

In this project, we are interested in measuring those characteristics for points scattered in three dimensional spaces. Following the similar methodology and using the 3D geometric concepts, we extended the scagnostics application to three-dimensional spaces. The graph features are re-evaluated in 3D context and definitions of corresponding measures are reviewed. An implementation is given in R and Java package.

# Contents

# Chapter 1

# Introduction

## 1.1 Viewing High Dimensional Datasets

### 1.1.1 The Problem

The human visual system can perceive at most three-dimensional images. A $p$-dimensional dataset($p > 3$) has to be broken down into $\binom{p}{3}$ or $\binom{p}{2}$ subsets to be displayed. Some salient patterns can be found out by examining the plots one by one. There are many methods available nowadays. For example, parallel coordinate plots, scatterplot matrix, 3D scatterplot are well known (Figure 1.1). Simply plotting the variables pairwise is not sufficient for illustrating all underlying patterns. Dynamically morphing between scatterplots of subsets of variables is the approach taken by the system `GGobi`.

Here is a brief introduction of these methods.

- Parallel coordinate plots
  A parallel coordinate system uses parallel lines representing the axes. Each line corresponds to a variable of the dataset such that a point in the data set is transformed into a curve in the parallel coordinate plot.

- Scatterplot matrix
  A scatterplot reveals relationships between two variables. When there is more than three variables in a data set, a scatterplot matrix (also called SPLOM) can be created to show the relationships between every pair of variables. Each element of the matrix

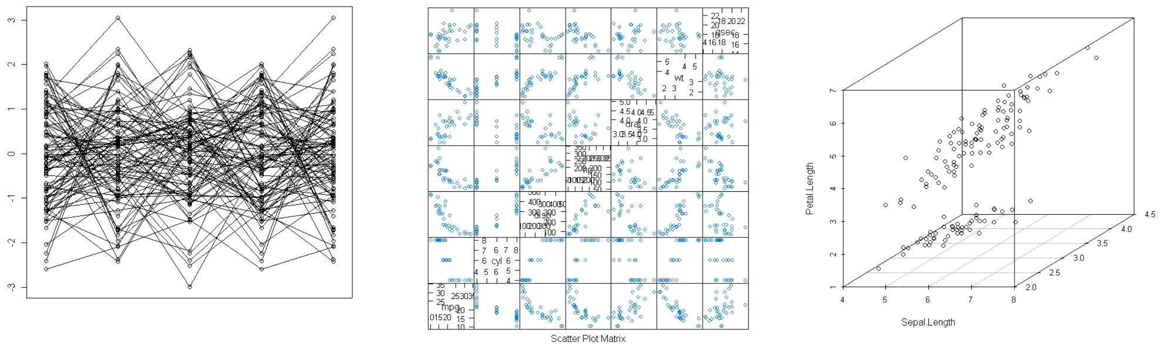


Figure 1.1: Parallel coordinate plot, scatterplot matrix and 3D scatterplot

is a pairwise scatterplot. The matrix has $p$ rows and $p$ columns for a data set with $p (p > 2)$ variables.

- 3D scatterplot
  3D scatterlpots visualize the three dimensional data. A static 3D scatterplot does not reveal the pattern comprehensively, but it has to be viewed from different angles. Interactive or rotatable 3D scatterplots are designed for this purpose. An example is the automatic rotating 3D scatterplots in `GGobi`.

- Dynamic 2D tours
  The 2D tour is a kind of grand tour. "A grand tour attempts to provide the viewer with an overview of a multivariate point scatter by presenting a continuous (dynamic) sequence of low ($d$, usually=1,2,3) dimensional projections. " [5]. When $d$ is taken to be 2, the grand tour is a 2D tour. It generates a continuous sequence of 2D projections of the active variable space. The projected data can be displayed as a scatterplot.

Problems arise when the number of variables is large. First of all, the static plots, such as parallel coordinate plots and scatterplot matrices, become very cumbersome and hard to read. For example, we get $\binom{100}{2}= 4950$ scatterplots in the matrix for a data set with 100 variables. How to choose a pair which is worthy of investigation at a glance? Secondly, it is visually difficult to identify interesting patterns in a dynamic system. This problem inspired the invention of cognostics, and its special case, scagnostics.

### 1.1.2 Possible Solutions

The term ‘cognostics’ is the abbreviation for **co**mputer **g**uiding diag**nostics**. Just as the name implies, “ a ‘cognostic’ is a diagnostic to be interpreted by a computer rather than by a human. ” [4]. This is especially useful for a large number of variables. Computers evaluate and rank the predefined cognostics so as to filter the most interesting patterns (relatively few) to be viewed by users [10].

Cognostics designed for scatterplots are called ‘scagnostics’. Tukey and Tukey proposed this approach to address the problem of the effectiveness of scatterplot matrices (Tukey and Tukey, 1985). The idea is to compute measures for each of several different patterns of points in a scatter plot. Then make a scatterplot matrix of the measures themselves, which act as a guide to generate scatterplot matrices for interesting scatters. Since the measures are of fixed number $k$ ($k = 9$ were suggested), an $O(p^2)$visual task is reduced to an $O(k^2)$ visual task [26]. A 2D `scagnostics` package was published as a contributed R package in 2007 - a corrected version in 2009. Figure 1.2 illustrates an index of nine measures and the scagnostics SPLOM for the Abalone dataset.

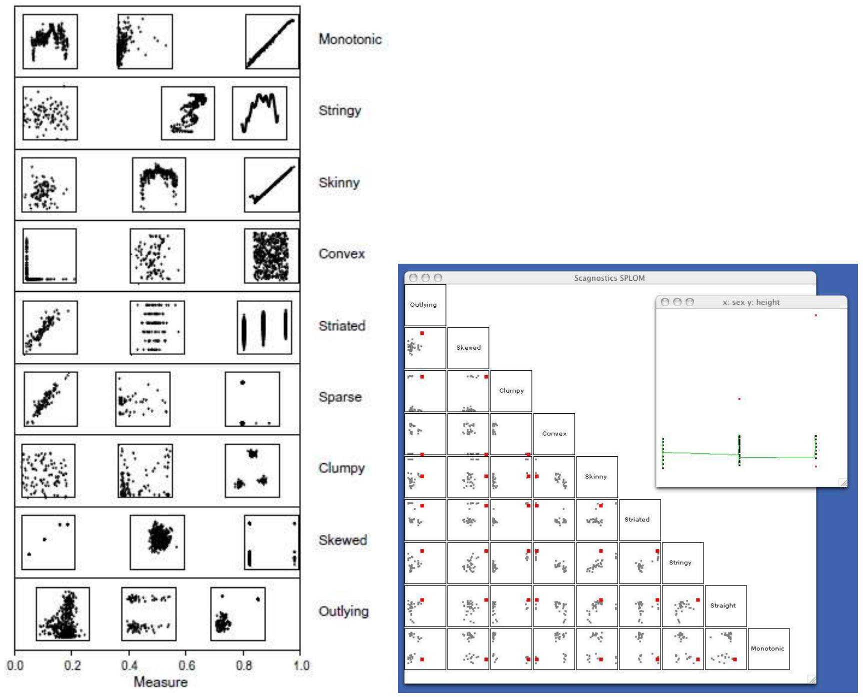


Figure 1.2: 1. indices of nine measures; 2. scagnostics 2D examples[26].

As graph-theory finds more applications in high dimensional data visualization, the demand of higher dimensional scagnostics emerges. Hurley and Oldford (2008) suggested a new navigation infrastructure, which uses 3D scagnostics as indices to guide the navigation. This project aims to provide such a three-dimensional scagnostics solution.

## 1.2 Review of scagnostics 2D

Tukey and Tukey (1985) presented the idea without many details on how to implement it. Wilkinson et al. (2005) examined the nature of the measures and took a different approach from Tukey's suggestion. They pointed out that Tukey's approach can be improved in two aspects, the presumption on continuous empirical or theoretical distribution and the computational complexity ($O(n^3)$). As a conclusion, graph-theoretic measures can be an ideal solution to these problems[26]. They claimed in the paper that,

> "First, the graph-theoretic measures we will use do not presume a connected plane of support. They can be metric over discrete spaces. Second, the measures we will use are $O(n \log n)$ in the number of points because they are based on subsets of the Delaunay triangulation. Third, we employ adaptive hexagon binning before computing our graphs to further reduce the dependence on $n$." [Wilkinson et al. 2005, page 158]

Delaunay triangulation for a set of $P$ points, denote $\mathrm{DT}(P)$, is a method to subdivide $P$ into disjointed triangles such that no point is included in any circumscribed circle of any triangle in $\mathrm{DT}(P)$.

### 1.2.1 Graph-theoretic Approach

The graphs used in the derivation of measures include the convex hull, alpha hull and minimum spanning tree (Figure 1.3). These graphs belong to the subsets of Delaunay triangulation.

- Convex hull
  The convex hull for a set of points $X$ in a real vector space $V$ is the minimal convex set containing $X$. A set of points in $R^2$ is convex if all the straight line segments connecting any pair of its points are contained in the set. In computational geometry, the convex hull is used for the boundary of a set of points.

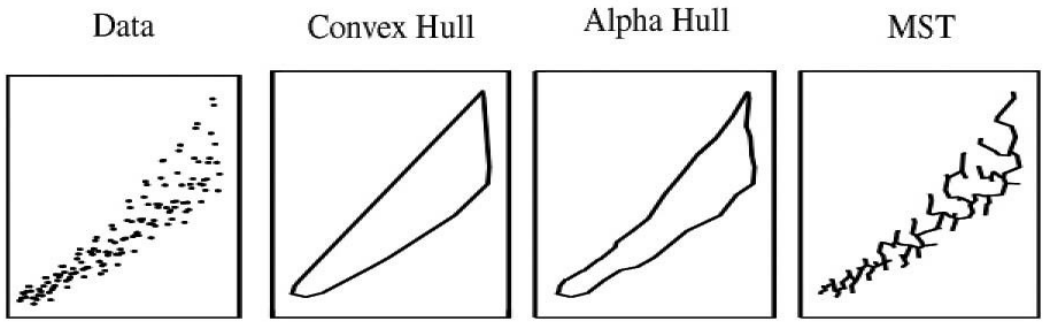


Figure 1.3: Graphs used as bases for computing scagnostics measures, figure taken from [26]

- Alpha hull
  The alpha hull of a set of points is a generalization of its convex hull. For $0 \leq \alpha \leq \infty$, the $\alpha$-hull is defined as the complement of the union of all empty $\alpha$-balls[7]. Alpha hull is a closed polytope.

  Then what is alpha ball? For a positive real alpha, an $\alpha$-ball is an open ball with radius alpha. For $\alpha = 0$, the $\alpha$-ball is a point, and for $\alpha = \infty$, it is an open half-space (for a given fixed point on the surface of the ball) [7]. In 2D, the $\alpha$-ball with radius $0 \leq \alpha \leq \infty$ is a circle. We say an $\alpha$-ball is empty when there is no points enclosed.

  To explain how to find the alpha hull, we need to define alpha shape. An alpha shape describes the geometric shape of a data points set. In an alpha shape graph, an edge exists between any pair of points that can be touched by an empty $\alpha$-ball . The alpha hull can be derived from the boundary of an alpha shape graph [26]. The choice of the $\alpha$ value is critical to the shape of an alpha hull. Figure 1.4 shows the alpha hull for the same points set obtained by specifying different alpha value.

- MST
  A tree is an acyclic, connected, simple graph. Given a connected, undirected graph $G(V, E)$, a spanning tree of $G$ is a subgraph which is a tree that connects all the vertices of $V$. If $G$ is a weighted graph, a minimum spanning tree (MST) of $G$ is a spanning tree with the minimum total edge weights. The edge weights could be any measures of interest, here we take them to be Euclidean distances. There can be more than one minimum spanning tree in a graph. For a graph of $|V|$ vertices, a minimum spanning tree contains $|V| - 1$ edges.

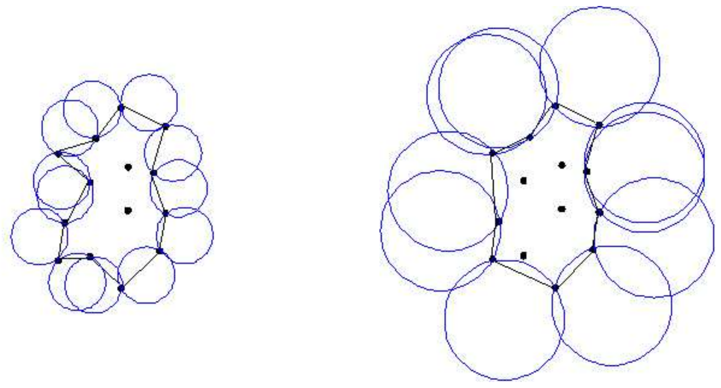


Figure 1.4: Alpha hull for the same points set, given different alpha value

The minimum spanning tree can be found in polynomial time. Common algorithms include Prim (1957) and Kruskal (1956). The average time complexity is usually $O(|V||E|)$ for a graph with $|E|$ edges and $|V|$ vertices. With a well designed data structure such as union-find, it can be improved to $O(|E| \log |E|)$ time [22].

### 1.2.2 Measures

The 2D `scagnostics` package evaluates nine measures on three aspects of the dataset: density, shape and association [17]. In terms of derivation, these measures can also be categorized as being one of three types: MST-based, convex/alpha hull-based and non-graph based.

Wilkinson et al. (2005) [26] gave the notions of geometric features being used in the derivation.

> "The length of an edge, *length*$(e)$, is the Euclidean distance between its vertices.
> The length of a graph, *length*$(G)$, is the sum of the lengths of its edges.
> ...
> The perimeter of a polygon, *perimeter*$(P)$, is the length of its boundary.
> The area of a polygon, *area*$(p)$ is the area of its interior. " [Wilkinson et al. 2005, page 159]

The formula to compute these measures are as follows.

1. Density measures
   - Outlying
   Outlying measures the ratio of outliers to the entire points set. A point is deemed as an outlier if all its associated edges in the MST has length $\geq \omega$, where $\omega = q_{75} + 1.5(q_{75} - q_{25})$, $q_{75}$ and $q_{25}$ are the 75th and the 25th percentile of the MST edge lengths respectively (Tukey, 1977).

$$C_{Outlying} = length(T_{outliers})/length(T) \tag{1.1}$$

   where $T_{outliers}$ = all the edges whose endpoints contain an outlier, $T$ = MST.

   The outliers are found by peeling the MST iteratively. Initially we get the MST for original data and the associated $\omega$. Then the MST is updated (peeled) by extracting the outliers and $\omega$ is changed as well. This procedure is repeated until no outliers can be found for the MST.

   - Skewed
   Skewness is evaluated by the distribution of edge lengths of the minimum spanning tree.

$$C_{Skewed} = (q_{90} - q_{50})/(q_{90} - q_{10}) \tag{1.2}$$

   where $q_{90}$ = 90th percentile of the MST edge lengths; $q_{50}$ = 50th percentile of the MST edge lengths; $q_{10}$ = 10th percentile of the MST edge lengths.

   - Sparse
   Sparse is defined as the 90th percentile of the distribution of edge lengths in the MST.

$$C_{Sparse} = q_{90} \tag{1.3}$$

   - Clumpy
   The measure of clustering is base on the RUNT statistic, which was proposed by Hartigan and Mohanty (1992). Wilkinson et al. (2005) recapped the definition of RUNT as "The runt size of a dendrogram node is the smaller of the number of leaves of each of the two subtrees joined at that node". They further explained that since there is an isomorphism between a single-linkage dendrogram and the MST, a runt size ($r_j$) can be associated with each edge ($e_j$) in the MST [23].

$$C_{Clumpy} = \max_j \left[1 - \max_k \left[length(e_k)\right]/length(e_j)\right] \tag{1.4}$$

where $j$ indexes edges in the MST and $k$ indexes edges in each runt set derived from an edge indexed by $j$.

- Striated
The striated measure concerns the existence of parallel lines. A general measure "based on the number of adjacent edges whose cosine is less than -0.75" is taken [17]. Note here we consider the edges in the peeled MST.

$$C_{Striated} = \frac{1}{|V|} \sum_{v \in V^{(2)}} I(\cos \theta_{e(v,a)e(v,b)} < -.75) \tag{1.5}$$

where $V^{(2)} \subseteq V$ is the set of all vertices of degree 2 in $V$; $I()$ is an indicator function.

2. Shape
- Convex
The convexity measure is based on the ratio of the area of the alpha hull and the area of the convex hull. The $\alpha$ value is taken as the 90th percentile of the peeled MST edge lengths.

$$C_{Convex} = area(A)/area(H) \tag{1.6}$$

where $A$ = alpha hull computed by $\alpha = q_{90}$, $H$ = convex hull.

- Skinny
The skinny measure is based on the ratio of perimeter to the area of the alpha hull.

$$C_{Skinny} = 1 - \sqrt{4\pi area(A)}/perimeter(A) \tag{1.7}$$

- Stringy
Stringy measure depends on peeled minimum spanning tree. It is the proportion of number of vertices of degree 2 to the number of vertices which are not single-degree.

$$C_{Stringy} = \frac{|V^{(2)}|}{|V| - |V^{(1)}|} \tag{1.8}$$

3. Association
- Monotonic
Monotonic measure is the only one not based on the graphs. It was evaluated by squared Spearman correlation coefficient, which is a Pearson correlation on the ranks of $x$ and $y$ [26].

$$C_{Monotonic} = r^2_{Spearman} \tag{1.9}$$

## 1.3 Prospective Applications

### 1.3.1 Graph-theoretic Navigation

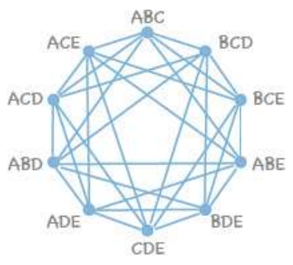
In high dimensional spaces, a visualization task could be divided into two steps. Firstly, we make combinations of subsets and plot them. Usually each contains at most 3 variables and all $\binom{p}{3}$ combinations should be considered. Secondly, walk through all the subsets and examine the pattern transformation. The pattern transformation between these separated subsets encode the additional variables' impact. They can be visually expressed by dynamic navigation. Specifically, a transformation is decomposed into a sequence of basic ones so that only a single variable changes in each move. Graph theory can be used to organize the navigation, see Hurley and Oldford (2008).

### 1.3.2 3d Spaces Graph

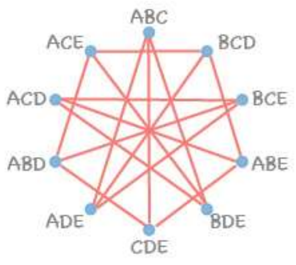
A convenient way to represent the navigating infrastructure is the 3D spaces graph. Hurley and Oldford (2008) defined 3D spaces graph as follows.

> "The 3d space graph is reminiscent of the Kneser graph. The nodes correspond to all $\binom{p}{3}$ subsets of three variables from the $p$ available, but instead of edges between disjoint subsets edges exist between nodes that share a single variable. In general, if the complete graph on the 3d space nodes is $K_n$ where $\binom{n=(p}{3)}$ with vertex set $V$ and edge set $E$, then this graph can be decomposed into 3 distinct graphs $S_0$, $S_1$, and $S_2$ where $S_i$ has vertex set $V$ but edge set $E_i$ and $e \in E_i$ if and only if the vertices of $e$ share exactly $i$ variables. " [Hurley and Oldford, 2008, pp. 10-11]

By this definition, 3d space graphs can be built as shown in Figure 1.5.



(a) The graph of 3d spaces ($S_2$).



(b) Complement of the 3d space graph ($S_1$ since $p = 5$).

Figure 1.5: 3d variable spaces for p = 5 variables.

In the case of $i = 2$, an edge corresponds to a morph of subsets - through a rigid rotation. Once the 3d spaces graph is constructed, we can employ graph traversal algorithm to tour among the nodes or edges. Scagnostics, under this circumstance, can play a role of guide. By assigning the 3D scagnostics measures to the weights of edges, a tour characterized with interesting patterns can be identified.

# Chapter 2

# Scagnostics in 3D

This project follows Wilkinson's graph-theoretic approach and extends these scagnostics to three-dimensional spaces. In this chapter, we will discuss the related graph properties and the derivation of measures in 3D.

First of all, the definition of 3D scagnostics projections could be obtained by extending the 2D definition [26].

Let $\pi_i \times \pi_j \times \pi_k$ denote the projection $R^p \to R^3, x \to (x_i, x_j, x_k)$, Given a set $X \subset R^p$, let $X_{[i,j,k]}$ denote the set $(\pi_i \times \pi_j \times \pi_k)(X) \subset R^3$.

Suppose we have $k$ measures $c_1, c_2, ..., c_k$ defined on sets $X$ in $R^3$, given a data set $X$ containing $n$ points in $R^p$, the 3D scagnostics transform $\tau(X)$ is the data set of $p(p-1)(p-2)/6$ points in $R^k$ containing the following points:

$$(c_1(X_{[i,j,k]}, ..., c_k(X_{[i,j,k]})) \in R^k$$

Intuitively, we could think 3D scagnostics as a method to compute the measures for 3D scatterplots.

## 2.1 3D Graphs

In three-dimensional spaces, the geometric properties of convex hull, alpha hull and minimum spanning tree can also be used to compute the measures. Again, these graphs are subgraphs of Delaunay triangulation in three-dimensional spaces.

### 2.1.1 Convex Hull

The 3D convex hull definition is similar to that defined in the plane. A subset $S \subset R^3$ is convex if all straight line segments with both end points in the subset are contained in $S$. The convex hull in three-dimensional space is a closed polytope. Constructed by Delaunay triangulation, its surface facets consist of triangles.

### 2.1.2 Alpha Hull

The procedure to construct an alpha hull in three-dimensional spaces is identical to that in 2D, except the geometric concepts need to be replaced accordingly. In 3D, an $\alpha-$ ball is an open sphere with radius $\alpha$. The inclusion criterion is revised to "a facet exists between any three corners of a triangle that can be touched by an empty $\alpha$-ball".

### 2.1.3 Minimum Spanning Tree

The concept of minimum spanning tree does not vary with the dimension. However, the Euclidean distances are now calculated in $R^3$ instead of $R^2$. The candidate edges of the MST are still contained in the edges of Delaunay triangulation. Once again, the MST is peeled to get rid of the outliers in three-dimensional spaces.

### 2.1.4 Delaunay Triangulation

In higher dimension, we use the notion $n$-simplex instead of triangles. A $n$-simplex is the convex hull of a set of $(n + 1)$ affinely independent points in Euclidean space of dimension $n$ [9]. When $n = 3$, a simplex is a tetrahedron.

There have been a wide variety of algorithms to build a Delaunay triangulation for a set of points. Nowadays some open source software, such as Qhull and CGAL are available. They are designed for the different purposes. CGAL (Computational Geometry Algorithms Library) provides data structures and algorithms like triangulations, Voronoi diagrams, alpha shapes and convex hull algorithms etc. It is implemented as a C++ library. Qhull offers similar functions such as the convex hull, Delaunay triangulation, Voronoi diagrams etc. It is available in the form of external executables (UNIX shell and Windows executable). Compared to CGAL, Qhull requires less efforts on investigating the uses of the data structures (which are deliberately organized and possibly intricate). Since we just need to compute Delaunay triangulation (other graphs can be derived from DT results), Qhull is a good choice.

## 2.2 Geometric Properties

The combinatorial data of a polytope - vertices, edges and facets in 3D - have corresponding geometric data. The following properties are used in computation of 3D scagnostics measures.

- Volume of tetrahedra
  Given the coordinates of four vertices of a tetrahedron, $(x_i\ ,\ y_i\ ,\ z_i\ )$, $i = 0, 1, 2, 3$, a formula to compute the volume is given by: [13]
  $$V = \frac{1}{6}|det(A)| \tag{2.1}$$
  where $det(\cdot)$ is the determinant and $A = \begin{bmatrix} x_0 & x_1 & x_2 & x_3 \\ y_0 & y_1 & y_2 & y_3 \\ z_0 & z_1 & z_2 & z_3 \\ 1 & 1 & 1 & 1 \end{bmatrix}$

- Area of triangles
  Suppose the length of the three edges are $a$, $b$ and $c$, the formula to compute the triangle's area is:
  $$A = \sqrt{p(p-a)(p-b)(p-c)}, \tag{2.2}$$
  where $p = \frac{1}{2}(a+b+c)$.

- Plane equation
  A general form of a plane is $ax + by + cz + d = 0$, and the unit normal $\boldsymbol{n} = (a,\ b,\ c)$. Three points in general position determine a plane. Given the coordinates of the three points $\boldsymbol{p}_0 = (x_0,\ y_0,\ z_0)$, $\boldsymbol{p}_1 = (x_1,\ y_1,\ z_1)$, $\boldsymbol{p}_2 = (x_2,\ y_2,\ z_2)$, a vector perpendicular to the plane can be calculated by:
  $$\boldsymbol{v = (p_1 - p_0) \times (p_2 - p_1)} \tag{2.3}$$
  where operator $\times$ means the cross product of the two vectors.
  The unit normal $\boldsymbol{n}$ equals to the normalized $\boldsymbol{v}$. The coefficient $d = \boldsymbol{n}\cdot\ \boldsymbol{p}_0$ (dot product of the two vectors).

- Angles between adjacent planes
  The cosine of angle between two adjacent planes in 3D is used in this project.
  $$\cos(\theta) = \frac{a_1a_2 + b_1b_2 + c_1c_2}{\sqrt{a_1^2 + b_1^2 + c_1^2}\sqrt{a_2^2 + b_2^2 + c_2^2}} \tag{2.4}$$
  where $a_i, b_i$ and $c_i$ are the coefficients of plane $i = 1, 2$, as determined by each triangle.

## 2.3 Measures

The nine measures discussed in scagnostics 2D need to be examined in the 3D context. The measures derived from MST are identical to those in 2D. The measures based on the alpha hull and the convex hull need to be adjusted according to the 3D geometric properties.

### 2.3.1 Density Measures

Among the five measures, the outlying, skewed, clumpy and sparse keep are MST-based, therefore no change. The only difference is that Euclidean distance is now taken in $R^3$, not $R^2$. Striated measure needs to be re-evaluated. In three-dimensional spaces, both parallel lines and parallel planes should be considered as striated patterns. Therefore, the measure is revised to count on the angle between adjacent planes formed by three consecutive edges. I took the same threshold as 2D, i.e. $\cos\theta < -.75$ [17]. The striated measure is computed by:

$$C_{Striated} = \frac{1}{(|V|)} \sum_{v \in V^{(\geq 2)}} I(\cos\theta_{p(e,e_1)p(e,e_2)} < -.75) \tag{2.5}$$

where $V^( \geq 2) \subseteq V$, in which all the vertices degree $\geq 2$.

### 2.3.2 Shape

As proposed by Wilkinson et al. (2005), convexity, skinny and stringy describe the shape of the points set. The measures convexity and skinny are computed by geometric metrics of convex hull and alpha hull. Surface area and volume are the natural extensions to the perimeter and area in this context. Stringy measure is not adjusted for being MST-based.

- Convex
  The ratio of the volume of the alpha hull to the volume of the convex hull will be used to assess the convexity in 3D. Similar to the 2D case, it equals 1 if the alpha hull and convex hull has the same volume.

$$C_{Convexity} = volume(A)/volume(H) \tag{2.6}$$

- Skinny
  Conceptually, the degree of skinny is the ratio of volume to the surface area of alpha hull. The ratio is normalized such that a sphere yields 0.

$$C_{Skinny} = 1 - \sqrt[6]{36\pi}\,\sqrt[3]{volume(A)}/\sqrt{surfacearea(A)} \tag{2.7}$$

### 2.3.3 Association

Similar to 2D, the monotonicity is measured by squared partial correlations on the ranks of the variables. I measured the three possible partial correlations and took the maximum as the value of monotonicity.

To compute the partial correlation between ranks of the variables $X$ and $Y$ under condition $Z$ (denote $\rho_{XY\cdot Z}$), I need to get the Spearman's rank correlation coefficient first, which is calculated by:

$$\rho_{XY} = \frac{n(\sum x_i y_i) - \sum x_i \sum y_i}{\sqrt{n \sum (x_i)^2 - (\sum x_i)^2}\sqrt{n \sum (y_i)^2 - (\sum y_i)^2}} \tag{2.8}$$

where $n$ is the number of observations, $x_i$ and $y_i$ are the ranks of raw data $X_i$ and $Y_i$.

Then the partial correlation is given by:

$$\rho_{XY\cdot Z} = \frac{\rho_{XY} - \rho_{XZ}\rho_{YZ}}{\sqrt{1 - (\rho_{XY})^2}\sqrt{1 - (\rho_{YZ})^2}} \tag{2.9}$$

where $\rho_{**}$ is the Spearman's rank correlation coefficient for specified two variables.

Finally, the monotonicity is assessed by:

$$C_{Monotonic} = \max\left[\rho^2_{XY\cdot Z}, \rho^2_{XZ\cdot Y}, \rho^2_{YZ\cdot X}\right] \tag{2.10}$$

# Chapter 3

# Implementation

## 3.1 Infrastructure

### 3.1.1 Framework

Considering the compatibility, we basically kept the infrastructure of `scagnostics` 2D. The main functions of measures computation are included in a Java package. We call the executable files provided by Qhull in Java to deal with Delaunay triangulation [1]. Finally an R script defines function 'scagnostics3D()' as the R interface. For the end users, the only interface is the predefined R functions. Figure 3.1 shows the three layers and the overall framework.

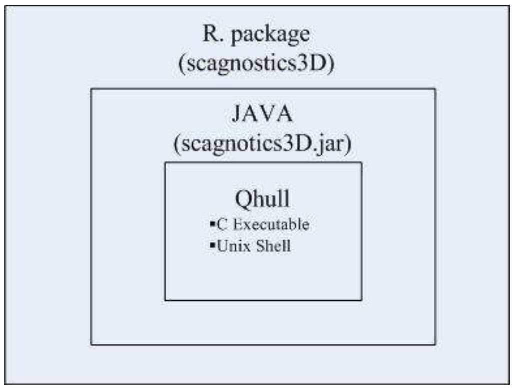


Figure 3.1: Overall framework

### 3.1.2 Setup Instructions

This package was tested on two types of OS: Windows XP and Macintosh. The files for running on Macintosh include *qdelaunay*, *scagnostics3D.jar, cmdfile* and *scag.R*. The files for Windows XP include *qdelaunay, scagnostics3D.jar* and *scag.R*. The setup instructions for both testing environments are as follows.

1. Prerequisites
   R2.7.1 or R2.8.0 is recommended. The `rJava` package is required. Other useful packages include `rggobi` and `scatterplot3d`.

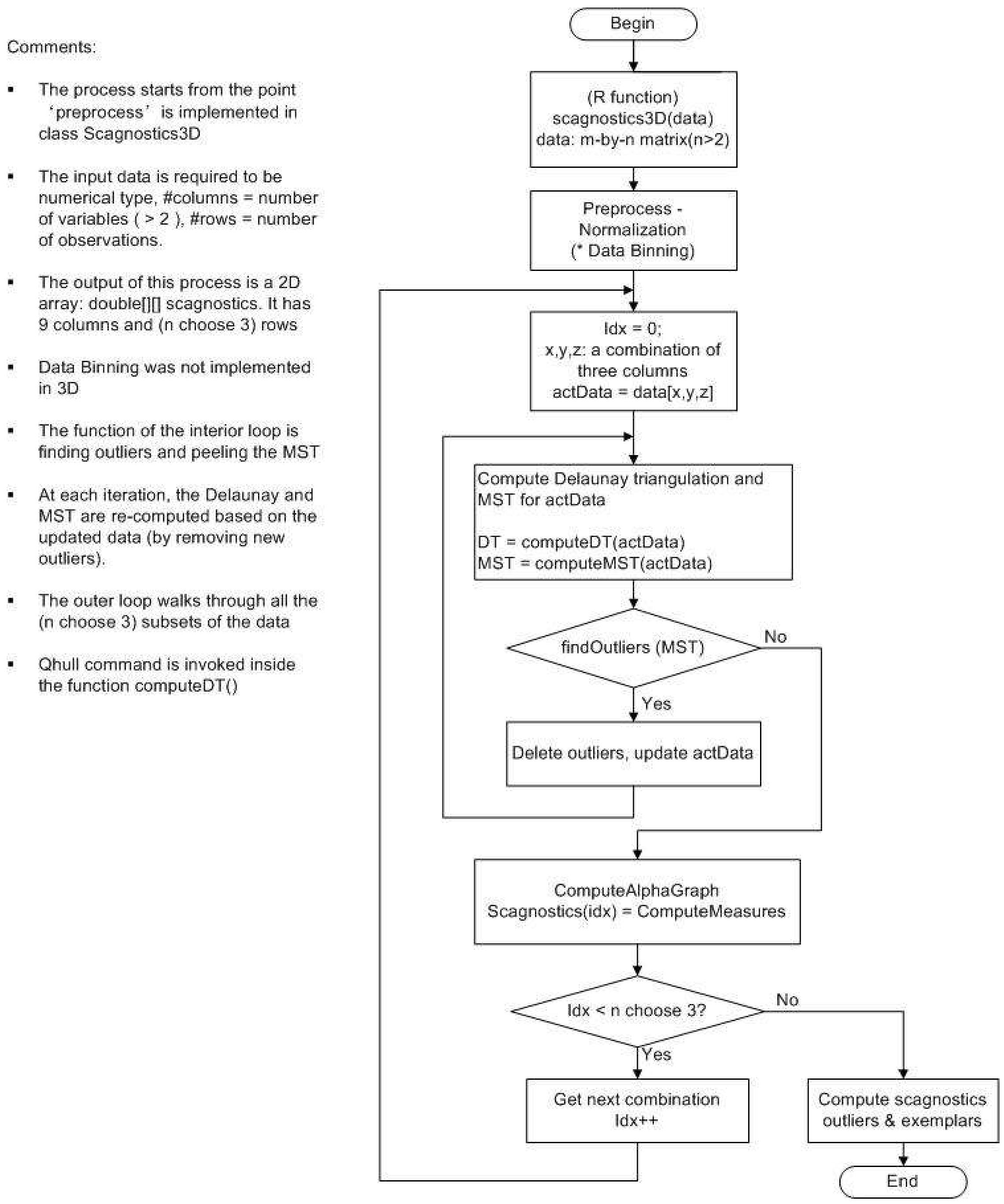
2. Copy files
   If you are using a Macintosh, you should create a folder '/tmp/scagnostics3D/' and copy files *qdelaunay*, *cmdfile* and *scagnostics3D.jar* to this folder.
   If you are using Windows XP, you should create a folder 'C:/scagnostics3D' and copy files *qdelaunay* and *scagnostics3D.jar* to this folder.

3. Define R functions
   Launch R and run the entire scripts in *scag.R*, which will create functions 'scagnostics3D()' for different data types.
   If there are not any error messages during the running, you should read the scagnostics values in R console, which indicates the setup is finished successfully. Now you can create your own test scripts to experience `scagnostics3D`.

## 3.2 Detail Design

### 3.2.1 The Work flow

Figure 3.2 shows the work flow. Major functions are written in Java.


Begin
(R function) scagnostics3D(data) data: m-by-n matrix(n>2)
Preprocess - Normalization (* Data Binning)
Idx = 0; x,y,z: a combination of three columns actData = data[x,y,z]
Compute Delaunay triangulation and MST for actData
DT = computeDT(actData)
MST = computeMST(actData)
findOutliers (MST)
No
Yes
Delete outliers, update actData
ComputeAlphaGraph Scagnostics(idx) = ComputeMeasures
Idx < n choose 3?
No
Yes
Get next combination Idx++
Compute scagnostics outliers & exemplars
End

Comments:

- The process starts from the point 'preprocess' is implemented in class Scagnostics3D
- The input data is required to be numerical type, #columns = number of variables ( > 2 ), #rows = number of observations.
- The output of this process is a 2D array: double[][] scagnostics. It has 9 columns and (n choose 3) rows
- Data Binning was not implemented in 3D
- The function of the interior loop is finding outliers and peeling the MST
- At each iteration, the Delaunay and MST are re-computed based on the updated data (by removing new outliers).
- The outer loop walks through all the (n choose 3) subsets of the data
- Qhull command is invoked inside the function computeDT()

Figure 3.2: Flow chart of the process

### 3.2.2 Data Structures

The primitive elements of 3D Delaunay Triangulation include vertices, edges, facets and tetrahedra. I need to represent the entire set of these elements and their neighbourhood relations [2]. An efficient Delaunay tetrahedralization algorithm requires deeply insight of the complicated relations between the elements [8]. Since we call Qhull to get the DT results, the data structure can be simplified. I ignored the orientation of the embedded components (facets vs. tetrahedron, points vs. facets, etc.) On the other hand, I deliberately selected the neighborhood relations to be included in the data scheme. Figure 3.3 shows the detail class diagram of `scagnostics3D`.

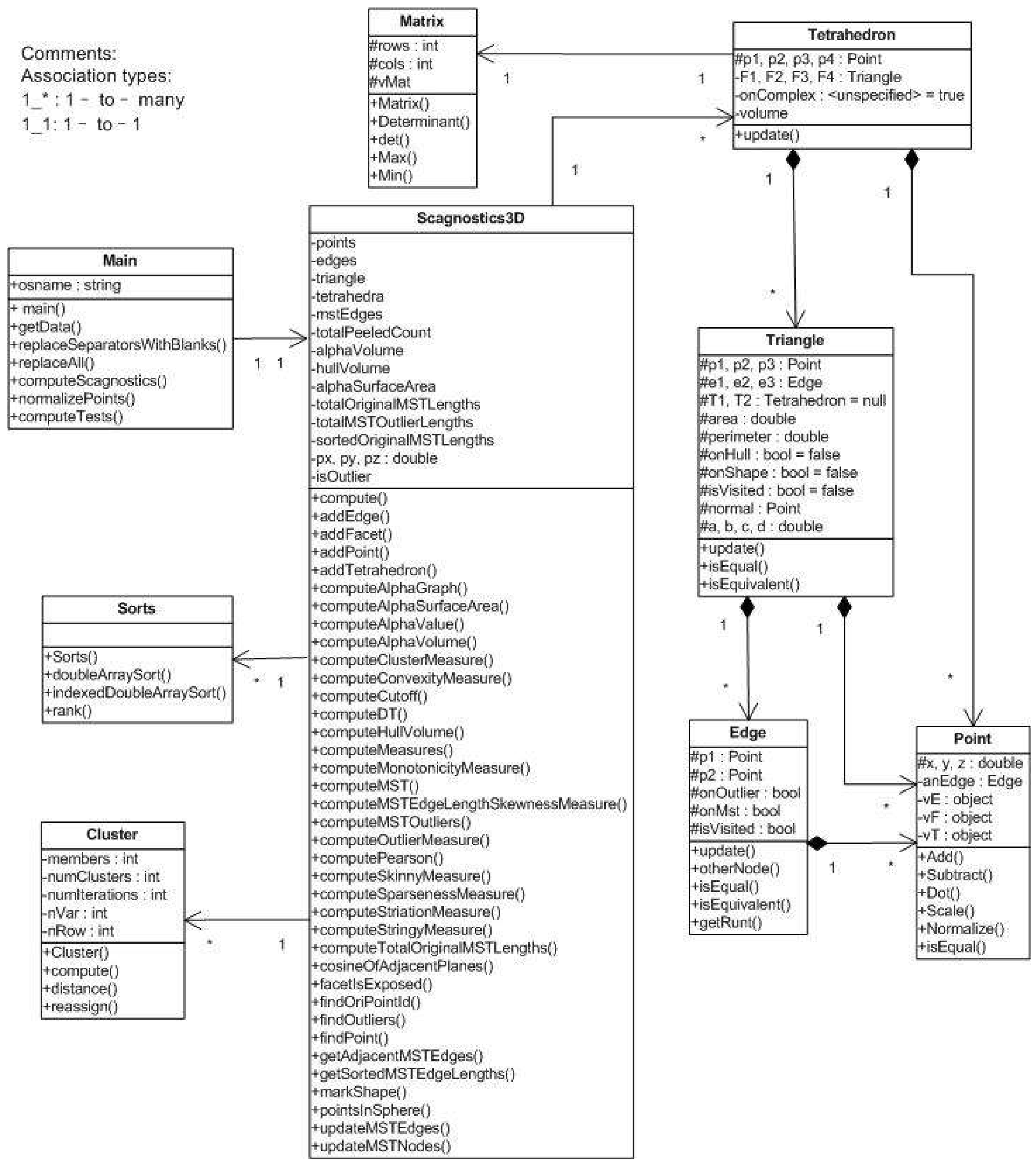


Figure 3.3: Detail class diagram

### 3.2.3 Algorithms

We maintain master lists of tetrahedra, facets, edges, points and MST edges throughout the whole process. As a recursive structure, the data consistency is guaranteed by referring to the master lists whenever a neighbourhood is updated. This section gives pseudo code of some crucial routines including construction of the tetrahedra list and alpha hull.

- Tetrahedra list

**Algorithm 1**: TetrhedraConstruction

```
Data: $file_{DT}$=rec(Point $p1, p2, p3, p4$)
Result: list of tetrahedra, facets, edges and points
open the $file_{DT}$
repeat
    read a line from $file_{DT}$
    $pi$ = new Point $p1, p2, p3, p4$
    $T$ = new Tetrahedron($p1, p2, p3, p4$)
    if $pi$ is not in list points then
        points.add($pi$)
    else
        get id from list points
        assign $T.pi$ = id
    end
    tetrahedra.add($T$)
    $fi = T.F1, T.F2, T.F3, T.F4$
    if $fi$ is not in list facets then
        facets.add($fi$)
    else
        get id from list facets
        assign $T.fi$ = id
    end
    $ei = fi.e1, fi.e2, fi.e3$
    if $ei$ is not in list edges then
        edges.add($fi$)
    else
        get id from list edges
        assign $fi.ei$ = id
    end
until end of $file_{DT}$
```

- Alpha hull

**Algorithm 2**: AlphaHullIdentification

```
Data: $DT$=(List facets, List tetrahedra), $alpha$ real
Result: facets in DT are marked with flag indicating if it is a facet of alpha hull;
        tetrahedra are marked as well.
mark all the DT facets as 'not on alpha shape'
mark all the tetrahedra 'on alpha complex'
repeat
    set flag $delete = false$
    forall $facets\ in\ DT$ do
        current facet $f$ = facets.next
        current tetrahedron $T = f.T1$
        if $alpha \leq$ radius of circumscircle($f$) then
            set delete = true
            mark $T$ as ' not in alpha hull'
        else
            if $f$ is not alpha-exposed then
                set $delete = true$
                mark $T$ as 'not in alpha hull'
            end
        end
    end
until $delete == false$
for $i$=1 to $max$ do  mark facets on $alpha$-shape
mark tetrahedra in alpha hull
```

# Chapter 4

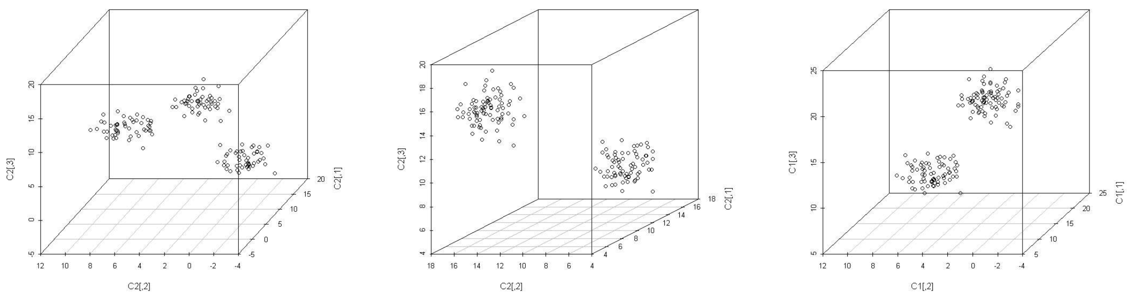
# Experiments

## 4.1 Testing Cases

The data sets with significant patterns were generated following Wilkinson's suggestions [17]. Testing environments are R (recommended version) on Macintosh and Windows XP. The testing results are illustrated and discussed in this section.

- Clumpy
  Clumpy data are generated by translating normally distributed random numbers. The first set of data points are isolated as two clusters, distance in x-axis is 12. The second with x-distance 4 and the third set is divided into three clusters with x-distances 8 and 7.

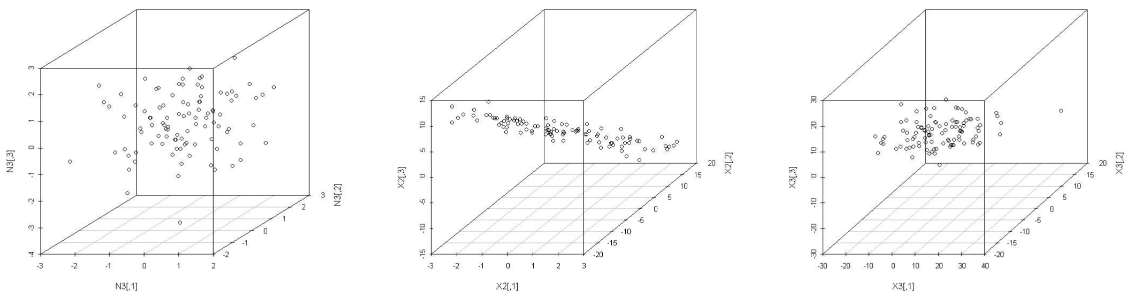
  Comparing the clumpy values of the first case to the second case, it decreases as the distance between the partitions is smaller. More over, the clumpy measure is positively affected by the size of the clump.



0.57 0.80 0.86

Figure 4.1: scatterplots for 150 points: 80-70 clustering; 80-70 with a closer distance; 50-50-50 clustering

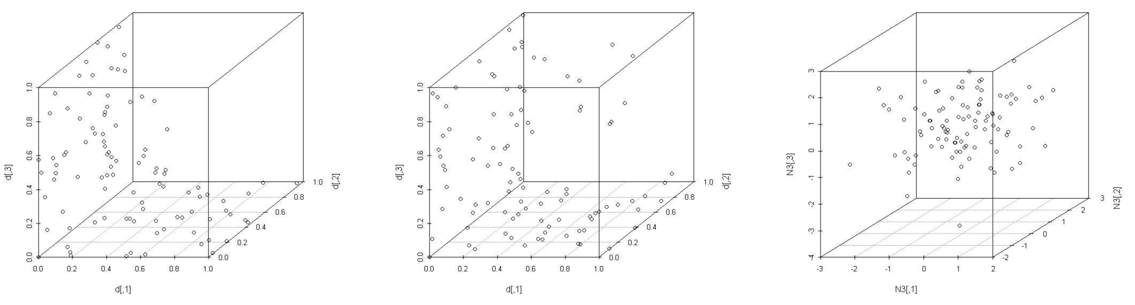
- Skinny
  This group of data are also created on the basis of normal distribution. Take 100 data points and transform the data matrix so that the level of skinny is increasing in sequence.



0.17 0.43 0.94

Figure 4.2: scatterplots for 100 points: normal distribution; a cloud along the diagonal; a thin slice shape

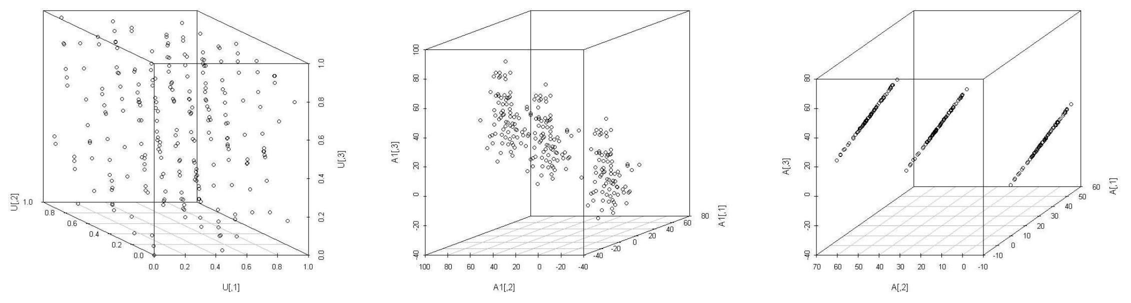
- Convexity
  I created three points sets to test the convexity measure. Figure 4.3 shows the scatterplots and the testing results. From left to right, the index of the convexity is getting larger.



0.12 0.47 0.87

Figure 4.3: scatterplots for 100 points, uniform data along the three coordinates planes; scattered a bit; normal distribution

- Striated
  The first points set is created from uniformly distributed random data. The second and third are generated by translating the skinny data. The most distinguished striated pattern is viewed in the third data set. The values of striated measure reflect this trend.



0.57 0.74 0.84

Figure 4.4: scatterplots for 240 points: many parallel planes; 3 parallel planes; 3 parallel 'lines'

## 4.2 Analysis

The project achieved the goal to compute the measures in three-dimensional spaces. However, there are several potential or known issues.

First, the program still need to be tested with more real data. Currently, all the experiments are conducted on random data, which can not cover all the use cases. It is necessary to develope a completed test suite, designed for different purposes. For example, a test suite covering all the interesting patterns, a test suite to check sensitivity of a signgle index, etc. will be very helpful.

Secondly, the current algorithm to reconstruct the tetrahedra is based on the assumption of general positions. When degeneracy cases occur, the program will throw out an exception. There are many practical methods on degeneracy processing in computational geometry literature. Some common practices are based on approximation, perturbation or transformation [11]. We could utilize these solutions and make further processes in our program. Nevertheless, the side effects need to be carefully coped with.

# Chapter 5

# Conclusions

## 5.1 Summary of the approach

Graph-theoretic scagnostics were developed by Wilkinson et al.. Although designed in planar space these can be easily extended to higher dimensional spaces due to the analytic properties of Delaunay triangulation based graphs.

In this project, I reviewed the related graph theory contents and analyzed the feasibility of the approach in three-dimensional spaces. The classic nine measures on density, shape and association of data points were re-examined under the 3D context. The formula for computing convexity, skinny and monotonic were revised accordingly. The 3D Delaunay triangulation algorithms were investigated.

The Java package (scagnostic3D.jar) and the R interface was implemented and tested. Testing runs on some data sets with typical patterns are satisfactory. We need more experiments for thoroughness test though. Note that the program is not robust enough for no processing of degeneracy cases.

## 5.2 Discussions on the Implementation

The program could be improved in the following aspects:

1. Class design
   The class 'Scagnostics3D' contains core routines for computing the scagnostics, including the process of Delaunay triangulation, convex hull, alpha hull and MST. It

mixed so many functions that made it difficult to read and maintain. Separating these graphs into single classes could be a better design.

2. Work flow
   As mentioned in 3.2.1, the major work flow consists of nested loops, inside which the Delaunay triangulation and MST are computed. This is considered as a time-consuming procedure. Since the main actions in the interior loop are to reconstruct the MST after deleting outliers, we could seek more efficient algorithms than rebuilding the Delaunay triangulation [6].

3. 3D data binning
   Data binning can improve the performance especially for the large number of observations. However, we need to find an efficient three dimensional binning method.

## 5.3 Future Work

We are expecting more applications of graph theory in high dimensional data visualization, and the cognostics as well. Based on this project, some research work on high dimensional navigation could be experimented (Hurley and Oldford, 2008). We could also consider extending scagnostics to a higher dimensional space.

# APPENDICES

# Appendix A

# List of Source Code

## A.1 R Scripts

Remarks:

Basically followed the structure of *scagnostics.R* in `scagnostics` package. Changed the major logic on computing the dimension.

```
library(rJava)

## update this list when scagnostics are added to the Java code!
.scagnostics3D.names <- c("Outlying", "Skewed", "Clumpy", "Sparse", "Striated", "Convex",
"Skinny", "Stringy", "Monotonic")

scagnostics3D <- function(x, ...) UseMethod("scagnostics3D", x)

## calls scagnostics Java code for a given data which must be a Java reference of the class "[[D"
.scagnostics3D.for.data <- function(data, names=NULL) {
  results <- .jcall("scagnostics3D/Main","[[D","computeScagnostics",data)
  r <- lapply(results, .jevalArray)
  n <- length(r)
  s <- length(r[[1]])
  if (n == 1) {
    r <- r[[1]]
    names(r) <- .scagnostics3D.names
  } else {
    r <- matrix(unlist(r), s)
    rownames(r) <- .scagnostics3D.names
    if (!is.null(names)) {
      dn <- length(names)
      l <- data.frame(x=rep(1:dn,(dn*dn)),y=rep(1:dn,each=dn),z=rep(1:dn,each=dn*dn))
      l <- unique(l[l$x < l$y, ]) # collapse to those satisfying x < y
      l <- unique(l[l$y < l$z, ]) # AND those satisfying y < z
      colnames(r) <- paste(names[l$x], "*", names[l$y], "*", names[l$z])
    }
  }
  class(r) <- "scagnostics3D"
  r
}

## --- scagnostics3D methods for different data types ---

scagnostics3D.default <- function(x, y, z,...) {
  if (length(x) != length(y)) stop("x and y must have the same length")
  if (length(y) != length(z)) stop("y and z must have the same length")

  complete <- !is.na(x) & !is.na(y)
  x <- as.double(x[complete])
  y <- as.double(y[complete])
  z <- as.double(z[complete])

  data <- .jarray(list(.jarray(x),.jarray(y),.jarray(z)),"[D")
  .scagnostics3D.for.data(data)
}

scagnostics3D.data.frame <- function(x, ...) {
  if (dim(x)[2] < 2) stop("need at least three variables")
  data <- .jarray(lapply(x, function(x) .jarray(as.double(x))), "[D")
  .scagnostics3D.for.data(data, names(x))
}

scagnostics3D.list <- function(x, ...) {
  if (length(x) < 3) stop("need at least three variables")
```

```
  n <- unlist(lapply(x, length))
  if (!all(n == n[1])) stop("all variables must have the same length")
  data <- .jarray(lapply(x, function(x) .jarray(as.double(x))), "[D")
  nam <- names(x)
  if (is.null(nam)) nam <- as.character(1:length(x))
  .scagnostics3D.for.data(data, nam)
}

scagnostics3D.matrix <- function(x, ...) {
  if (dim(x)[2] < 2) stop("need at least three variables")
  data <- .jarray(lapply(1:(dim(x)[2]), function(i) .jarray(as.double(x[,i]))), "[D")
  nam <- colnames(x)
  if (is.null(nam)) nam <- as.character(1:length(x))
  .scagnostics3D.for.data(data, nam)
}

################################################################################
# testing
require(rggobi)
require(scatterplot3d)

## quote
newwindow <- function(...) {
    gui <- .Platform$GUI
    if (gui=="AQUA") quartz(...) else
      if (gui=="X11") X11(...) else
       if (gui=="Rgui") windows(...) else {
         print("Sorry.  Can't tell what the GUI is to pop the correct window. Will just use
plot(1)")
        plot(1)}
    }
## end


.jinit()

# for Mac
#.jaddClassPath("/tmp/scagnostics3D/scagnostics3D.jar")

# for Windows
.jaddClassPath("C:/scagnostics3D/scagnostics3D.jar")

# test... to make sure the JAR can be loaded
stest <- scagnostics3D(rnorm(10), rnorm(10), rnorm(10))
stest
```

## A.2 Java Code

Remarks on the source code attached:

1. Files taken from `scagnostics` package
   Sort.java (no change)
   Cluster.java (no change)

2. Files keep the `scagnostics` structure
   Scagnostics3D.java (new file, follows scagnostics.java in 2D package. Changed major routines on alpha hull, MST and DT construction )
   Main.java (changed the dimension-related logic)

3. Files newly created for this project
   Matrix.java
   Point.java
   Edge.java
   Triangle.java
   Tetrahedron.java

```java
package scagnostics3D;
import java.util.*;

class Point {
    protected double x, y, z;    // 3D coordinate X,Y,Z
    protected double[] v;        // array of coordinates
    protected List<Edge> vE;          // edges sharing this node
    protected List<Triangle> vF;      // facets sharing this node
    protected List<Tetrahedron> vT; // tetrahedra sharing this node
    protected boolean onMST;
    protected boolean isVisited = false;
    protected int mstDegree;
    protected int pointID;


    protected Point(double x, double y, double z, int pointID) {
        this.x = x;
        this.y = y;
        this.z = z;
        vE = new ArrayList<Edge>();
        vF = new ArrayList<Triangle>();
        vT = new ArrayList<Tetrahedron>();
        this.pointID = pointID;
        this.v = new double[3];
        v[0] = x;
        v[1] = y;
        v[2] = z;
    }

    protected Point(double x, double y, double z) {
        this.x = x;
        this.y = y;
        this.z = z;
        vE = new ArrayList<Edge>();
        vF = new ArrayList<Triangle>();
        vT = new ArrayList<Tetrahedron>();
    }

    protected boolean isEqual(Point v){
        double zero = Double.MIN_VALUE;
        if ((Math.abs(v.x - this.x)<= zero) && (Math.abs(v.y - this.y)<=zero)
                && (Math.abs(v.z - this.z)<=zero))
            return true;
        else
            return false;

    }
    public Point Add(Point n){
        double ax = this.x + n.x;
        double ay = this.y + n.y;
        double az = this.z + n.z;

        return new Point(ax, ay, az);
    }
```

```java
    public Point Subtract(Point n){
        double ax = this.x - n.x;
        double ay = this.y - n.y;
        double az = this.z - n.z;

        return new Point(ax, ay, az);
    }

    public Point Scale(double s){
        double ax = this.x * s;
        double ay = this.y * s;
        double az = this.z * s;
        return new Point(ax, ay, az);
    }

    public double Dot(Point n){
        return (this.x * n.x + this.y * n.y + this.z * n.z);
    }

    public Point Normalize(){
        return Scale( 1/Length() );
    }

    public double Length(){
        return Math.sqrt(Dot(this));
    }

    public Point Cross(Point n){
        return new Point((this.y * n.z - n.y * this.z),
                (this.z * n.x - n.z * this.x),
                (this.x * n.y - n.x * this.y) );
    }

    protected double distToPoint(double px, double py, double pz) {
        double dx = px - x;
        double dy = py - y;
        double dz = pz - z;
        return Math.sqrt(dx * dx + dy * dy + dz * dz);
    }


    protected Iterator<Edge> getNeighborIterator() {
        return vE.iterator();
    }

    protected Edge shortestEdge(boolean mst) {
        Edge emin = null;
        if (vE != null) {
            Iterator<Edge> it = vE.iterator();
            double wmin = Double.MAX_VALUE;
            while (it.hasNext()) {
                Edge e = (Edge) it.next();
                if (mst || !e.otherNode(this).onMST) {
                    double wt = e.weight;
                    if (wt < wmin) {
```

```
                    wmin = wt;
                    emin = e;
                }
            }
        }
    }
    return emin;
}

protected int getMSTChildren(double cutoff, double[] maxLength) {
    int count = 0;
    if (isVisited)
        return count;
    isVisited = true;
    Iterator<Edge> it = vE.iterator();
    while (it.hasNext()) {
        Edge e = (Edge) it.next();
        if (e.onMST) {
            if (e.weight < cutoff) {
                if (!e.otherNode(this).isVisited) {
                    count += e.otherNode(this).getMSTChildren(cutoff, maxLength);
                    double el = e.weight;
                    if (el > maxLength[0])
                        maxLength[0] = el;
                }
            }
        }
    }
    count++;
    return count;
}
}
```

```java
package scagnostics3D;
import java.util.*;

public class Edge {
    protected Point p1, p2;    // start and end point of the edge
    protected Edge nextE = null;   // next edge in the triangle
    protected List <Triangle> inT = null;   // triangle containing this edge
    protected double weight;

    protected boolean onMST = false;
    protected boolean onOutlier = false;
    // sweep all the edges for the striated measure
    protected boolean isVisited = false;

    protected Edge(Point p1, Point p2) {
        update(p1, p2);
    }

    protected void update(Point p1, Point p2) {
        this.p1 = p1;
        this.p2 = p2;
        double dy = p2.y - p1.y;
        double dx = p2.x - p1.x;
        double dz = p2.z - p1.z;
        weight = Math.sqrt(dx * dx + dy * dy + dz * dz);
        inT = new ArrayList<Triangle>();
    }

    protected boolean isEqual(Edge e) {
        return ( (e.p1.isEqual(this.p1) && e.p2.isEqual(this.p2)) );
    }

    protected boolean isEquivalent(Edge e) {
        return ( (e.p1.isEqual(this.p1) && e.p2.isEqual(this.p2)) ||
                (e.p1.isEqual(this.p2) && e.p2.isEqual(this.p1)));
    }

    protected Point otherNode(Point n) {
        if (n.isEqual(p1))
            return p2;
        else
            return p1;
    }

    protected boolean isNewEdge(Point n) {
        Iterator<Edge> it = n.getNeighborIterator();
        while (it.hasNext()) {
            Edge e2 = (Edge) it.next();
            if (e2.isEquivalent(this))
                return false;
        }
        return true;
    }

    protected int getRunts(double[] maxLength) {
```

```java
        double cutoff = weight;
        double[] maxLength1 = new double[1];
        double[] maxLength2 = new double[1];
        int count1 = p1.getMSTChildren(cutoff, maxLength1);
        int count2 = p2.getMSTChildren(cutoff, maxLength2);

        if (count1 < count2) {
            maxLength[0] = maxLength1[0];
            return count1;
        } else if (count1 == count2) {          // take more tightly clustered child
            if (maxLength1[0] < maxLength2[0])
                maxLength[0] = maxLength1[0];
            else
                maxLength[0] = maxLength2[0];
            return count1;
        } else {
            maxLength[0] = maxLength2[0];
            return count2;
        }
    }
}
```

```java
package scagnostics3D;

public class Triangle {
    protected Point p1, p2, p3;
    protected Edge e1, e2, e3;
    // tetrahedra share this facet
    protected Tetrahedron T1 = null, T2 = null;

    protected double area, perimeter;
    protected boolean onHull = false;
    protected boolean onShape = false;
    protected boolean isVisited = false;

    // normal to the plane determined  by this triangle
    protected Point normal ;
    // plane equation parameters. aX+bY+cZ+d=0, normal = (a,b,c)
    protected double a, b, c, d;

    protected Triangle(Point n1, Point n2, Point n3){
        e1 = new Edge(n1, n2);
        e2 = new Edge(n2, n3);
        e3 = new Edge(n3, n1);
        update(e1, e2, e3);
    }

    protected Triangle( Edge a1, Edge a2, Edge a3 ) {
        update(a1, a2, a3);
    }

    public boolean Inside (Point x){
        return normal.Dot(x) > d;
    }

    protected void update(Edge e1, Edge e2, Edge e3) {
        e1.nextE = e2;
        e2.nextE = e3;
        e3.nextE = e1;

        p1 = e1.p1;
        p2 = e1.p2;
        if (e2.p1.isEqual(p2)){
            p3 = e2.p2;
        } else {
            p3 = e2.p1;
        }

        // compute the plane equation coefficients
        normal = p2.Subtract(p1).Cross(p3.Subtract(p1)).Normalize();

        a = normal.x;
        b = normal.y;
        c = normal.z;

        d = normal.Dot(p1);
```

```java
        perimeter = e1.weight + e2.weight + e3.weight;
        double p = 0.5 * perimeter;
        area = Math.sqrt(p * (p - e1.weight) * (p - e2.weight) * (p - e3.weight));
        this.onHull = false;
        this.onShape = false;
    }

    protected boolean isEqual(Triangle t1){
        Point v1 = t1.p1;
        Point v2 = t1.p2;
        Point v3 = t1.p3;
        if (v1.isEqual(this.p1) && v2.isEqual(this.p2) && v3.isEqual(this.p3))
            return true;
        else
            return false;
    }

    protected boolean isEquivalent(Triangle t){
        Triangle t1 = new Triangle(t.p1, t.p3, t.p2);
        Triangle t2 = new Triangle(t.p2, t.p1, t.p3);
        Triangle t3 = new Triangle(t.p2, t.p3, t.p1);
        Triangle t4 = new Triangle(t.p3, t.p1, t.p2);
        Triangle t5 = new Triangle(t.p3, t.p2, t.p1);

        return ( this.isEqual(t) || this.isEqual(t1) || this.isEqual(t2) || this.isEqual(t3)
                || this.isEqual(t4) || this.isEqual(t5));
    }

}
```

**Tetrahedron.java**

```java
package scagnostics3D;

public class Tetrahedron {

    // viewed from n1, n2\n3\n4 in counter-clockwise order
    protected Point p1, p2, p3, p4;
    // facets of this tetrahedron
    protected Triangle F1, F2, F3, F4;
    protected boolean onComplex = true;
    protected double volume;

    protected Tetrahedron(Triangle t, Point p) {
        update(t, p);
    }

    protected Tetrahedron(Point p1, Point p2, Point p3, Point p4) {
        Triangle t = new Triangle(p1, p2, p3);
        update(t, p4);
    }

    protected void update(Triangle t, Point p) {
        this.onComplex = true;
        this.p1 = t.p1;
        this.p2 = t.p2;
        this.p3 = t.p3;
        this.p4 = p;
        this.F1 = t;
        this.F2 = new Triangle(p1, p3, p4);
        this.F3 = new Triangle(p1, p4, p2);
        this.F4 = new Triangle(p3, p2, p4);
        Matrix Mat = new Matrix(p1, p2, p3, p4);
        double vol = Mat.Determinant()/6.0;
        if (vol > 0)
        {
            this.p1 = t.p3;
            this.p3 = t.p1;
        }
        this.volume = Math.abs(vol);
        this.onComplex = true;
    }

}
```

```java
package scagnostics3D;

public class Matrix {
    protected int rows, cols;
    protected double[][] vMat;

public Matrix(double[][] pts){
    rows = pts.length + 1;
    cols = pts[0].length ;
    vMat = new double[rows][cols];

    if (cols > 3) {
        vMat[rows - 1] = new double[]{1, 1, 1, 1};
    } else {
        vMat[rows - 1] = new double[]{1, 1, 1};
    };
    for (int i = 0; i < rows - 1; i++ ){
        System.arraycopy(pts[i], 0, vMat[i], 0, cols);
    }
}

public Matrix(Point n1, Point n2, Point n3, Point n4){
    rows = 4 ;
    cols = 4 ;
    vMat = new double[4][4];
    vMat[0] = new double[]{n1.x, n2.x, n3.x, n4.x};
    vMat[1] = new double[]{n1.y, n2.y, n3.y, n4.y};
    vMat[2] = new double[]{n1.z, n2.z, n3.z, n4.z};
    vMat[3] = new double[]{1, 1, 1, 1};
}

public Matrix(double[] x, double[] y, double[] z) {
    rows = 3 ;
    cols = x.length ;
    vMat = new double[3][cols];
    vMat[0] = x;
    vMat[1] = y;
    vMat[2] = z;
}

public double Determinant(){
    return det(vMat);
}

private double det(double[][] mat) {

    double result = 0;

    if(mat.length == 1) {
        result = mat[0][0];
        return result;
    }

    if(mat.length == 2) {
        result = mat[0][0] * mat[1][1] - mat[0][1] * mat[1][0];
```

```java
        return result;
    }

    for(int i = 0; i < mat[0].length; i++) {
        double temp[][] = new double[mat.length - 1][mat[0].length - 1];

        for(int j = 1; j < mat.length; j++) {
            for(int k = 0; k < mat[0].length; k++) {
                if(k < i) {
                    temp[j - 1][k] = mat[j][k];
                } else if(k > i) {
                    temp[j - 1][k - 1] = mat[j][k];
                }
            }
        }
        result += mat[0][i] * Math.pow(-1, (double)i) * det(temp);
    }

    return result;
}

public double[] Max() {

    double[] maximum = new double[rows]; // start with the first value
    double[] t = null;
    for (int r=0; r<rows; r++) {
        t = vMat[r];

        for (int i=0; i<t.length; i++) {
            if (t[i] > maximum[r]) {
                maximum[r] = t[i];
            }
        }
    }
    return maximum;
}

public double[] Min() {

    double[] minimum = new double[rows]; // start with the first value
    double[] t = null;
    for (int r = 0; r < rows; r++) {
        t = vMat[r];
        for (int i = 0; i < t.length; i++) {
            if (t[i] < minimum[r]) {
                minimum[r] = t[i];
            }
        }
    }
    return minimum;
}

}
```

```java
package scagnostics3D;

import java.util.Arrays;
import java.util.Comparator;

public class Sorts {

    private Sorts() {}

    public static void doubleArraySort(double[] x, int fromIndex, int toIndex) {
        if (fromIndex == toIndex) {
            fromIndex = 0;
            toIndex = x.length;
        }
        Arrays.sort(x, fromIndex, toIndex);
    }

    @SuppressWarnings("unchecked")
    public static int[] indexedDoubleArraySort(final double[] x, int fromIndex, int toIndex) {
        if (fromIndex == toIndex) {
            fromIndex = 0;
            toIndex = x.length;
        }
        Integer[] sortOrder = new Integer[toIndex - fromIndex];
        for (int i = 0; i < sortOrder.length; i++) {
            sortOrder[i] = new Integer(i);
        }
        Arrays.sort(sortOrder, fromIndex, toIndex, new Comparator() {
            public int compare(Object object1, Object object2) {
                int firstIndex = ((Integer) object1).intValue();
                int secondIndex = ((Integer) object2).intValue();
                return Double.compare(x[firstIndex], x[secondIndex]);
            }
        });
        int[] result = new int[sortOrder.length];
        for (int i = 0; i < result.length; i++) {
            result[i] = sortOrder[i].intValue();
        }
        return result;
    }

    public static double[] rank(double[] a) {

        int k, k1, k2, kind, kms, l, lind, n;
        double ak, am, freq;
        boolean insert;

        n = a.length;

        double[] ranks = new double[n];

        int[] index = indexedDoubleArraySort(a, 0, n);

        lind = index[0];
        am = a[lind];
```

```java
        k1 = 0;
        k2 = 0;
        ak = 1.0;
        /* kms allows for missing data */
        kms = 1;
        for (k = 1; k < n; k++) {
            kind = index[k];
            insert = true;
            if (!Double.isNaN(am)) {
                freq = 1.0;
                /*
      if (wt != null)
        freq = Math.floor(wt[kind]);
                */
                kms += (int) freq;
                if (freq > 1.0) {
                    ak += 0.5 * (freq - 1.0);
                    k1 = k;
                    k2 = k;
                } else if (a[kind] == am) {
                    k2 = k;
                    ak += 0.5;
                    if (k < n - 1)
                        insert = false;
                }
                if (insert) {
                    for (l = k1; l <= k2; l++) {
                        lind = index[l];
                        ranks[lind] = ak;
                    }
                    if (k2 != n - 1 && k == n - 1)
                        ranks[kind] = kms;
                }
            }
            if (insert) {
                k1 = k;
                k2 = k;
                ak = kms;
                am = a[kind];
            }
        }
        return ranks;
    }
}
```

```java
package scagnostics3D;
public class Cluster {

    private int[] members;
    private int numClusters;
    private int numIterations;
    private int nVar;
    private int nRow;

    public Cluster(int numClusters, int numIterations) {
        this.numIterations = 3;
        this.numClusters = 0;
        if (numIterations != 0) this.numIterations = numIterations;
        if (numClusters != 0) this.numClusters = numClusters;
    }

    public int[] compute(double[][] data) {
        nRow = data.length;
        nVar = data[0].length;
        boolean useStoppingRule = false;
        double[][] ssr = null;
        if (numClusters == 0) {
            useStoppingRule = true;
            numClusters = 25;
            ssr = new double[numClusters][nVar];
        }

        double[][] center = new double[numClusters][nVar];
        double[][] count = new double[numClusters][nVar];
        double[][] mean = new double[numClusters][nVar];
        double[][] min = new double[numClusters][nVar];
        double[][] max = new double[numClusters][nVar];
        double[][] ssq = new double[numClusters][nVar];
        int[] closestPoints = new int[numClusters];
        double[] closestDistances = new double[numClusters];

        members = new int[nRow];
        int[] mem = new int[nRow];

        for (int k = 0; k < numClusters; k++) {

            /* find best assignments for current number of clusters */
            for (int iter = 0; iter < numIterations; iter++) {
                boolean reassigned = false;
                for (int l = 0; l <= k; l++) {
                    for (int j = 0; j < nVar; j++) {
                        if (iter == 0 || center[l][j] != mean[l][j]) {
                            reassign(k, data, center, count, mean, min, max, ssq,
                                    closestPoints, closestDistances);
                            reassigned = true;
                            break;
                        }
                    }
                    if (reassigned)
                        break;
```

```java
                }
                if (!reassigned || k == 0)
                    break;
            }

            /* check whether we should stop */

            if (useStoppingRule) {
                double ssq1 = 0;
                double ssq2 = 0;
                for (int j = 0; j < nVar; j++) {
                    for (int l = 0; l <= k; l++) {
                        ssq1 += ssr[l][j];
                        ssq2 += ssq[l][j];
                        ssr[l][j] = ssq[l][j];
                    }
                }
                double pre = (ssq1 - ssq2) / ssq1;   //proportional reduction of error
                if (pre > 0 && pre < .1) {
                    numClusters = k;
                    reassign(k, data, center, count, mean, min, max, ssq,
                            closestPoints, closestDistances);
                    System.arraycopy(mem, 0, members, 0, nRow);
                    break;
                } else {
                    System.arraycopy(members, 0, mem, 0, nRow);
                }
            }

            /* now split a cluster to increment number of clusters */

            if (k < numClusters - 1) {
                int kn = k + 1;
                double dmax = 0;
                int jm = 0;
                int km = 0;
                double cutpoint = 0;
                for (int l = 0; l <= k; l++) {
                    for (int j = 0; j < nVar; j++) {
                        double dm = max[l][j] - min[l][j];
                        if (dm > dmax) {
                            cutpoint = mean[l][j];
                            dmax = dm;
                            jm = j;
                            km = l;
                        }
                    }
                }
                for (int i = 0; i < nRow; i++) {
                    if (members[i] == km && data[i][jm] > cutpoint) {
                        for (int j = 0; j < nVar; j++) {
                            count[km][j]--;
                            count[kn][j]++;
                            mean[km][j] -= (data[i][j] - mean[km][j]) / count[km][j];
                            mean[kn][j] += (data[i][j] - mean[kn][j]) / count[kn][j];
                        }
                    }
                }
            }
        }

        int nc = 0;
        double cutoff = .1;
        for (int k = 0; k < numClusters; k++) {
            if (count[k][0] / (double) nRow > cutoff) nc++;
        }

        int[] exemplars = new int[nc];
        nc = 0;
        for (int k = 0; k < numClusters; k++) {
            if (count[k][0] / (double) nRow > cutoff) {
                exemplars[nc] = closestPoints[k];
                nc++;
            }
        }
        return exemplars;
    }

    private void reassign(int nCluster, double[][] data, double[][] center, double[][] count,
            double[][] mean, double[][] min, double[][] max, double[][] ssq,
            int[] closestPoints, double[] closestDistances) {

        /* initialize cluster statistics */

        for (int k = 0; k <= nCluster; k++) {
            closestPoints[k] = -1;
            closestDistances[k] = Double.POSITIVE_INFINITY;
            for (int j = 0; j < nVar; j++) {
                center[k][j] = mean[k][j];
                mean[k][j] = 0;
                count[k][j] = 0;
                ssq[k][j] = 0;
                min[k][j] = Double.POSITIVE_INFINITY;
                max[k][j] = Double.NEGATIVE_INFINITY;
            }
        }

        /* assign each point to closest cluster and update statistics */


        for (int i = 0; i < nRow; i++) {

            /* find closest cluster */
            double dmin = Double.POSITIVE_INFINITY;
            int kmin = -1;
            for (int k = 0; k <= nCluster; k++) {
                double dd = distance(data[i], center[k]);
                if (dd < dmin) {
                    dmin = dd;
                    kmin = k;
```

```java
                    if (dmin < closestDistances[k]) {
                        closestDistances[k] = dmin;
                        closestPoints[k] = i;
                    }
                }
            }
            if (kmin < 0) {
                members[i] = -1;
            } else {
                members[i] = kmin;
            }

            /* update cluster statistics */

            for (int j = 0; j < nVar; j++) {
                if (!Double.isNaN(data[i][j])) {
                    count[kmin][j]++;
                    double xn = count[kmin][j];
                    double xa = data[i][j];
                    mean[kmin][j] += (xa - mean[kmin][j]) / xn;
                    if (xn > 1.)
                        ssq[kmin][j] += xn * (xa - mean[kmin][j]) * (xa - mean[kmin][j]) / (xn -
1.);
                    if (min[kmin][j] > xa)
                        min[kmin][j] = xa;
                    if (max[kmin][j] < xa)
                        max[kmin][j] = xa;
                }
            }
        }
    }

    private double distance(double[] a, double[] b) {
        double dist = 0;
        for (int i = 0; i < a.length; i++) {
            dist += (a[i] - b[i]) * (a[i] - b[i]);
        }
        return dist;
    }
}
```

```java
package scagnostics3D;

import java.util.*;
import java.io.*;

public class Scagnostics3D {

    private List<Point> points;          // nodes set
    private List<Edge> edges;          // edges set
    private List<Triangle> triangles;      // all the triangles involved in the tetrahedralization
    private List<Tetrahedron> tetrahedra;     // tetrahedra set
    private List<Edge> mstEdges;       // minimum spanning tree set
    private int totalPeeledCount;
    private double alphaVolume = 1, alphaSurfaceArea = 1, hullVolume = 1;
    private double totalOriginalMSTLengths;
    private double totalMSTOutlierLengths;
    private double[] sortedOriginalMSTLengths;
    private static int numScagnostics = 9;

    private final static int OUTLYING = 0, SKEWED = 1, CLUMPY = 2, SPARSE = 3,
    STRIATED = 4, CONVEX = 5, SKINNY = 6, STRINGY = 7, MONOTONIC = 8;

    private final static String[] scagnosticsLabels = {"Outlying", "Skewed", "Clumpy", "Sparse",
        "Striated", "Convex", "Skinny", "Stringy", "Monotonic"};

    private double[] px, py, pz;
    private boolean[] isOutlier;
    private double FUZZ = .999;

    public Scagnostics3D(double[] x, double[] y, double[] z) {
        points = new ArrayList<Point>();
        edges = new ArrayList<Edge>();
        triangles = new ArrayList<Triangle>();
        tetrahedra = new ArrayList<Tetrahedron>();
        mstEdges = new ArrayList<Edge>();
        px = x;
        py = y;
        pz = z;
    }

    public double[] compute() {

        if (px.length < 3)
            return null;
        double xx = px[0];
        double yy = py[0];
        double zz = pz[0];
        boolean isXConstant = true;
        boolean isYConstant = true;
        boolean isZConstant = true;
        for (int i = 1; i < px.length; i++) {
            if (px[i] != xx) isXConstant = false;
            if (py[i] != yy) isYConstant = false;
            if (pz[i] != zz) isZConstant = false;
        }
```

```java
        if (isXConstant || isYConstant || isZConstant)
            return null;

        findOutliers();
        computeAlphaGraph();
        computeAlphaVolume();
        computeAlphaSurfaceArea();
        computeHullVolume();

        return computeMeasures();
    }

    public static int getNumScagnostics() {
        return scagnosticsLabels.length;
    }

    public static String[] getScagnosticsLabels() {
        return scagnosticsLabels;
    }

    public static boolean[] computeScagnosticsExemplars(double[][] pts) {
        int nPts = pts.length;
        if (nPts < 3)
            return null;
        Cluster c = new Cluster(0, 0);
        int[] exemp = c.compute(pts);
        boolean[] exemplars = new boolean[nPts];
        for (int i = 0; i < exemp.length; i++)
            exemplars[exemp[i]] = true;
        return exemplars;
    }

    public static boolean[] computeScagnosticsOutliers(double[][] pts) {

        // Prim's algorithm
        int nPts = pts.length;      // p*(p-1)/2 points representing pairwise scatterplots
        int nVar = pts[0].length;   // number of scagnostics (9)
        if (nPts < 2)
            return null;
        int[][] edges = new int[nPts - 1][2];
        int[] list = new int[nPts];
        int[] degrees = new int[nPts];
        double[] cost = new double[nPts];
        double[] lengths = new double[nPts - 1];

        list[0] = 0;
        cost[0] = Double.POSITIVE_INFINITY;
        int cheapest = 0;

        for (int i = 1; i < nPts; i++) {
            for (int j = 0; j < nVar; j++) {
                double d = pts[i][j] - pts[0][j];
                cost[i] += d * d;
            }
            if (cost[i] < cost[cheapest])
```

```
                cheapest = i;
        }
        for (int j = 1; j < nPts; j++) {
            int end = list[cheapest];
            int jp = j - 1;
            edges[jp][0] = cheapest;
            edges[jp][1] = end;
            lengths[jp] = cost[cheapest];
            degrees[cheapest]++;
            degrees[end]++;
            cost[cheapest] = Double.POSITIVE_INFINITY;
            end = cheapest;

            for (int i = 1; i < nPts; i++) {
                if (cost[i] != Double.POSITIVE_INFINITY) {
                    double dist = 0.;
                    for (int k = 0; k < nVar; k++) {
                        double d = pts[i][k] - pts[end][k];
                        dist += d * d;
                    }
                    if (dist < cost[i]) {
                        list[i] = end;
                        cost[i] = dist;
                    }
                    if (cost[i] < cost[cheapest]) cheapest = i;
                }
            }
        }
        double cutoff = findCutoff(lengths);
        boolean[] outliers = new boolean[nPts];
        for (int i = 0; i < nPts; i++)
            outliers[i] = true;
        for (int i = 0; i < nPts - 1; i++) {
            if (lengths[i] < cutoff) {
                for (int k = 0; k < 2; k++) {
                    int node = edges[i][k];
                    outliers[node] = false;
                }
            }
        }
        return outliers;
    }

    private void clear() {
        points.clear();
        edges.clear();
        triangles.clear();
        tetrahedra.clear();
        mstEdges.clear();
    }

    private void findOutliers() {
        //this.counts = bdata.getCounts();
        isOutlier = new boolean[px.length];
        computeDT();
```

```
        computeMST();
        sortedOriginalMSTLengths = getSortedMSTEdgeLengths();
        double cutoff = computeCutoff(sortedOriginalMSTLengths);
        computeTotalOriginalMSTLengths();

        boolean foundNewOutliers = computeMSTOutliers(cutoff);
        double[] sortedPeeledMSTLengths;
        while (foundNewOutliers) {
            clear();
            computeDT();
            computeMST();
            sortedPeeledMSTLengths = getSortedMSTEdgeLengths();
            cutoff = computeCutoff(sortedPeeledMSTLengths);
            foundNewOutliers = computeMSTOutliers(cutoff);
        }
    }

    private double[] computeMeasures() {
        double[] results = new double[numScagnostics];
        // Do not change order of these calls!
        results[OUTLYING] = computeOutlierMeasure();
        results[CLUMPY] = computeClusterMeasure();
        results[SKEWED] = computeMSTEdgeLengthSkewnessMeasure();
        results[CONVEX] = computeConvexityMeasure();
        results[SKINNY] = computeSkinnyMeasure();
        results[STRINGY] = computeStringyMeasure();
        results[STRIATED] = computeStriationMeasure();
        results[SPARSE] = computeSparsenessMeasure();
        results[MONOTONIC] = computeMonotonicityMeasure();
        return results;
    }

    private void computeDT() {

        String fp = "/tmp/scagnostics3D/qhull/points.txt";
        String fres1 = "/tmp/res1";

        boolean isWindows = false;
        if (Main.osname.equalsIgnoreCase("Windows XP")){
            fp = "points.txt";
            fres1 = "res1";
            isWindows = true;
        }
        totalPeeledCount = 0;

        double[] tpx, tpy, tpz;
        tpx = new double[totalPeeledCount];
        tpy = new double[totalPeeledCount];
        tpz = new double[totalPeeledCount];

        // Stream to write file
        FileOutputStream fout;
        try
        {
            fout = new FileOutputStream (fp);
            for (int i = 0; i < px.length; i++) {
                if (!isOutlier[i]) {
                    totalPeeledCount ++;
                }
            }
            tpx = new double[totalPeeledCount];
            tpy = new double[totalPeeledCount];
            tpz = new double[totalPeeledCount];

            new PrintStream(fout).println ("3");
            new PrintStream(fout).println (totalPeeledCount);
            int j = 0;
            for (int i = 0; i < px.length; i++) {
                if (!isOutlier[i]) {
                    new PrintStream(fout).println (px[i]+ " " + py[i] + " " + pz[i]);
                    tpx[j] = px[i];
                    tpy[j] = py[i];
                    tpz[j] = pz[i];
                    j++;
                }
            }
            fout.close();
        }

        catch (IOException e)
        {
            System.err.println ("Unable to write to file");
            System.exit(-1);
        }

// UNIX/MAC VERSION:
        if (!isWindows) {
            String cmd = "sh /tmp/scagnostics3D/qhull/cmdfile";

            try {
                Process process = Runtime.getRuntime().exec(cmd);
                int exitval = process.waitFor();

                BufferedReader buf = new BufferedReader(new InputStreamReader(process.
getInputStream()));
                String line = "";
                while ((line=buf.readLine())!=null) {
                    System.out.println(line);
                }

                if (exitval == 0){
                    process.destroy();
                }
            }
            catch (Exception e) {
                e.printStackTrace();
            }
        }
//      WINDOWS VERSION:
        else {
```

```
            String[] cmd1 = new String[] { "cmd.exe","/C",
                    "more points.txt | qdelaunay.exe i Qt TO res1"};

            try {

                Process process = Runtime.getRuntime().exec(cmd1);
                int exitval = process.waitFor();
                if (exitval == 0){
                    process.destroy();
                }
            }
            catch (Exception e) {
                e.printStackTrace();
            }

        }
        // read data from res1(delaunay) and res2(convex hull)
        int[][] tetra = getData(new File(fres1), false);
        for (int i=0; i<tetra.length;i++){
            int pt0 = tetra[i][0];
            int pt1 = tetra[i][1];
            int pt2 = tetra[i][2];
            int pt3 = tetra[i][3];
            Point p0 = new Point(tpx[pt0], tpy[pt0], tpz[pt0], pt0);
            Point p1 = new Point(tpx[pt1], tpy[pt1], tpz[pt1], pt1);
            Point p2 = new Point(tpx[pt2], tpy[pt2], tpz[pt2], pt2);
            Point p3 = new Point(tpx[pt3], tpy[pt3], tpz[pt3], pt3);

            addTetrahedron(new Tetrahedron(p0, p1, p2, p3));
        }
    }

    private static int[][] getData(File fname, boolean isFacet ) {
        java.io.BufferedReader fin;
        try {
            fin = new java.io.BufferedReader(new java.io.FileReader(fname));
        } catch (java.io.FileNotFoundException fe) {
            javax.swing.JOptionPane.showMessageDialog(null, "File not found!", "Alert",
                    javax.swing.JOptionPane.ERROR_MESSAGE);
            return null;
        }
        try {
            int numVars = 0;
            if (isFacet)
                numVars = 3;
            else
                numVars = 4;

            int numRows = 0;

            String record;
            record = fin.readLine();
            if (record == null)
                return null;
            record = replaceSeparatorsWithBlanks(record);
```

```java
            StringTokenizer st = new StringTokenizer(record, " ");
            numRows = Integer.parseInt(st.nextToken());

            int[][] data = new int[numRows][numVars];
            int j = 0;
            record = fin.readLine();

            while (record != null) {
                record = replaceSeparatorsWithBlanks(record);
                st = new StringTokenizer(record, " ");

                for (int i = 0; i < numVars; i++) {
                    try {
                        String tmp = st.nextToken();
                        data[j][i] = Integer.parseInt(tmp);
                    } catch (Exception ie) {
                        javax.swing.JOptionPane.showMessageDialog(null,
                                "Error reading from the file", "Alert",
                                javax.swing.JOptionPane.ERROR_MESSAGE);
                        return null;
                    }
                }
                record = fin.readLine();
                j++;
            }
            fin.close();

            return data;
        } catch (java.io.IOException ie) {
            javax.swing.JOptionPane.showMessageDialog(null,
                    "Error reading from the file", "Alert",
                    javax.swing.JOptionPane.ERROR_MESSAGE);
            return null;
        }
    }

    private void computeMST() {

        if (points.size() > 1) {
            List<Point> mstNodes = new ArrayList<Point>();
            Point mstNode = points.get(0);
            updateMSTNodes(mstNode, mstNodes);
            int count = 1;
            while (count < points.size()) {
                Edge addEdge = null;
                double wmin = Double.MAX_VALUE;
                Point nmin = null;
                Iterator<Point> mstIterator = mstNodes.iterator();
                while (mstIterator.hasNext()) {
                    mstNode = mstIterator.next();
                    Edge candidateEdge = mstNode.shortestEdge(false);
                    if (candidateEdge != null) {
                        double wt = candidateEdge.weight;
                        if (wt < wmin) {
                            wmin = wt;
```

```java
                            nmin = mstNode;
                            addEdge = candidateEdge;
                        }
                    }
                }
                if (addEdge != null) {
                    Point addNode = addEdge.otherNode(nmin);
                    int ptid = findPoint(addNode, points);
                    addNode = points.get(ptid);
                    int k;
                    for (k = 0; k < addNode.vE.size(); k++){
                        if (addNode.vE.get(k).isEquivalent(addEdge))
                            break;
                    }
                    addNode.vE.get(k).p1.onMST = true;
                    addNode.vE.get(k).p2.onMST = true;
                    addNode.vE.get(k).onMST = true;
                    updateMSTNodes(addNode, mstNodes);
                    updateMSTEdges(addEdge);
                }
                count++;
            }
        }
    }
    private static String replaceSeparatorsWithBlanks(String record) {
        record = replaceAll(record, ",,", ",~,");
        record = replaceAll(record, "\t\t", "\t~\t");
        record = replaceAll(record, ",", " ");
        record = replaceAll(record, "\t", " ");
        return record;
    }
    private static String replaceAll(String source, String toReplace, String replacement) {
        int idx = source.lastIndexOf(toReplace);
        if (idx != -1) {
            StringBuffer sb = new StringBuffer(source);
            sb.replace(idx, idx + toReplace.length(), replacement);
            while ((idx = source.lastIndexOf(toReplace, idx - 1)) != -1) {
                sb.replace(idx, idx + toReplace.length(), replacement);
            }
            source = sb.toString();
        }

        return source;
    }
    private static double findCutoff(double[] distances) {
        int[] index = Sorts.indexedDoubleArraySort(distances, 0, 0);
        int n50 = distances.length / 2;
        int n25 = n50 / 2;
        int n75 = n50 + n50 / 2;
        return distances[index[n75]] + 1.5 * (distances[index[n75]] - distances[index[n25]]);
    }

    private boolean computeMSTOutliers(double omega) {

        boolean found = false;
        Iterator<Point> it = points.iterator();
        while (it.hasNext()) {
            Point n = it.next();
            Iterator<Edge> ie = n.vE.iterator();
            boolean delete = true;
            while (ie.hasNext()) {
                Edge e = ie.next();
                if (e.onMST && (e.weight < omega) )
                {
                    delete = false;
                    break;
                }
            }
            if (delete) {
                ie = n.vE.iterator();
                double sumlength = 0;
                while (ie.hasNext()) {
                    Edge e = ie.next();
                    if (e.onMST && !e.onOutlier){
                        sumlength += e.weight;
                        e.onOutlier = true;
                    }
                }
                totalMSTOutlierLengths += sumlength;
                int opid = findOriPointId(n);
                isOutlier[opid] = true;
                found = true;
            }
        }
        return found;
    }


    private int findOriPointId(Point p) {
        for (int i=0; i < px.length; i++){
            Point temp = new Point(px[i], py[i], pz[i]);
            if ( temp.isEqual(p) )
                return i;
        }
        return -1;
    }

    private double computeCutoff(double[] lengths) {
        if (lengths.length == 0) return 0;
        int n50 = lengths.length / 2;
        int n25 = n50 / 2;
        int n75 = n50 + n25;
        return lengths[n75] + 1.5 * (lengths[n75] - lengths[n25]);
    }

    private double computeAlphaValue() {
        int length = sortedOriginalMSTLengths.length;
        if (length == 0) return 100.;
        int n90 = (9 * length) / 10;
        double alpha = sortedOriginalMSTLengths[n90];
```

```
        return Math.min(alpha, 100.);
    }

    private double computeMSTEdgeLengthSkewnessMeasure() {
        if (sortedOriginalMSTLengths.length == 0)
            return 0;
        int n = sortedOriginalMSTLengths.length;
        int n50 = n / 2;
        int n10 = n / 10;
        int n90 = (9 * n) / 10;
        double skewness = (sortedOriginalMSTLengths[n90] - sortedOriginalMSTLengths[n50]) /
        (sortedOriginalMSTLengths[n90] - sortedOriginalMSTLengths[n10]);
        return skewness;
    }

    private void updateMSTEdges(Edge addEdge) {

        addEdge.onMST = true;
        int ptid = findPoint(addEdge.p1, points);
        addEdge.p1 = points.get(ptid);
        ptid = findPoint(addEdge.p2, points);
        addEdge.p2 = points.get(ptid);
        addEdge.p1.mstDegree++;
        addEdge.p2.mstDegree++;
        mstEdges.add(addEdge);
    }

    private void updateMSTNodes(Point addNode, List<Point> mstNodes) {
        mstNodes.add(addNode);
        addNode.onMST = true;
        //sweep all mstNodes
        for (int i=0; i< mstNodes.size(); i++){
            Point p = mstNodes.get(i);
            for (int j=0; j< p.vE.size(); j++){
                if (p.vE.get(j).p1.isEqual(addNode))
                    p.vE.get(j).p1.onMST = true;
                if (p.vE.get(j).p2.isEqual(addNode))
                    p.vE.get(j).p2.onMST = true;
            }
        }
    }

    private double[] getSortedMSTEdgeLengths() {
        double[] lengths = new double[mstEdges.size()];
        for (int i=0; i< mstEdges.size(); i++){
            lengths[i] = mstEdges.get(i).weight;
        }

        Sorts.doubleArraySort(lengths, 0, 0);
        return lengths;
    }

    private void computeTotalOriginalMSTLengths() {
        for (int i = 0; i < sortedOriginalMSTLengths.length; i++)
            totalOriginalMSTLengths += sortedOriginalMSTLengths[i];
```

```java
    }

    private double computeOutlierMeasure() {
        return totalMSTOutlierLengths / totalOriginalMSTLengths;
    }

    private boolean pointsInSphere(Point n, Point center, double radius) {
        double xc = center.x;
        double yc = center.y;
        double zc = center.z;

        double r = FUZZ * radius;
        int pn = findPoint(n, points);
        // pn >= 0 always true
        n.vE = points.get(pn).vE;
        Iterator<Edge> i = n.vE.iterator();
        while (i.hasNext()) {
            Edge e = i.next();
            Point no = e.otherNode(n);
            double dist = no.distToPoint(xc, yc, zc);
            if (dist < r)
                return true;
        }
        return false;
    }

    private void computeAlphaGraph() {  // requires initializing SEdge.onShape = false

        boolean deleted;
        double alpha = computeAlphaValue();
        int tid = -1, tid2 = -1;
        do {
            Iterator<Triangle> i = triangles.iterator();
            deleted = false;
            while (i.hasNext()) {
                Triangle f = i.next();
                tid = tetrahedra.indexOf(f.T1);
                f.T1 = tetrahedra.get(tid);
                if (f.T2 != null){
                    tid2 = tetrahedra.indexOf(f.T2);
                    f.T2 = tetrahedra.get(tid2);
                }

                if (f.T1.onComplex) {
                    if (f.T2 != null)
                    {
                        if (f.T2.onComplex)
                            continue;
                    }
                    double minradius = f.e1.weight*f.e2.weight*f.e3.weight/(4*f.area);
                    if (alpha < minradius) {
                        f.T1.onComplex = false;
                        tetrahedra.get(tid).onComplex = false;
                        deleted = true;
                    } else if (!facetIsExposed(alpha, f)) {
```

```java
                        f.T1.onComplex = false;
                        tetrahedra.get(tid).onComplex = false;
                        deleted = true;
                    }
                }
            }
        } while (deleted);
        markShape();
    }

    private void markShape() {
        Iterator<Triangle> i = triangles.iterator();
        while (i.hasNext()) {
            Triangle f = i.next();
            if (f.T1.onComplex) {
                if (f.T2 == null)
                    f.onShape = true;
                else if (!f.T2.onComplex)
                    f.onShape = true;
            }
            else if ((f.T2 != null) && (f.T2.onComplex)){
                f.onShape = true;
            }
        }
    }

    private boolean facetIsExposed(double alpha, Triangle f) {

        // circumcircle of triangle f - radius and center
        // denominator
        double dn = ((f.p1.Subtract(f.p2)).Cross(f.p2.Subtract(f.p3))).Length();
        double a1 = Math.pow(f.p2.Subtract(f.p3).Length(),2) * (f.p1.Subtract(f.p2)).Dot(f.p1.
Subtract(f.p3));
        double a2 = Math.pow(f.p1.Subtract(f.p3).Length(),2) * (f.p2.Subtract(f.p1)).Dot(f.p2.
Subtract(f.p3));
        double a3 = Math.pow(f.p1.Subtract(f.p2).Length(),2) * (f.p3.Subtract(f.p1)).Dot(f.p3.
Subtract(f.p2));
        a1 = 0.5 * a1/(dn*dn);
        a2 = 0.5 * a2/(dn*dn);
        a3 = 0.5 * a3/(dn*dn);
        Point pc = (f.p1.Scale(a1).Add(f.p2.Scale(a2)).Add(f.p3.Scale(a3)));
        double rd = f.e1.weight*f.e2.weight*f.e3.weight/(4*f.area);
        double dis = Math.sqrt(alpha*alpha - rd*rd);
        Point sc1 = pc.Add((f.normal).Scale(dis));
        Point sc2 = pc.Subtract((f.normal).Scale(dis));

        boolean pointsInSphere1 = pointsInSphere(f.p1, sc1, alpha) || pointsInSphere(f.p2, sc1,
alpha) || pointsInSphere(f.p3, sc1, alpha);
        boolean pointsInSphere2 = pointsInSphere(f.p1, sc2, alpha) || pointsInSphere(f.p2, sc2,
alpha) || pointsInSphere(f.p3, sc2, alpha);
        return !(pointsInSphere1 && pointsInSphere2);
    }

    private double computeStringyMeasure() {
        int count1 = 0;
        int count2 = 0;
        Iterator<Point> it = points.iterator();
        while (it.hasNext()) {
            Point n = it.next();
            if (n.mstDegree == 1)
                count1++;
            if (n.mstDegree == 2)
                count2++;
        }
        double result = (double) count2 / (double) (points.size() - count1);
        return result * result * result;
    }

    private double computeClusterMeasure() {
        Iterator<Edge> it = mstEdges.iterator();
        double[] maxLength = new double[1];
        double maxValue = 0;
        while (it.hasNext()) {
            Edge e = it.next();
            clearVisits();
            e.onMST = false;  // break MST at this edge
            int runts = e.getRunts(maxLength);
            e.onMST = true;   // restore this edge to MST
            if (maxLength[0] >= 0) {
                double value = runts * (1 - maxLength[0] / e.weight);
                if (value > maxValue)
                    maxValue = value;
            }
        }
        return 2 * maxValue / totalPeeledCount;
    }

    private void clearVisits() {
        Iterator<Point> it = points.iterator();
        while (it.hasNext()) {
            Point n = it.next();
            n.isVisited = false;
        }
    }

    private double computeMonotonicityMeasure() {
        int n = px.length;
        double[] ax = new double[n];
        double[] ay = new double[n];
        double[] az = new double[n];

        double[] weights = new double[n];
        for (int i = 0; i < n; i++) {
            ax[i] = px[i];
            ay[i] = py[i];
            az[i] = pz[i];
            weights[i] = 0;
            Point temp = new Point(ax[i], ay[i], az[i]);
            for (int l=0; l<points.size(); l++)
```

```
            {
                if (temp.isEqual(points.get(l)))
                    weights[i] += 1;
            }
        }
        double[] rx = Sorts.rank(ax);
        double[] ry = Sorts.rank(ay);
        double[] rz = Sorts.rank(az);

        double s_xy = computePearson(rx, ry, weights);
        double s_xz = computePearson(rx, rz, weights);
        double s_yz = computePearson(ry, rz, weights);
        double rho1 = (s_xy - s_xz * s_yz)/Math.sqrt((1-s_xz * s_xz)*(1-s_yz * s_yz));
        double rho2 = (s_xz - s_xy * s_yz)/Math.sqrt((1-s_xy * s_xy)*(1-s_yz * s_yz));
        double rho3 = (s_yz - s_xy * s_xz)/Math.sqrt((1-s_xy * s_xy)*(1-s_xz * s_xz));
        double rho = Math.max(rho1*rho1,rho2*rho2);
        return Math.max(rho,rho3*rho3);
    }

    private double computePearson(double[] x, double[] y, double[] weights) {
        int n = x.length;
        double xmean = 0;
        double ymean = 0;
        double xx = 0;
        double yy = 0;
        double xy = 0;
        double sumwt = 0;
        for (int i = 0; i < n; i++) {
            double wt = weights[i];
            if (wt > 0 && !isOutlier[i]) {
                sumwt += wt;
                xx += (x[i] - xmean) * wt * (x[i] - xmean);
                yy += (y[i] - ymean) * wt * (y[i] - ymean);
                xy += (x[i] - xmean) * wt * (y[i] - ymean);
                xmean += (x[i] - xmean) * wt / sumwt;
                ymean += (y[i] - ymean) * wt / sumwt;
            }
        }
        xy = xy / Math.sqrt(xx * yy);
        return xy;
    }

    private double computeSparsenessMeasure() {
        int n = sortedOriginalMSTLengths.length;
        int n90 = (9 * n) / 10;
        double sparse = Math.min(sortedOriginalMSTLengths[n90] / 1000, 1);
        return sparse;
    }

    private double computeStriationMeasure() {
        double numEdges = 0;
        Iterator<Edge> it = mstEdges.iterator();
        while (it.hasNext()) {
            Edge e = it.next();
            Point n1 = e.p1;
            Point n2 = e.p2;
            if (n1.mstDegree >= 2 && n2.mstDegree >= 2) {
                Iterator<Edge> e1it = getAdjacentMSTEdges(n1, e);
                Iterator<Edge> e2it = getAdjacentMSTEdges(n2, e);
                while (e1it.hasNext()){
                    Edge e1 = e1it.next();
                    if (e1.isEquivalent(e))
                        continue;
                    while (e2it.hasNext()){
                        Edge e2 = e2it.next();
                        if (e2.isEquivalent(e))
                            continue;
                        double cp =  cosineOfAdjacentPlanes(e1, e, e2);
                        if (cp < -0.75){
                            numEdges++;
                        }
                    }
                }
            }
        }
        return numEdges / (double) (mstEdges.size()-3);
    }

    private Iterator<Edge> getAdjacentMSTEdges(Point n, Edge e) {
        Iterator<Edge> nt = n.vE.iterator();
        while (nt.hasNext()) {
            Edge et = nt.next();
            if (et.onMST && !e.isEquivalent(et)) {
                return nt;
            }
        }
        return null;
    }

    private double cosineOfAdjacentPlanes(Edge e1, Edge e, Edge e2) {
        Edge e3 = null;
        if (e.p1.isEqual(e1.p1)){
            e3 = new Edge(e.p2, e1.p2);
        }
        else if (e.p1.isEqual(e1.p2)){
            e3 = new Edge(e.p2, e1.p1);
        }
        else if  (e.p2.isEqual(e1.p1)){
            e3 = new Edge(e.p1, e1.p2);
        }
        else {
            e3 = new Edge(e.p1, e1.p1);
        }

        Edge e4 = null;
        if (e.p1.isEqual(e2.p1)){
            e4 = new Edge(e.p2, e2.p2);
        }
        else if (e.p1.isEqual(e2.p2)){
            e4 = new Edge(e.p2, e2.p1);
```

```
        }
        else if  (e.p2.isEqual(e2.p1)){
            e4 = new Edge(e.p1, e2.p2);
        }
        else {
            e4 = new Edge(e.p1, e2.p1);
        }

        Triangle plane1 = new Triangle(e1, e, e3);
        Triangle plane2 = new Triangle(e, e2, e4);

        double a1 = plane1.a;
        double a2 = plane2.a;
        double b1 = plane1.b;
        double b2 = plane2.b;
        double c1 = plane1.c;
        double c2 = plane2.c;

        double p1 = Math.sqrt(a1 * a1 + b1 * b1 + c1 * c1);
        double p2 = Math.sqrt(a2 * a2 + b2 * b2 + c2 * c2);
        return (a1 * a2 + b1 * b2 + c1 * c2)/(p1 * p2);
    }

    private double computeConvexityMeasure() {
        if (hullVolume == 0) // points in general position
            return 1;
        else {
            double convexity = alphaVolume / hullVolume;
            return convexity;
        }
    }

    private double computeSkinnyMeasure() {
        if (alphaSurfaceArea > 0)
        {
            double c = Math.pow(36*Math.PI, 1.0/6.0);
            return 1 -  c * Math.pow(alphaVolume, 1.0/3.0)/Math.sqrt(alphaSurfaceArea);
        }
        else
            return 1;
    }

    private void computeAlphaVolume() {
        double vol = 0.0;
        Iterator<Tetrahedron> tr = tetrahedra.iterator();
        while (tr.hasNext()) {
            Tetrahedron t = tr.next();
            if (t.onComplex) {
                vol += t.volume;
            }
        }
        alphaVolume = vol;
    }

    private void computeHullVolume() {
```

```java
        double vol = 0.0;
        Iterator<Tetrahedron> tr = tetrahedra.iterator();
        while (tr.hasNext()) {
            Tetrahedron t = tr.next();
            vol += t.volume;
        }
        hullVolume = vol;
    }

    private void computeAlphaSurfaceArea() {
        double sum = 0.0;
        Iterator<Triangle> it = triangles.iterator();
        while (it.hasNext()) {
            Triangle f = it.next();
            if (f.onShape) {
                sum += f.area;
            }
        }
        alphaSurfaceArea = sum;
    }

    private int addPoint(Point p){

        int id = findPoint(p, points);
        if ( id < 0)
        {
            p.pointID = points.size();
            points.add(p);
            return -1;
        }
        else
            return id;
    }

    private void addTetrahedron(Tetrahedron T){
        // add point
        int id = addPoint(T.p1);
        if ( id > 0)
            T.p1 = points.get(id);

        id = addPoint(T.p2);
        if (id > 0)
            T.p2 = points.get(id);

        id = addPoint(T.p3);
        if (id > 0)
            T.p3 = points.get(id);

        id = addPoint(T.p4);
        if (id > 0)
            T.p4 = points.get(id);

        T.p1.vT.add(T);
        T.p2.vT.add(T);
        T.p3.vT.add(T);
```

```java
        T.p4.vT.add(T);

        tetrahedra.add(T);
        int f_ind;
        f_ind = addFacet(T.F1, T);
        if (f_ind >= 0)
            T.F1 = triangles.get(f_ind);
        f_ind = addFacet(T.F2, T);
        if (f_ind >= 0)
            T.F2 = triangles.get(f_ind);
        f_ind = addFacet(T.F3, T);
        if (f_ind >= 0)
            T.F3 = triangles.get(f_ind);
        f_ind = addFacet(T.F4, T);
        if (f_ind >= 0)
            T.F4 = triangles.get(f_ind);
    }

    private int addFacet(Triangle f, Tetrahedron T){

        for (int j = 0; j< triangles.size(); j++){
            if (triangles.get(j).isEquivalent(f)){
                triangles.get(j).T2 = T;
                return j;
            }
        }
        f.p1.vF.add(f);
        f.p2.vF.add(f);
        f.p3.vF.add(f);
        f.T1 = T;
        triangles.add(f);

        int e_ind;
        e_ind = addEdge(f.e1, f);
        if (e_ind >= 0)
            f.e1 = edges.get(e_ind);

        e_ind = addEdge(f.e2, f);
        if (e_ind >= 0)
            f.e2 = edges.get(e_ind);

        e_ind = addEdge(f.e3, f);
        if (e_ind >= 0)
            f.e3 = edges.get(e_ind);

        return -1;
    }

    private int addEdge(Edge e, Triangle f) {
        // update the neighbors of Point(end points of e)
        for (int j = 0; j< edges.size(); j++){
            if (edges.get(j).isEquivalent(e)){
                {
                    edges.get(j).inT.add(f);
                    return j;
                }
            }
        }
        // update master list of edges
        int id1 = findPoint(e.p1, points);
        int id2 = findPoint(e.p2, points);
        points.get(id1).vE.add(e);
        points.get(id2).vE.add(e);
        e.inT.add(f);
        edges.add(e);

        return -1;
    }

    private int findPoint(Point p, List<Point> L) {
        for (int i=0; i< L.size(); i++){
            if (L.get(i).isEqual(p))
                return i;
        }
        return -1;
    }

}
```

```java
package scagnostics3D;

import java.io.File;
import java.util.*;

public class Main {
    static String osname = System.getProperty("os.name");

    public static void main(String[] argv) {

        String fname = "/tmp/scagnostics3D/iris.txt";

        if (osname.equalsIgnoreCase("Windows XP")){
            fname = "C:\\Test\\iris.txt";
        }

        double[][] points = getData(new File(fname));
        double[][] scagnostics = computeScagnostics(points);
        // test
        String[] sl = Scagnostics3D.getScagnosticsLabels();
        for (int l=0;l<scagnostics.length;l++){
            for (int kk=0; kk<9; kk++)
                System.out.println(sl[kk] + ": " + scagnostics[l][kk]);
            System.out.println("--------------------- ");
        }
        boolean[] outliers = Scagnostics3D.computeScagnosticsOutliers(scagnostics);
        boolean[] exemplars = Scagnostics3D.computeScagnosticsExemplars(scagnostics);
        computeTests(scagnostics, outliers, exemplars);
    }

    private static double[][] getData(File fname) {
        java.io.BufferedReader fin;
        try {
            fin = new java.io.BufferedReader(new java.io.FileReader(fname));
        } catch (java.io.FileNotFoundException fe) {
            javax.swing.JOptionPane.showMessageDialog(null, "File not found!", "Alert",
                    javax.swing.JOptionPane.ERROR_MESSAGE);
            return null;
        }
        try {
            String record;
            record = fin.readLine();
            //Get the scagnosticsLabels
            record = replaceSeparatorsWithBlanks(record);
            StringTokenizer st = new StringTokenizer(record, " ");
            int col = 0;
            int numVars = st.countTokens();
            String[] variableLabels = new String[numVars];

            while (st.hasMoreTokens()) {
                variableLabels[col] = st.nextToken();
                col++;
            }
            //Count the number of rows
            int numRows = 0;
```

```java
            record = fin.readLine();
            while (record != null) {
                record = fin.readLine();
                numRows++;
            }
            fin.close();
            System.out.println("Number of rows, cols " + numRows + " " + numVars);

            //Read in the data
            fin = new java.io.BufferedReader(new java.io.FileReader(fname));
            double[][] data = new double[numVars][numRows];
            record = fin.readLine();     //ignore line with scagnosticsLabels
            record = fin.readLine();
            int j = 0;
            while (record != null) {
                record = replaceSeparatorsWithBlanks(record);
                st = new StringTokenizer(record, " ");
                for (int i = 0; i < numVars; i++) {
                    try {
                        String tmp = st.nextToken();
                        data[i][j] = Double.parseDouble(tmp);
                    } catch (Exception ie) {
                        javax.swing.JOptionPane.showMessageDialog(null,
                                "Error reading from the file", "Alert",
                                javax.swing.JOptionPane.ERROR_MESSAGE);
                        return null;
                    }
                }
                record = fin.readLine();
                j++;
            }
            fin.close();

            return data;
        } catch (java.io.IOException ie) {
            javax.swing.JOptionPane.showMessageDialog(null,
                    "Error reading from the file", "Alert",
                    javax.swing.JOptionPane.ERROR_MESSAGE);
            return null;
        }
    }

    private static String replaceSeparatorsWithBlanks(String record) {
        record = replaceAll(record, ",,", ",~,");
        record = replaceAll(record, "\t\t", "\t~\t");
        record = replaceAll(record, ",", " ");
        record = replaceAll(record, "\t", " ");
        return record;
    }

    private static String replaceAll(String source, String toReplace, String replacement) {
        int idx = source.lastIndexOf(toReplace);
        if (idx != -1) {
            StringBuffer sb = new StringBuffer(source);
            sb.replace(idx, idx + toReplace.length(), replacement);
            while ((idx = source.lastIndexOf(toReplace, idx - 1)) != -1) {
                sb.replace(idx, idx + toReplace.length(), replacement);
            }
            source = sb.toString();
        }
        return source;
    }

    private static double[][] computeScagnostics(double[][] points ) {
        normalizePoints(points);
        int nDim = points.length;
        int numCells = nDim * (nDim - 1)* (nDim - 2) / 6;
        double[][] scagnostics = new double[numCells][Scagnostics3D.getNumScagnostics()];
        int l = 0;
        for (int i = 2; i < nDim; i++) {
            for (int j = 1; j < i; j++) {
                for (int k = 0; k < j; k++) {
                Scagnostics3D s = new Scagnostics3D(points[k], points[j], points[i]);
                scagnostics[l] = s.compute();
                l++;
                }
            }
        }
        return scagnostics;
    }

    private static void normalizePoints(double[][] points) {
        double[] min = new double[points.length];
        double[] max = new double[points.length];
        for (int i = 0; i < points.length; i++) {
            min[i] = Double.MAX_VALUE;
            max[i] = Double.MIN_VALUE;
        }
        for (int i = 0; i < points.length; i++) {
            for (int j = 0; j < points[0].length; j++) {
                if (min[i] > points[i][j]) min[i] = points[i][j];
                if (max[i] < points[i][j]) max[i] = points[i][j];
            }
        }
        for (int i = 0; i < points.length; i++) {
            for (int j = 0; j < points[0].length; j++) {
                points[i][j] = (points[i][j] - min[i]) / (max[i] - min[i]);
            }
        }
    }

    private static void computeTests(double[][] scagnostics, boolean[] outliers, boolean[]
exemplars) {
        double[][] test = getData(new File("scagnostics.txt"));

        for (int i = 0; i < test.length; i++) {
            for (int j = 0; j < test[0].length; j++) {
                if (scagnostics[j][i] != test[i][j]) System.out.println("error " + i + " " + j);
            }
        }
```

```
        for (int i = 0; i < outliers.length; i++) {
            if (i == 1 || i == 2 || i == 34 || i == 90) {
                if (!outliers[i]) System.out.println("error " + i);
            } else {
                if (outliers[i]) System.out.println("error " + i);
            }
        }

        for (int i = 0; i < exemplars.length; i++) {
            if (i == 13 || i == 95 || i == 118) {
                if (!exemplars[i]) System.out.println("error " + i);
            } else {
                if (exemplars[i]) System.out.println("error " + i);
            }
        }
    }
}
```

# References

[1] C. Bradford Barber, David P. Dobkin, and Hannu Huhdanpaa. The quickhull algorithm for convex hulls. *ACM Trans. Math. Softw.*, 22(4):469–483, 1996. 16

[2] E. Bruzzone and L. De Floriani. Two data structures for building tetrahedralizations. *Vis. Comput.*, 6(5):266–283, 1990. 19

[3] Andreas Buja, John Alan McDonald, John Michalak, and Werner Stuetzle. Interactive data visualization using focusing and linking. In *VIS '91: Proceedings of the 2nd conference on Visualization '91*, pages 156–163, Los Alamitos, CA, USA, 1991. IEEE Computer Society Press.

[4] William S. Cleveland. *The Collected Works of John W. Tukey: Graphics 1965-1985, Volume V.* Chapman and Hall/CRC, Boca Raton, FL, 1988. 3

[5] Dianne Cook, Andreas Buja, Javier Cabrera, and Catherine Hurley. Grand tour and projection pursuit. *Journal of Computational and Graphical Statistics*, 4:155–172, 1995. 2

[6] B. Das and M. C. Loui. Reconstructing a minimum spanning tree after deletion of any node, 2001. 28

[7] Herbert Edelsbrunner and Ernst P. Mücke. Three-dimensional alpha shapes. *ACM Trans. Graph.*, 13(1):43–72, 1994. 5

[8] D A Field. Implementing watson's algorithm in three dimensions. In *SCG '86: Proceedings of the second annual symposium on Computational geometry*, pages 246–259, New York, NY, USA, 1986. ACM. 19

[9] Steven Fortune. Voronoi diagrams and delaunay triangulations. pages 377–388, 1997. 12

[10] Jerome H. Friedman and Werner Stuetzle. John W. Tukey's work on interactive graphics. *Ann. Stat.*, 30(6):1629–1639, 2002. 3

[11] Francisco Gomez, Suneeta Ramaswami, and Godfried T. Toussaint. On removing non-degeneracy assumptions in computational geometry. In *CIAC '97: Proceedings of the Third Italian Conference on Algorithms and Complexity*, pages 86–99, London, UK, 1997. Springer-Verlag. 26

[12] J. Hartigan and Surya Mohanty. The runt test for multimodality. *Journal of Classification*, 9(1):63–70, January 1992.

[13] Martin Henk, Jürgen Richter-Gebert, and Günter M. Ziegler. Basic properties of convex polytopes. pages 243–270, 1997. 13

[14] C.B Hurley and R. W. Oldford. Graphs as navigational infrastructure for high dimensional data spaces. 2008.

[15] P.A. Tukey J. W. Tukey. Computer graphics and exploratary data analysis: an introduction. In *CIAC '97: Proceedings of the Sixth Annual Conference and Exposition: Computer Graphics 85*, Farifax, VA, United States, 1985. National Computer Graphics Association.

[16] Jr. Joseph B. Kruskal. On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical Society*, 7(1):48–50, 1956.

[17] Graham Wills Leland Wilkinson. Scagnostics distributions. *Journal of Computational and Graphical Statistics*, 17(2):473–491, 2008. 6, 8, 14, 23

[18] Jame F. Peters. *UNIX Programming Methods and Tools*. Harcourt Brace Jovanovich Publishers, San Diego, 1988.

[19] Franco P. Preparata. *Computational geometry: an introduction*. Springer-Verlag, London, 1985.

[20] Franco P. Preparata and Michael I. Shamos. *Computational geometry: an introduction*. Springer-Verlag New York, Inc., New York, NY, USA, second edition, 1985.

[21] R. C. Prim. Shortest connection networks and some generalizations. *Bell System Technical Journal*, 36:1389C1401, 1957.

[22] Steven S. Skiena. *The Algorithm Design Manual*. Springer, London, second edition, 2008. 6

[23] Werner Stuetzle. Estimating the cluster tree of a density by analyzing the minimal spanning tree of a sample. *Journal of Classification*, 20(1):025–047, 2003. 7

[24] Leland Wilkinson. *The Grammar of Graphics.* Springer, London, second edition, 2005.

[25] Leland Wilkinson and Anushka Anand. High-dimensional visual analytics: Interactive exploration guided by pairwise views of point distributions. *IEEE Transactions on Visualization and Computer Graphics*, 12(6):1363–1372, 2006. Member-Grossman, Robert.

[26] Leland Wilkinson, Anushka Anand, and Robert Grossman. Graph-theoretic scagnostics. In *INFOVIS '05: Proceedings of the Proceedings of the 2005 IEEE Symposium on Information Visualization*, page 21, Washington, DC, USA, 2005. IEEE Computer Society. 3, 4, 5, 6, 8, 11

[27] Leland Wilkinson, Anushka Anand, and Robert Grossman. High-dimensional visual analytics: Interactive exploration guided by pairwise views of point distributions. *IEEE Transactions on Visualization and Computer Graphics*, 12(6):1363–1372, 2006.