Elastic maps

Contents

[1 Introduction 2](#_Toc7196292)

[2 Map geometry 2](#_Toc7196293)

[3 Projection data points onto map 4](#_Toc7196294)

[3.1 Projection of data points to node 4](#_Toc7196295)

[3.2 Projection of data onto nearest edge 5](#_Toc7196296)

[3.3 Projection of data onto nearest face 6](#_Toc7196297)

[4 Fraction of variance unexplained 8](#_Toc7196298)

[5 Map extrapolation 9](#_Toc7196299)

[5.1 Extension for OneDMap 9](#_Toc7196300)

[5.2 Extension for rect2DMap 10](#_Toc7196301)

[5.3 Extension for tri2DMap 12](#_Toc7196302)

[6 EM (Elastic map) 14](#_Toc7196303)

[6.1 Elastic energy 14](#_Toc7196304)

[6.2 Weighted version 17](#_Toc7196305)

[6.3 and choice 17](#_Toc7196306)

[6.4 PQSQR for data 19](#_Toc7196307)

[6.5 Weighted PQSQ version 20](#_Toc7196308)

[7 Maps colouring 20](#_Toc7196309)

[7.1 Grid of colouring 21](#_Toc7196310)

[7.2 Function defined in nodes and in arbitrary points 21](#_Toc7196311)

[7.3 Function defined in data points 21](#_Toc7196312)

[8 Tests 22](#_Toc7196313)

[8.1 Two arcs 22](#_Toc7196314)

[8.2 Two arcs with x and y shift 23](#_Toc7196315)

[8.3 Fragment of sphere 26](#_Toc7196316)

[8.4 Two fragments of sphere 26](#_Toc7196317)

[9 Appendix A. Formulas derivation 28](#_Toc7196318)

[10 Appendix B. Examples of map descriptions 29](#_Toc7196319)

[10.1 OneDMap 29](#_Toc7196320)

[10.2 rect2DMap 30](#_Toc7196321)

[10.3 tri2DMap 32](#_Toc7196322)

[11 References 33](#_Toc7196323)

# Introduction

Elastic map is special techniques for the data visualisation and search of nonlinear low dimensional manifolds (dimensionality reduction). Previous version of Elastic maps was developed in 2000 and was widely used in many areas [1 – 5 ].

Presented package ElMap is developed to form nonlinear manifolds by Elastic Map (EM). This document contains theoretical description, user guide and technical description.

Examples of package usage are presented in the three auxiliary documents

* Test of bladder cancer.docx
* Test of breast cancer.docx
* Test of healthy tissues.docx

Additional information for data imputation is presented in <https://github.com/Mirkes/DataImputation/blob/master/doc/Data%20imputation%20for%20visualisation.docx>.

# Map geometry

Package developed for 1D and 2D maps. It is possible to use maps with more than two dimensions but this functionality is not basic. This class is basic class of package because EM are the techniques to fit map for data only.

There is only one type of 1D maps: piecewise linear (Figure 1a). It is possible to construct many different 2D maps. ElMap includes two standard 2D maps: rectangular (Figure 1b) and triangular (Figure 1c). Package also contains possibility to add user implemented map with arbitrary structure.

a

b

c

Figure 1. Standard map geometry: a) 1D map, b) rectangular 2D map and c) triangular 2D map. Red segments corresponds to example of ribs.

There are three standard descendants of ***MapGeometry*** class:

1. OneDMap is one dimensional map (Figure 1a);
2. rect2DMap is two dimensional map with rectangular grid (Figure 1b);
3. tri2DMap is two dimensional map with triangular grid (Figure 1c).

Next fragment of this section contains technical description of MapGeometry class.

Each map must be descendant of ***MapGeometry*** class and must include following property:

***Dimension*** is number of internal coordinates: internal map dimension.

***Internal coordinates*** is the array of coordinates for each node in the map defined coordinates. For example, for 1D map there is only one coordinate for each node: leftmost node has coordinate one, the next node has coordinate two and so on. For rectangular 2D map left bottom node has coordinates , the next node in the bottom line has coordinates , the node in the leftmost column and in the first line above bottom one has coordinates , the node in the intersection of th row from the bottom and th column from the left has coordinates . For triangular 2D map the nodes in the bottom row have coordinates ; nodes in the line above the top have coordinates ; nodes in the next rows have coordinates .

***Mapped coordinates*** is the set of coordinates of nodes in the data space. These coordinates are initially defined by initializing procedure and then adjusted by the map fitting. Procedure of map fitting is external with respect to the map and can be provided by EM fitting process.

***Links*** is the set of map edges and completely defined by the map geometry. Each edge is the fragment of straight line which connect the nearest nodes in the Figure 1: 4 edges in the subfigure a, 112 edges in the subfigure b and 111 edges in the subfigure c.

***Ribs*** is set of three adjacent nodes which are belonged to one straight line in the internal coordinates. For one dimension map rib is set of two adjacent edges. For rectangular two dimension map it is pair of horizontal adjacent edges or vertical adjacent edges. For triangular two dimension map there are three directions of ribs. Examples of ribs are presented in Figure 1 by red segments.

***Disp*** is dispersion measure for PQSQ approach [6]. To calculate disp we have to calculate distance from each data point to the nearest initial node and then take maximum of these distances.

***Preproc*** is true if data were preprocessed.

***Means*** contains mean of data otherwise.

***PCs*** contains set of PCs otherwise.

Each map must provide following methods.

***Constructor*** is method to create map. Constructor creates arrays of nodes, edges and ribs and defines internal coordinates of nodes. Name of constructor is map dependent. Constructor’s input arguments are map dependent too.

***Init*** is the method of map initialization. This method perform required data preprocessing (see methods “preprocessData” and “preprocessDataInit”). This method defines an initial mapped coordinates. In accordance of results of paper [7] three methods have to be implemented by each map: random initialization, random selection and principal component initialization. Input arguments of this method are set of data points and type of initialization. Default version of this method is implemented in the ***MapGeometry*** class.

***Project*** is the method to calculate projection of data point (points) into map. There are types of projection for dimensional map: 0 means projection into nearest node of map, 1 means projection onto nearest edge of map, 2 means projection onto nearest face of map. Projection can be calculated in the internal or mapped coordinates. There are three input arguments for this method: set of point to project, type of projection (integer number) and coordinates space for projection: ‘internal’ or ‘mapped’. ***MapGeometry*** class implements this method for the types 0, 1 and 2. Main functionality of this method is implemented in the function ***ProjectPrim.***

***getFaces*** is optional method. It must be implemented by ***MapGeometry*** descendant for projection onto faces. Each face is the set of three nodes. This method cannot be implemented for one dimensional map.

***extendPrim*** is primitive method to extrapolate map to reduce border effect. This method is used by ***extend*** method of ***MapGeometry*** class and is map specific.

***getBorder*** is method to form list of border nodes. This function is map specific.

Following methods is provided by the ***MapGeometry*** class:

***getDimension*** is method to get map dimension.

***getInternalCoordinates*** is method to access the internal coordinates of map.

***getMappedCoordinates*** is method to access the mapped coordinates of map.

***getLinks*** is method to access edges of map.

***getRibs*** is method to access ribs of map.

***preprocessDataInit*** is the method to preprocess data. User can specify number of the first principal components to use. Otherwise, if number of data points is less than dimension of data space, then the first number of points minus one principal components are used to preprocess data. Principal components are calculated ones in this method.

***preprocessData*** performs data preprocessing if ***preprocessDataInit*** method created nonempty preprocessing.

***deprocessData*** performs operation inverse with respect to ***preprocessData.***

***getDisp*** is method to access ***disp*** field of map.

***associate*** is method to find nearest node for each data point. This method returns the number of nearest node and squared distance to it.

***borderCases*** is method to calculate fraction of data points which are projected into border nodes (see “Map extrapolation”).

***extend*** is method to extrapolate maps without training. This method used ***extendPrim*** method of descendant.

***FVU*** is method to calculate Fraction of variance unexplained.

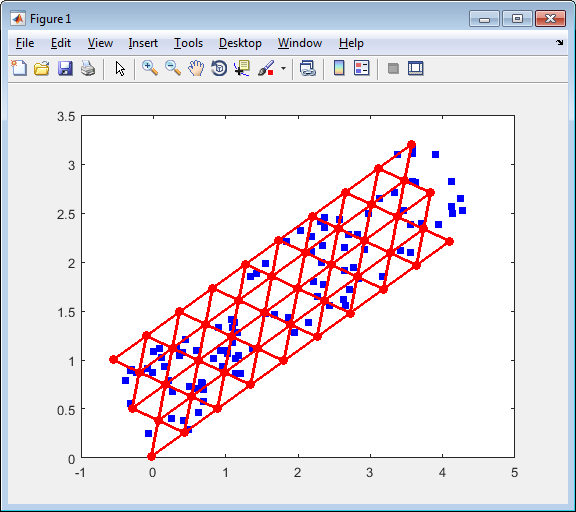
***putMapped*** is method to put fitted mapped coordinates to the map.

# Projection data points onto map

For each map we can consider several types of projections: projection into nearest node, projection onto nearest edge or face. In this section all formulas which are necessary for projection calculation are derived.

## Projection of data points to node

It is the simplest type of projection. Method calculates distances from each data points to each map node and selects the node with least distance for each data point. Examples of visualization of this type of projection are presented in Figure 2. Number of points which are projected to the same node is presented as size of circle.

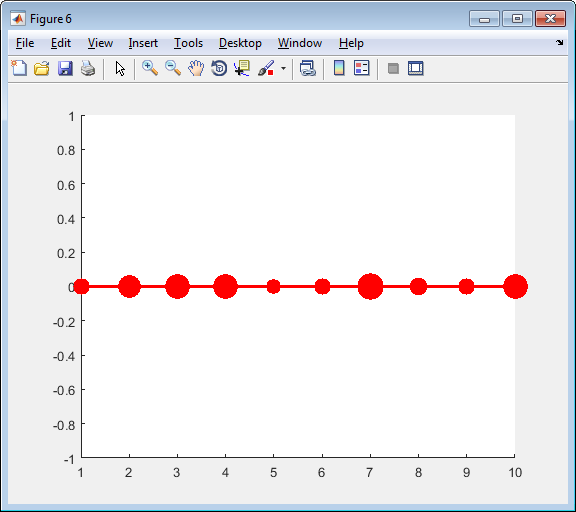
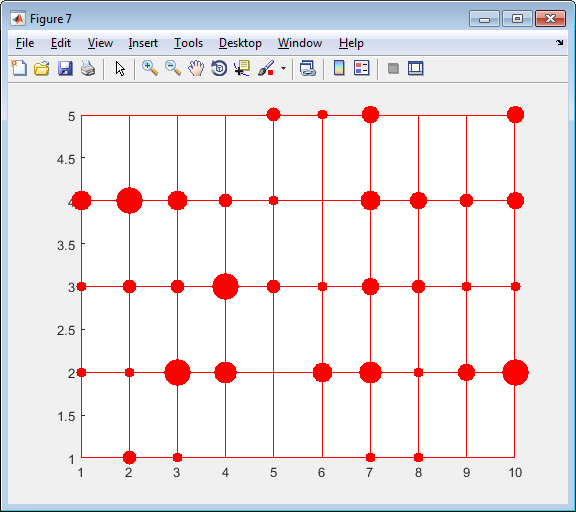
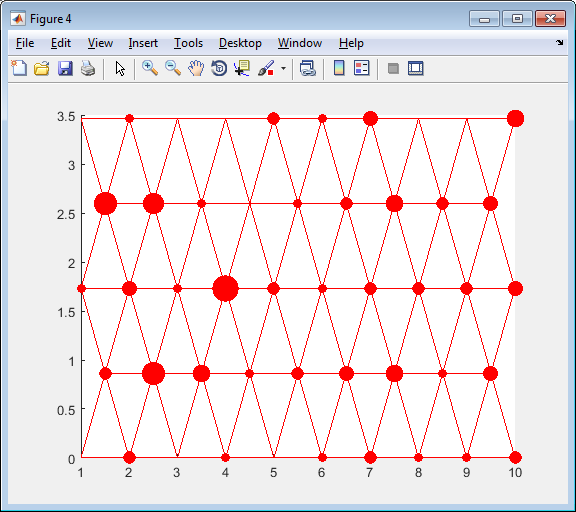
  

Figure 2. Examples of data points projection to the nearest node: top row contains graphs in the original space and bottom row contains corresponding graphs in the internal coordinates; left column presents the one dimensional map, central column presents rectangular 2D map and right column presents the triangular 2D map.

## Projection of data onto nearest edge

It is important to stress that globally nearest edge sometimes does not contain nearest node (see Figure 3). It is rare case. We consider the projection of point to globally nearest edge. Let us consider data point and edge defined by two nodes . Mapped coordinates of these nodes we denote . See Figure 4 for calculation illustration.

Figure 3

Projection to line which contain the edge can be written as convex combination of the nodes which define this edge:

|  |  |
| --- | --- |
|  | (1) |

Interior of the edge is defined by inequality . Let us find projection of arbitrary point :

Figure 4

a)

b)

c)

In the projection point this distance is minimal. To find parameter we have to differentiate squared distance with respect to and equal derivative to zero:

|  |  |
| --- | --- |
|  | (2) |

where is the scalar or dot product of two vectors.

We are interested in the internal points only. Case where corresponds to Figure 4b and case where corresponds to Figure 4c. It means that we have to adjust calculated parameter:

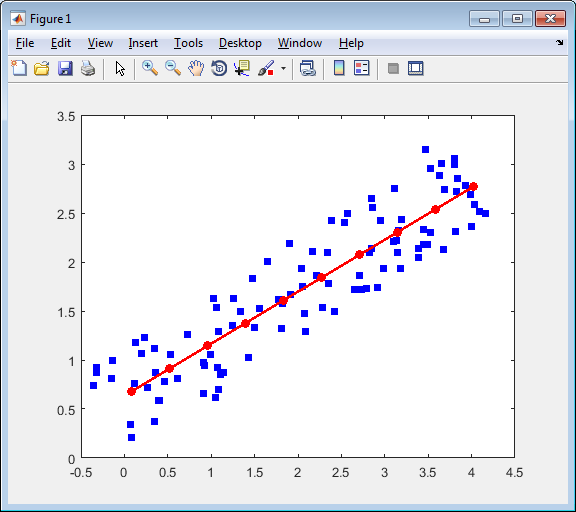
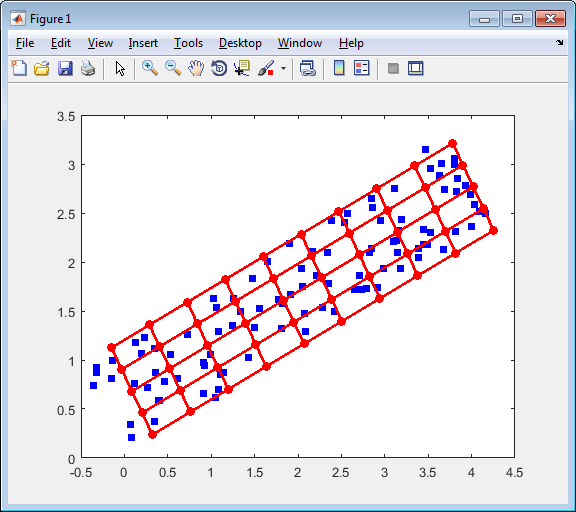
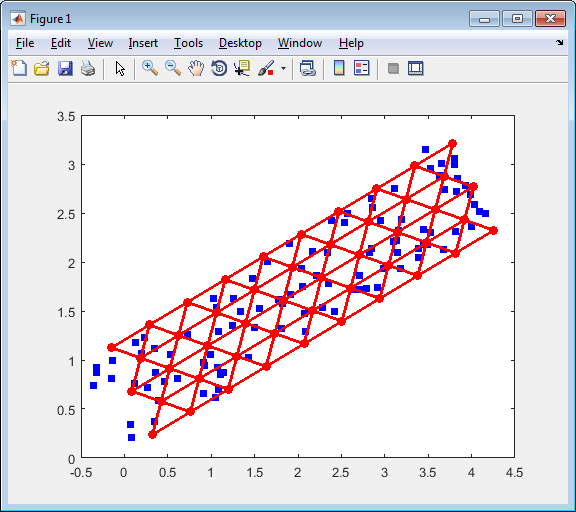
|  |  |
| --- | --- |
|  | (3) |

To calculate distance we can use formula

|  |  |
| --- | --- |
|  | (4) |

The general algorithm is:

1. Calculate parameters of projections by formula (2).
2. Calculate adjusted parameter by formula (3)
3. Calculate distances by formula (4). Select the nearest edge and calculate coordinates of projection onto nearest edge by formula (1).

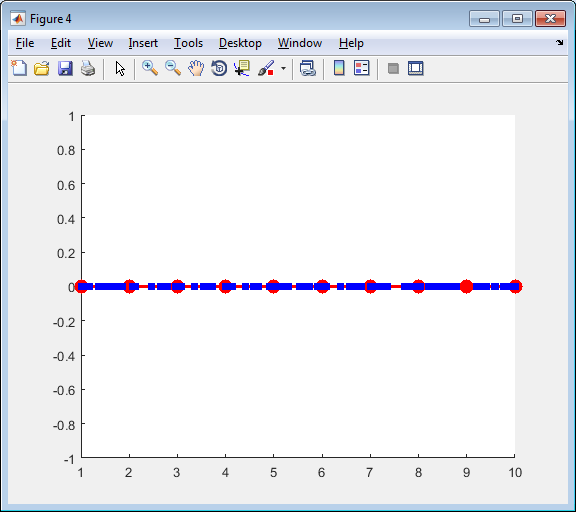
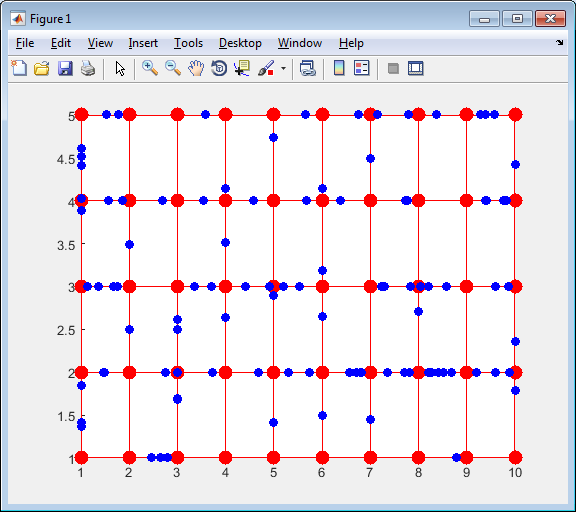
  

Figure 5. Examples of data point projection to the nearest node: top row contains graphs in the original space and bottom row contains corresponding graphs in the internal coordinates; left column presents the one dimensional map, central column presents rectangular 2D map and right column presents the triangular 2D map.

## Projection of data onto nearest face

Projection onto nearest face is not defined for all map geometries. To use this option used map geometry must implement method getFaces. Two of three standard map geometries are implemented it. For tri2DMap each triangle is a face. For rect2DMap faces are presented in the Figure 6.

Figure 6

Let us implement barycentric coordinates (see Figure 7). We need to find the point in the triangle which is nearest to the point . Barycentric coordinates of point are :

Figure 7

For barycentric coordinates there is restriction

Squared distance between points and is:

To find the required values of we need to find the minimum of squared distance. To do it differentiate squared distance with respect to and :

We can rewrite these equations as

By using the dot product we can rewrite the last equations as

We apply Cramer formula to solve this system of linear equations:

Barycentric coordinates system allows us to recognise cases when projection point is located out of face but cannot help us to improve this situation. It means that we have to calculate projection onto face plane and onto each edge by usage the formula (2) for each edge. Let us denote

In this case we can write the coefficients for projections for each edge as

|  |  |
| --- | --- |
|  | (5) |
|  | (6) |
|  | (7) |

Coefficients of projection onto face can be rewritten as

|  |  |
| --- | --- |
|  | (8) |
|  | (9) |

After calculation the values (5) – (9) we calculate normalized values

|  |  |
| --- | --- |
|  | (10) |

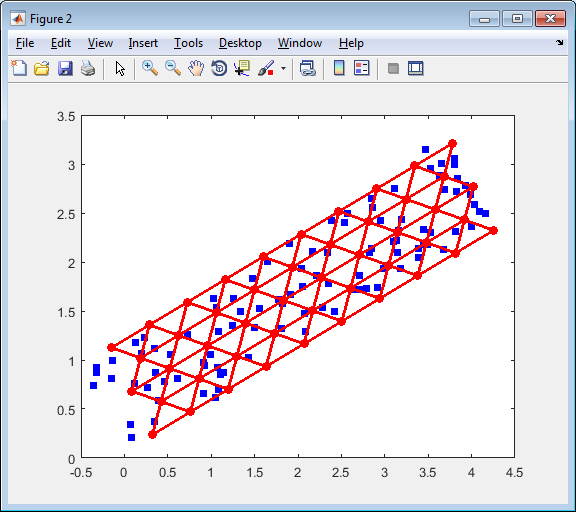
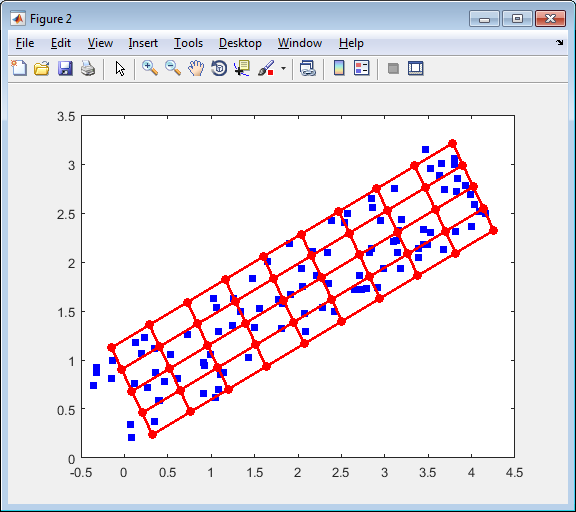
and

|  |  |
| --- | --- |
|  | (11) |

Now we can calculate the distance:

|  |  |
| --- | --- |
|  | (12) |

Examples of data points’ projection onto nearest face are presented in Figure 8.



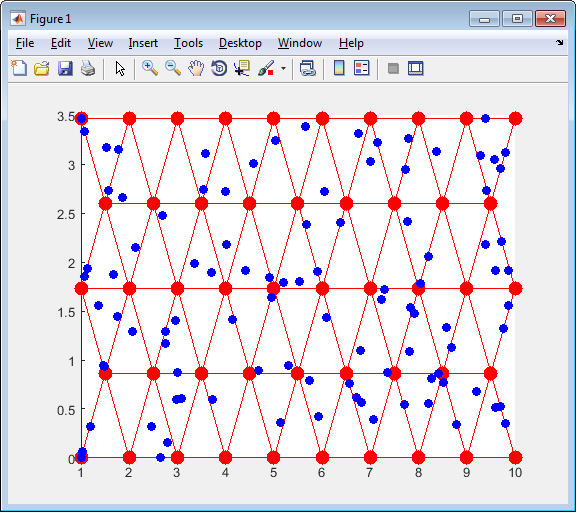
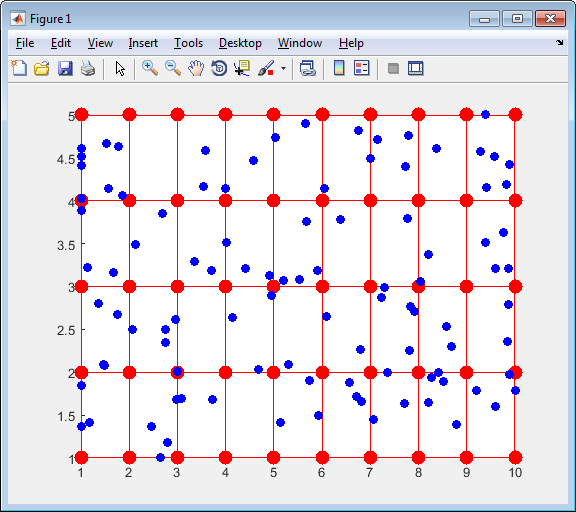


Figure 8. Examples of data points’ projection to the nearest face: top row contains graphs in the original space and bottom row contains corresponding graphs in the internal coordinates; left column presents the rectangular 2D map and right column presents the triangular 2D map.

# Fraction of variance unexplained

Fraction of Variance Unexplained (FVU) [8] is usually considered as measure of goodness of fit for specified statistical model. The less FVU means the better model. In this package FVU is unexplained variance divided by ‘zero model’ variance. This measure is closely related with coefficient of determination [8]. This measure usually applied to models with one dependent variable. In our case we consider multidimensional data and variance is not a scalar value. For such cases the sum of variances is widely used. Let us consider the dimensional data space. If we consider each coordinate as random variable then for th coordinate we have

|  |  |
| --- | --- |
|  | (13) |

where is number of data points. The full variance is

|  |  |
| --- | --- |
|  | (14) |

We can see that (14) defines variance as average squared distance between data points and mean point which is considered as zero model. We generalize (14) for arbitrary model as

|  |  |
| --- | --- |
|  | (15) |

where is the approximation of point by the model . FVU can be calculated as

|  |  |
| --- | --- |
|  | (16) |

There are several values of FVU can be considered for EM and SOM: this value is depends from the type of used projection (see “Projection data points onto map”). It is preferable to consider projection onto face for two dimensional maps and onto edges for one dimensional map. However, possibility to project onto face is optional and part of map geometries does not provide it. As a result we consider projection onto edges as default method for FVU calculation.

Important notion: it is necessary to stress that FVU is not the function which is minimized during the map fitness for EM.

# Map extrapolation

Most mapping techniques, includes Elastic maps, have many border points (see, for example, Figure 24 and Figure 25). This effect is result of map fitting procedures. The simplest way to resolve this problem is to add one or more map fragments outside the map. Examples of one layer extension for standard mars are presented in Figure 9.

a

b

c

Figure 9. Extension of standard maps: black is old map, red shows added edges, and blue shows new border

## Extension for OneDMap

In this map we add two edges: at the beginning and at the end. Both edges are replicas of the first and last edges, correspondingly.

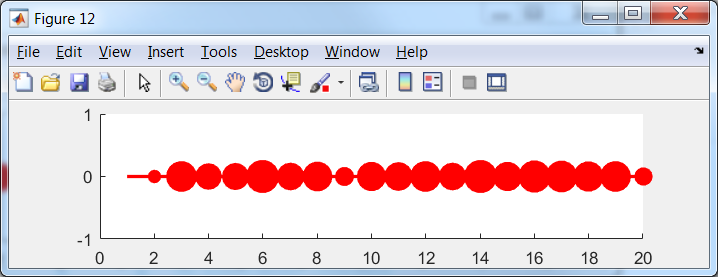
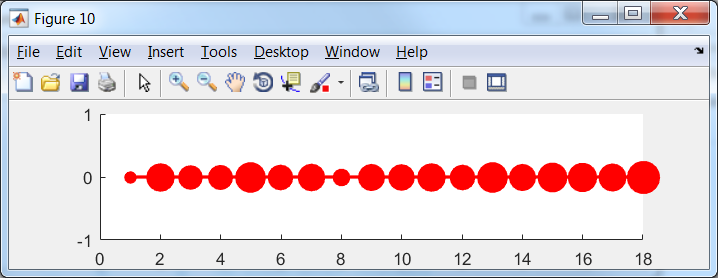
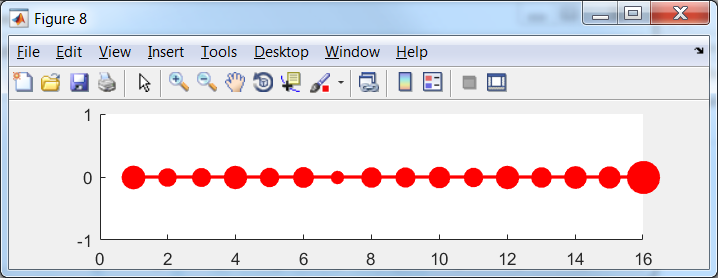
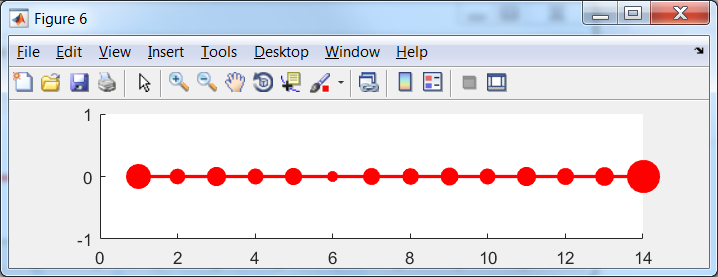
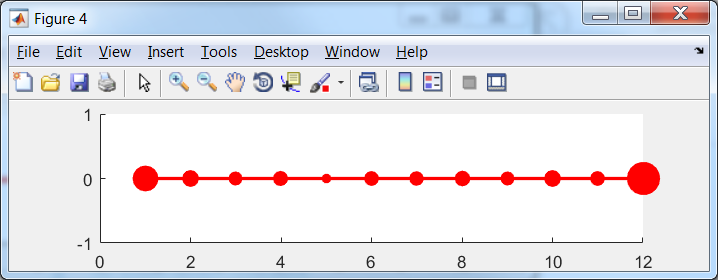
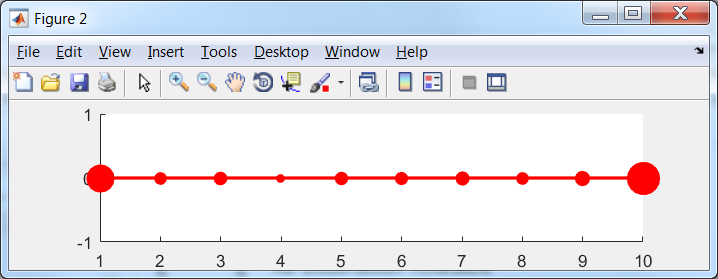


Figure 10. Extension of OneDMap(10) for EM(map, data, 'stretch', 0.01, 'bend', 0.1, 'potential', @LLog, 'Number\_of\_intervals', 5); left is original map and then each next (left to right and top to down) map have one more ribbon left and right

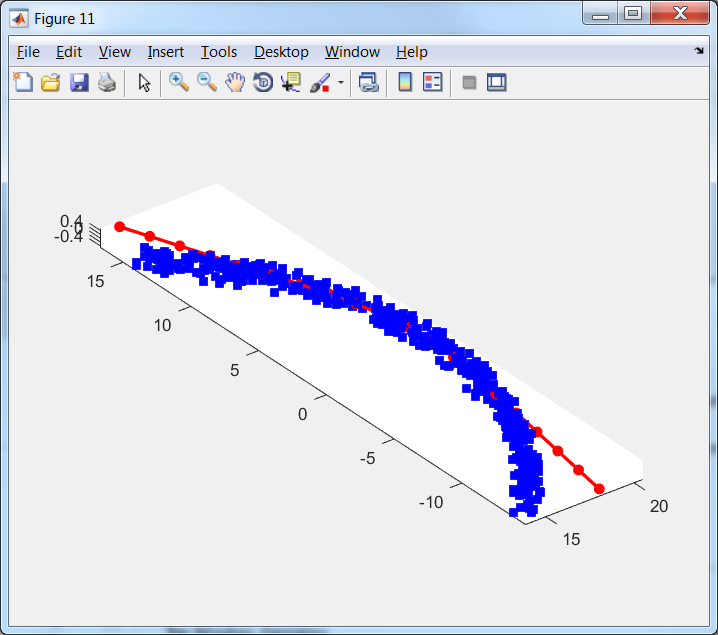
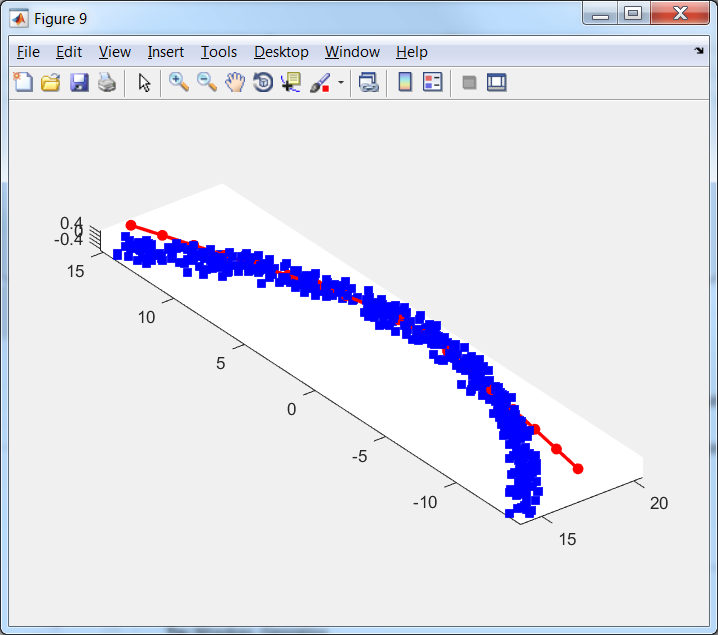
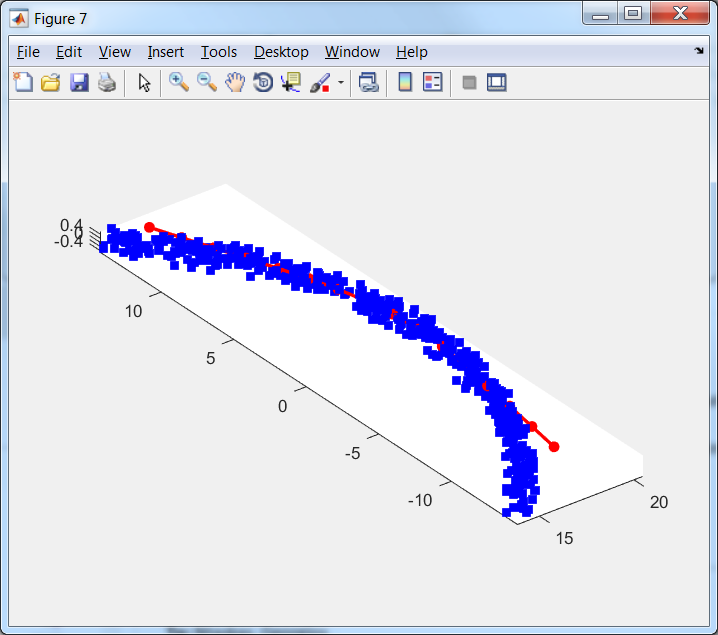
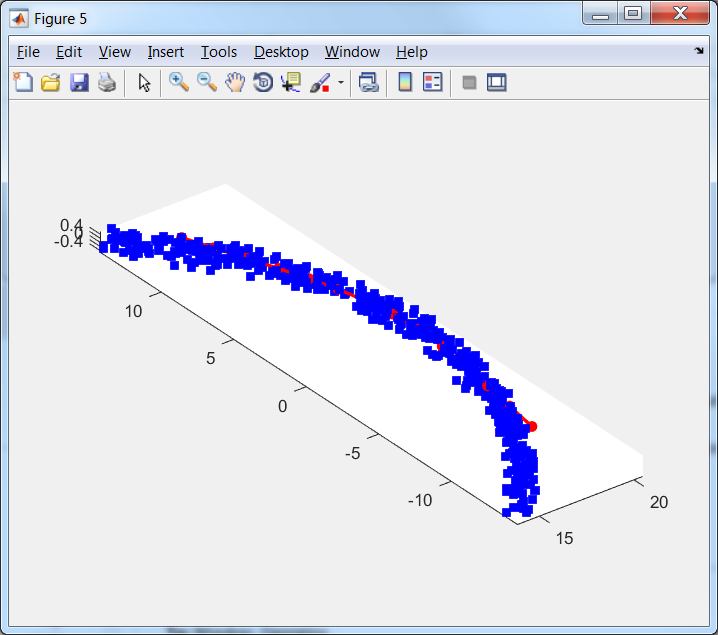
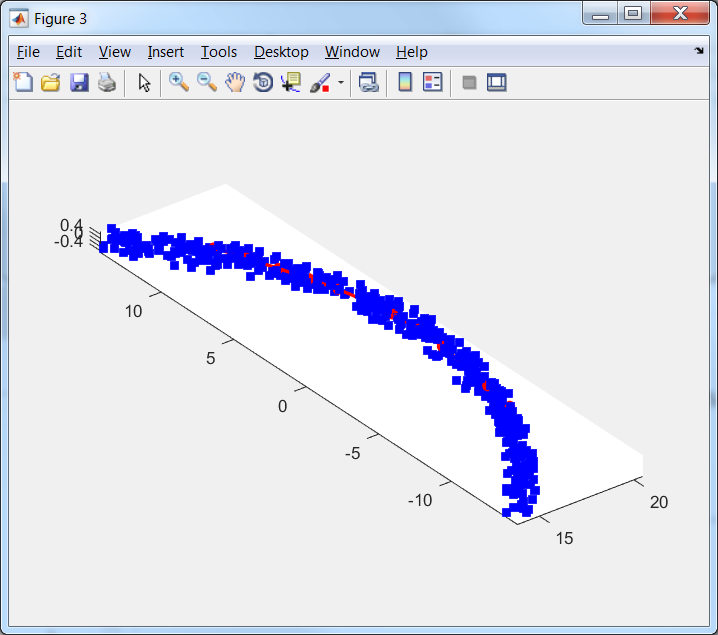
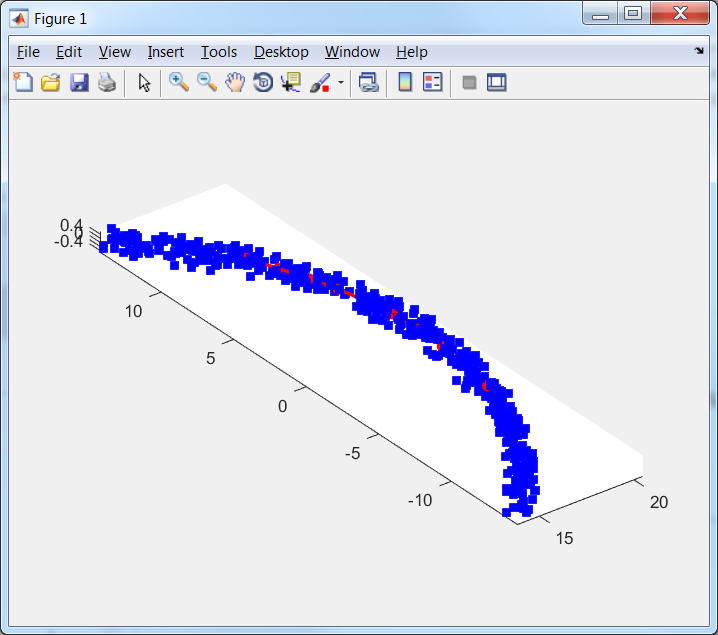
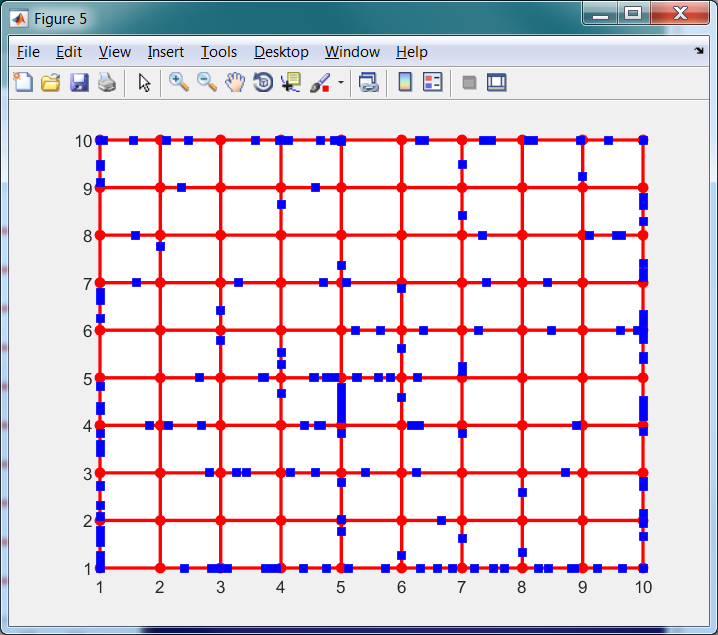
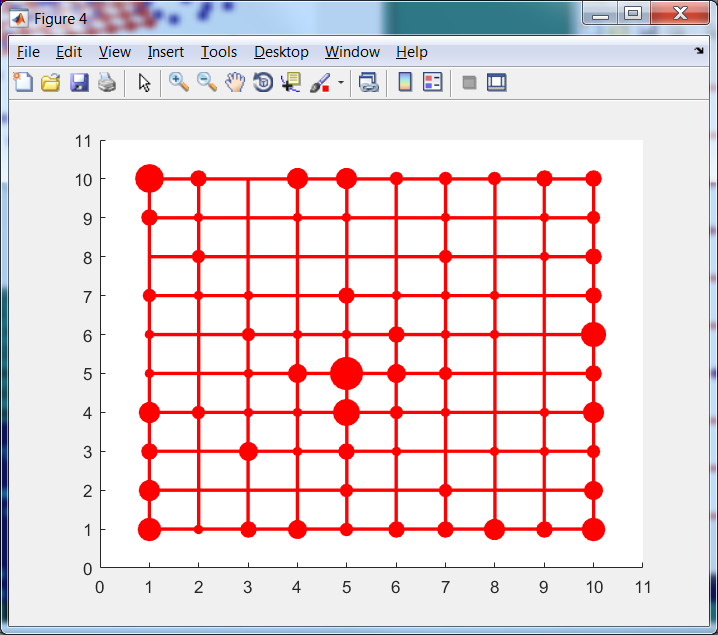
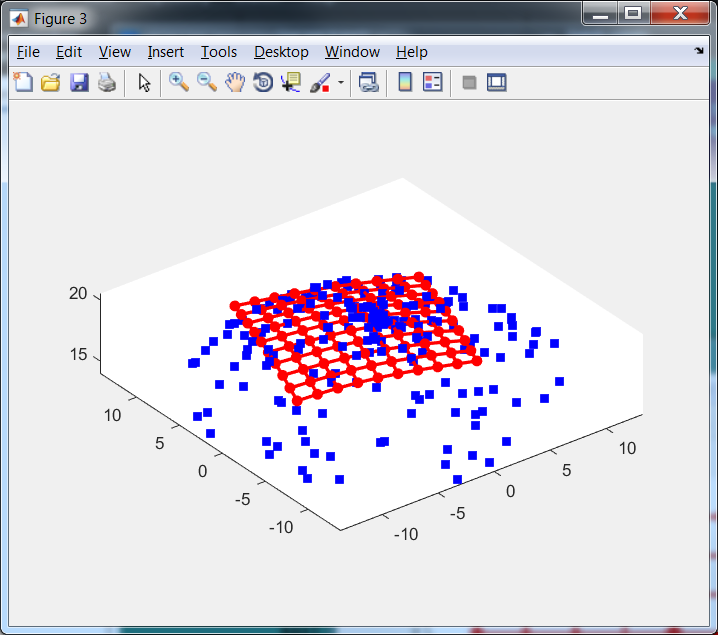


Figure 11. Extension of OneDMap for EM(map, data, 'stretch', 0.01, 'bend', 0.1, 'potential', @LLog, 'Number\_of\_intervals', 5); left is original map and then each next (left to right and top to down) map have one more ribbon left and right

As we can see in the original map fraction of points at the border case is considerable but then decrease. Moreover the last extension can be considered as unnecessary.

## Extension for rect2DMap

We train map by Elastic map with following arguments: EM(map, data, 'stretch', 0.001, 'bend', 0.1, 'potential', @LLog, 'Number\_of\_intervals', 5). This method produce not very good map but this map is appropriate to use extension.



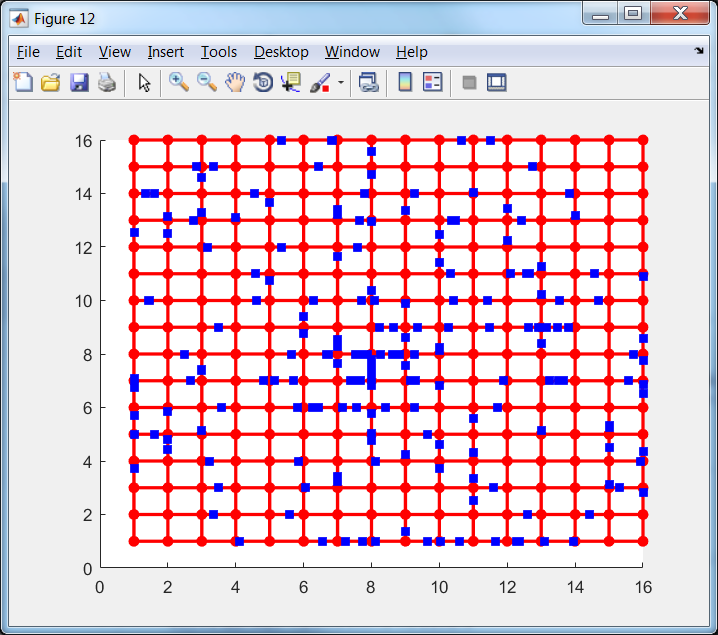
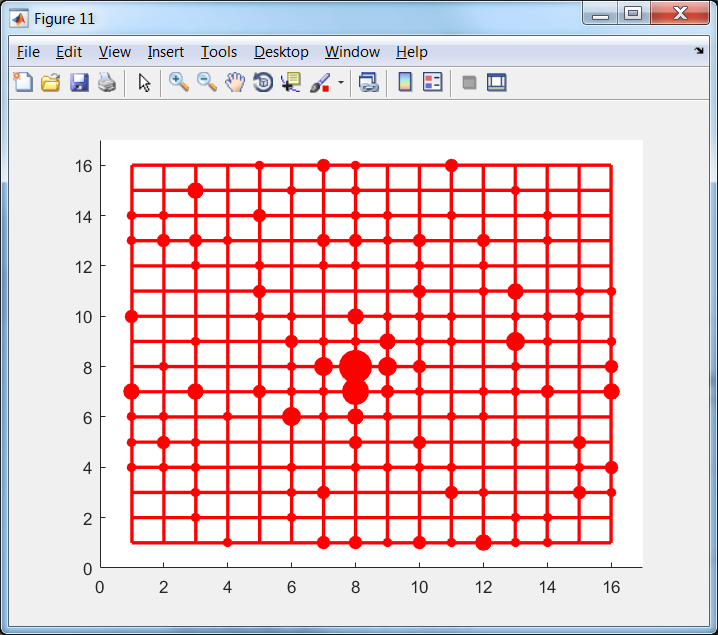
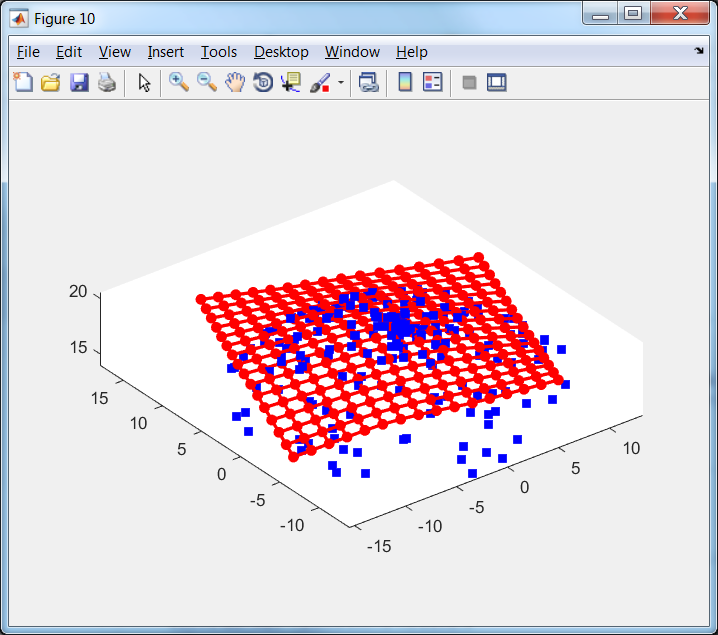
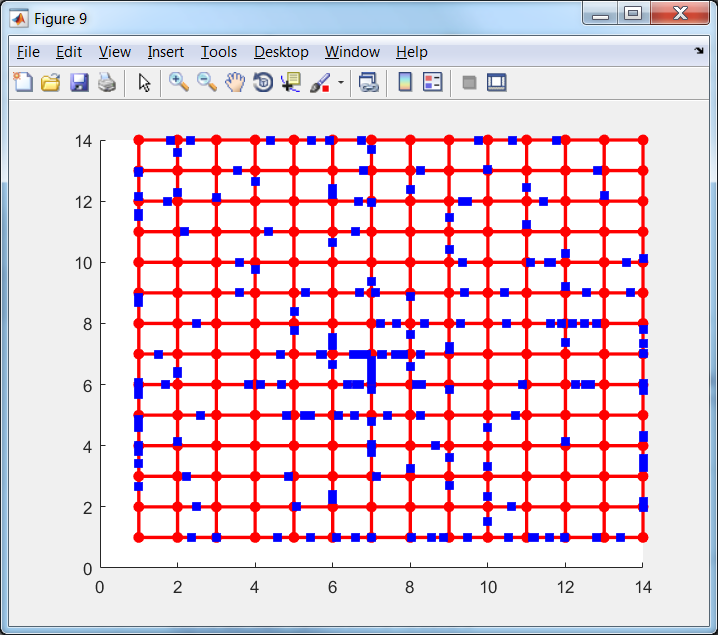
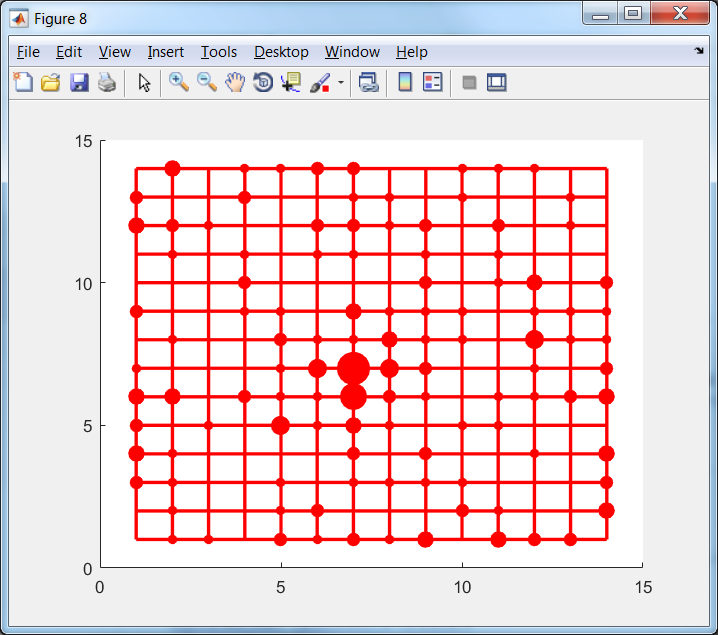
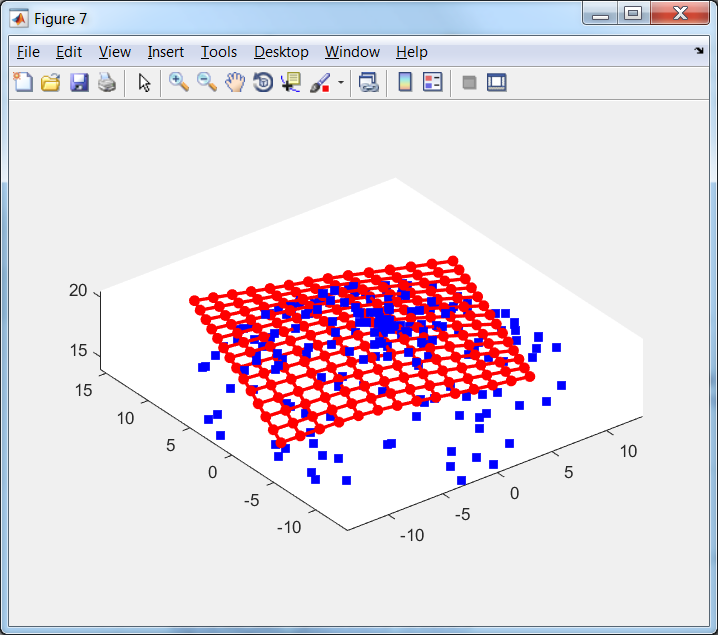
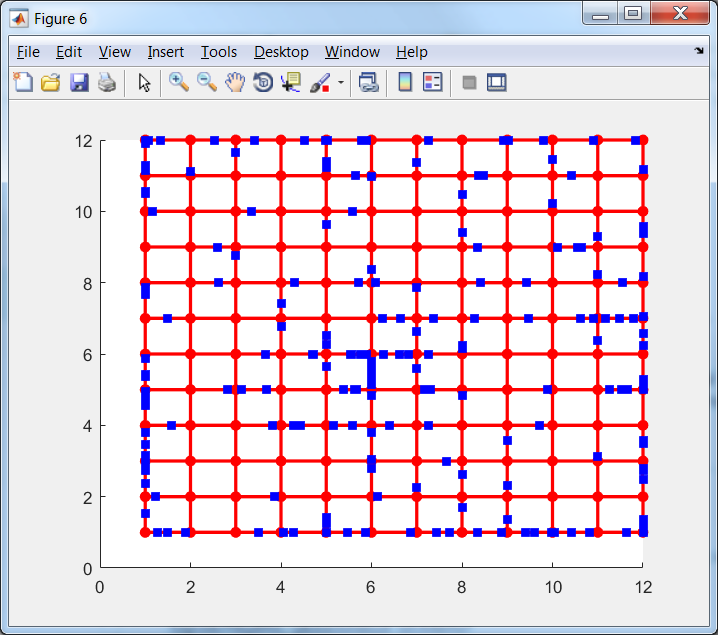
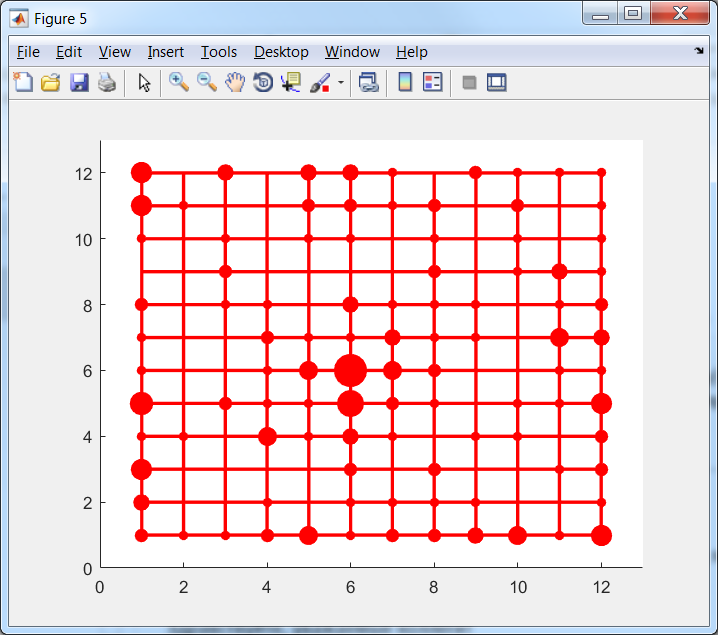
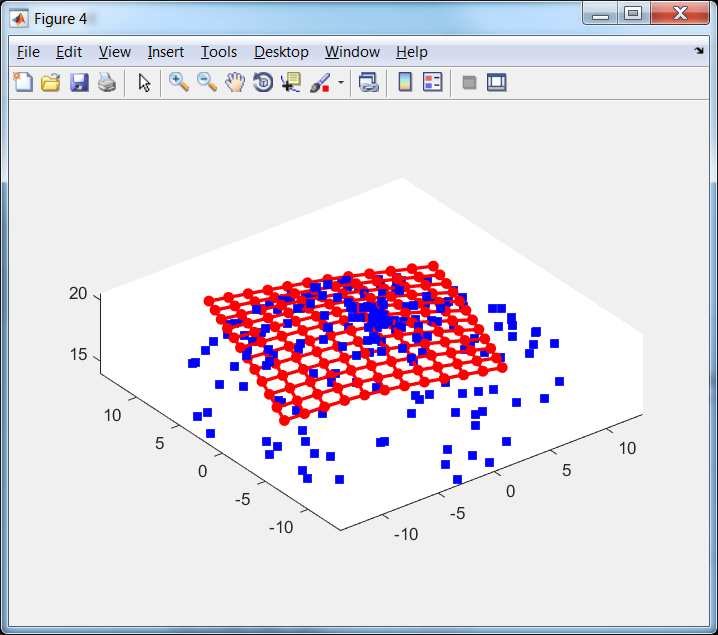
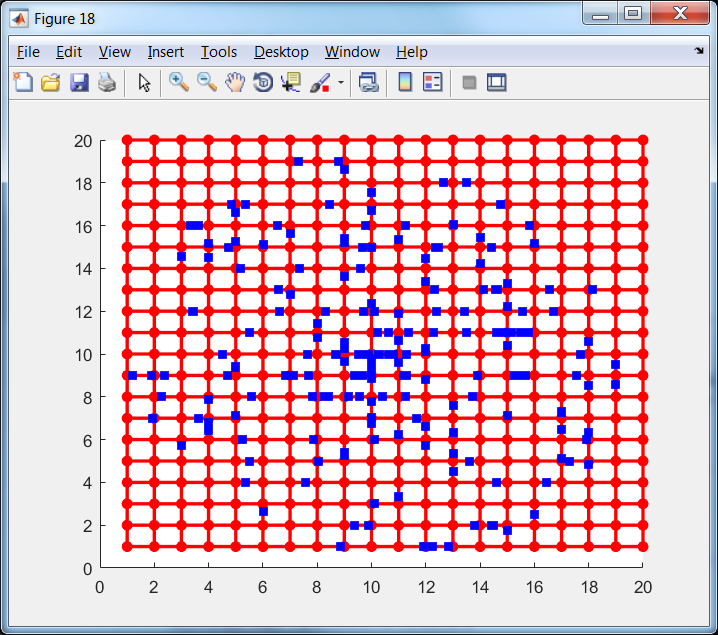
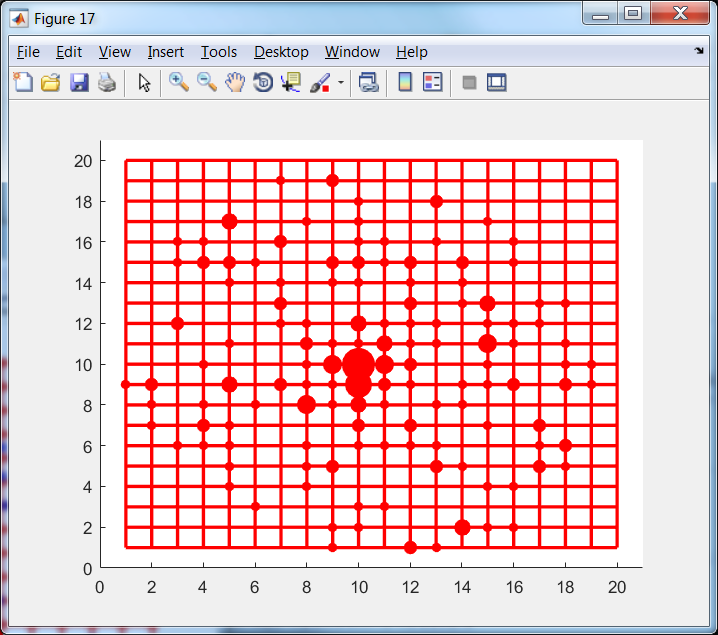
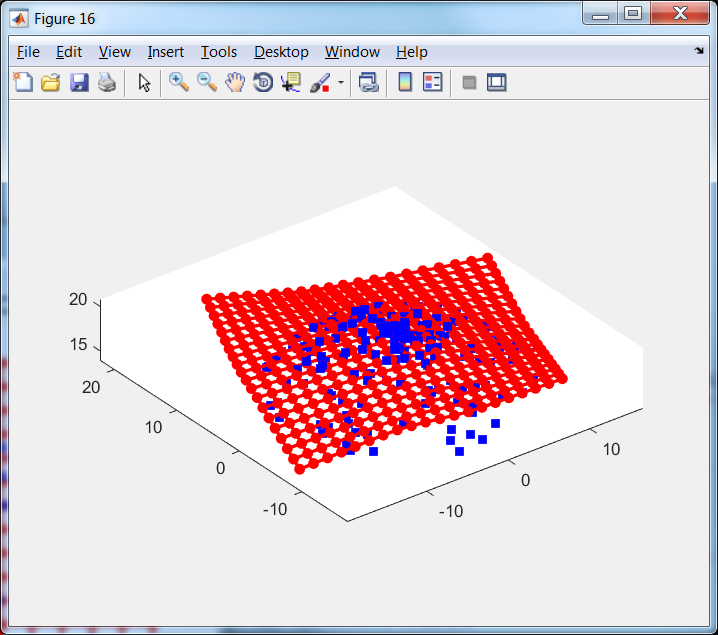
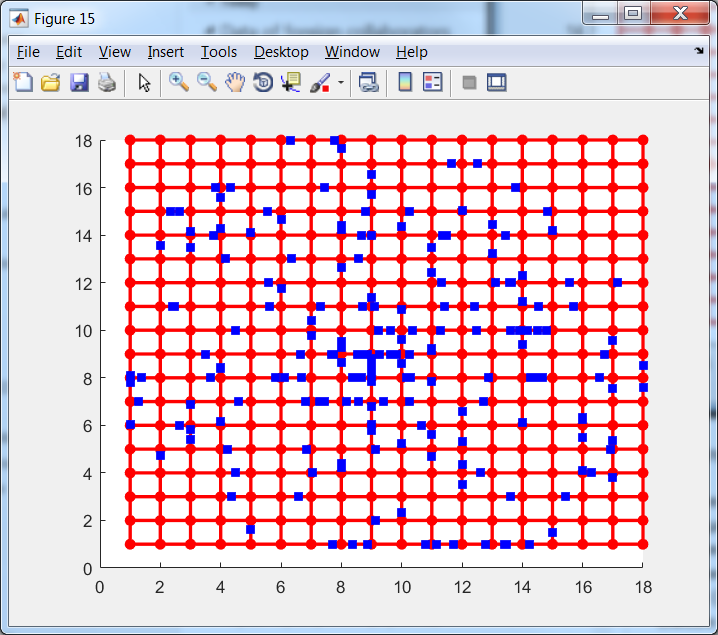
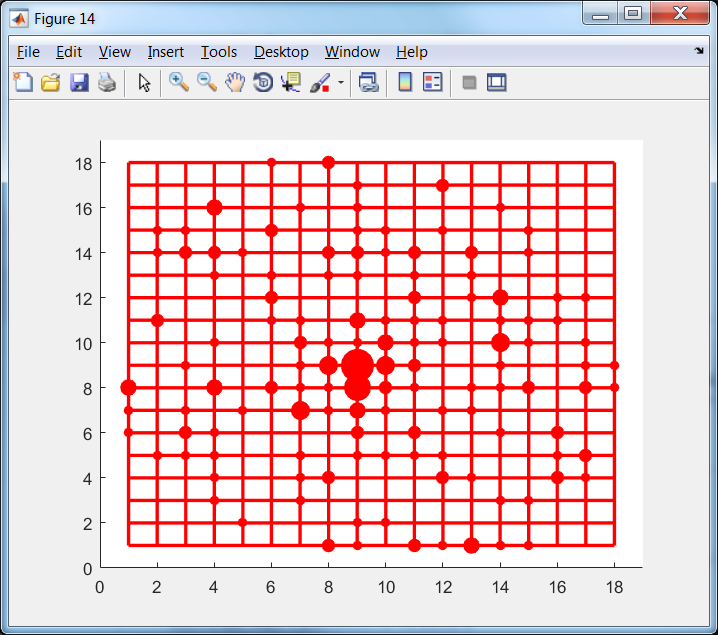
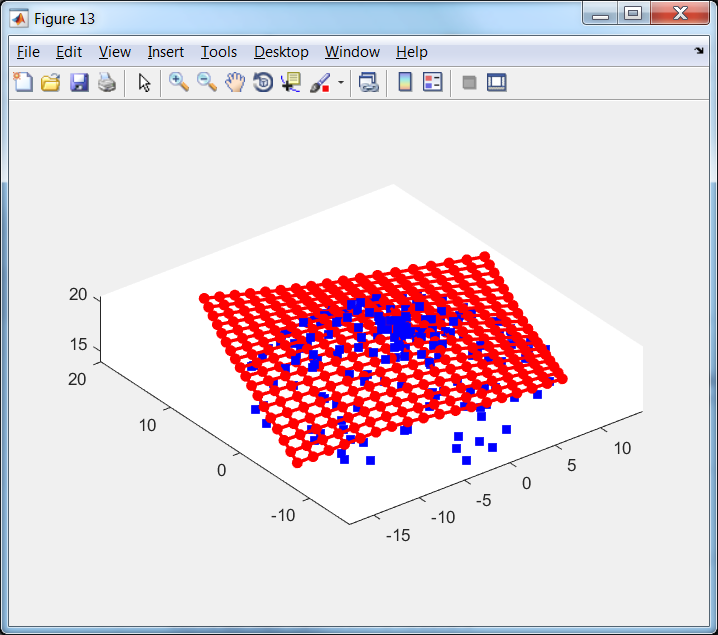
 

Figure 12. Extension of rect2DMap in 3D projection (left), in the internal coordinates with projection onto node (centre) and with projection onto edges (right)

Fractions of border cases are presented in Table 1.

Table 1. Fraction of border cases for example of rect2DMap

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Number of extension | 0 | 1 | 2 | 3 | 4 | 5 |
| Fraction of border cases | 0.595 | 0.460 | 0.325 | 0.200 | 0.105 | 0.025 |

## Extension for tri2DMap

We train map by Elastic map with following arguments: EM(map, data, 'stretch', 0.001, 'bend', 0.1, 'potential', @LLog, 'Number\_of\_intervals', 5). This method produce not very good map but this map is appropriate to use extension.

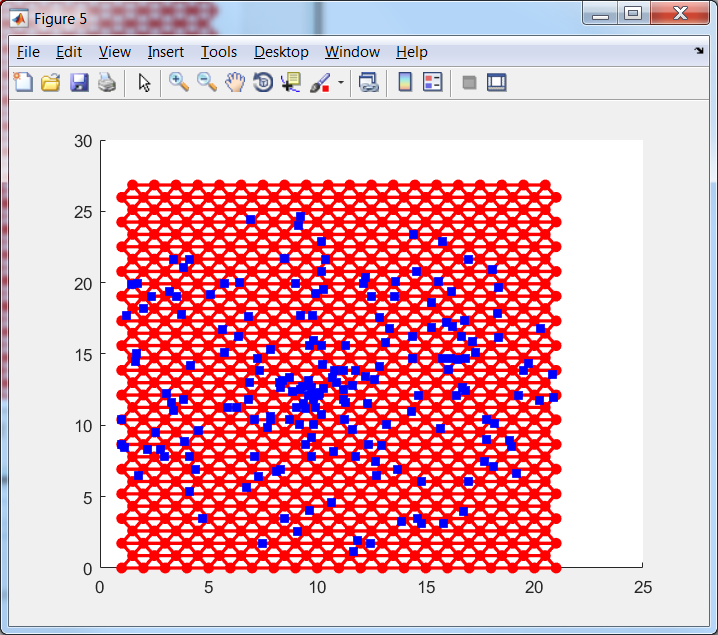
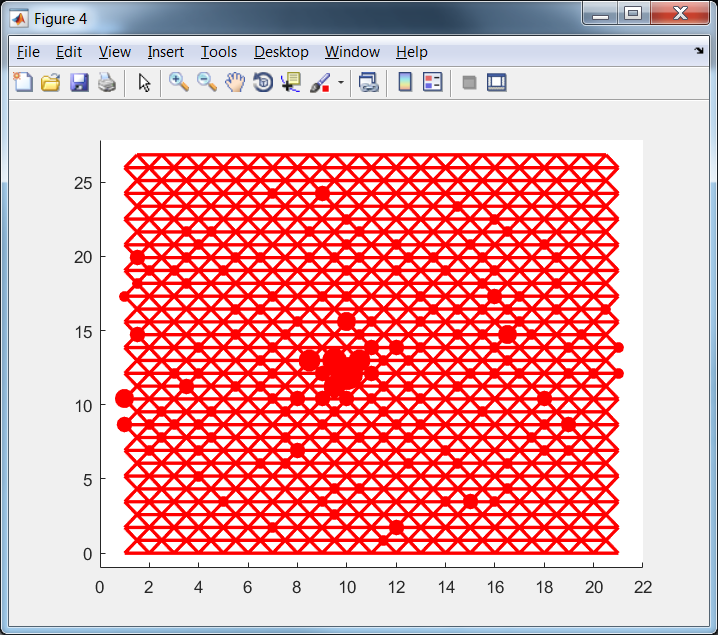
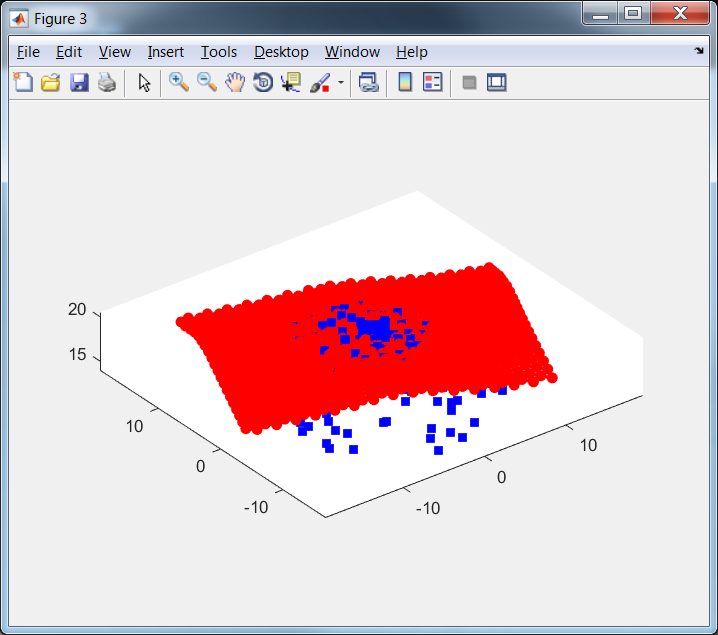
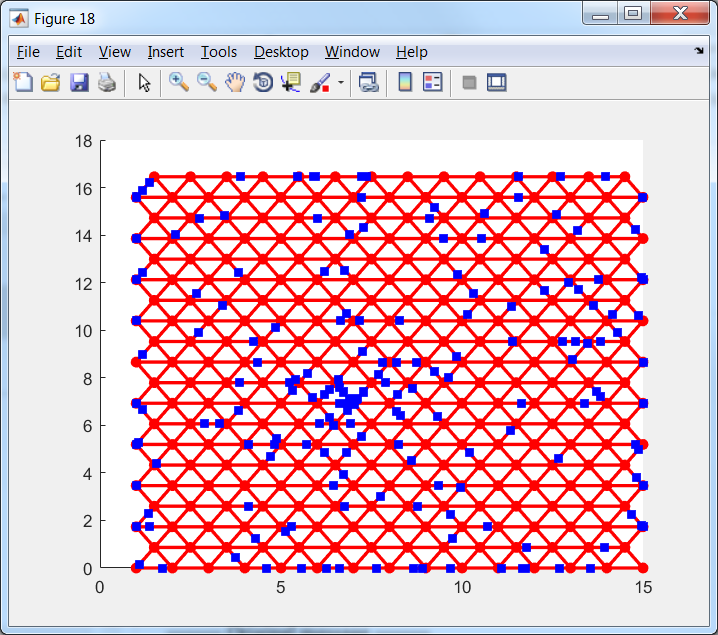
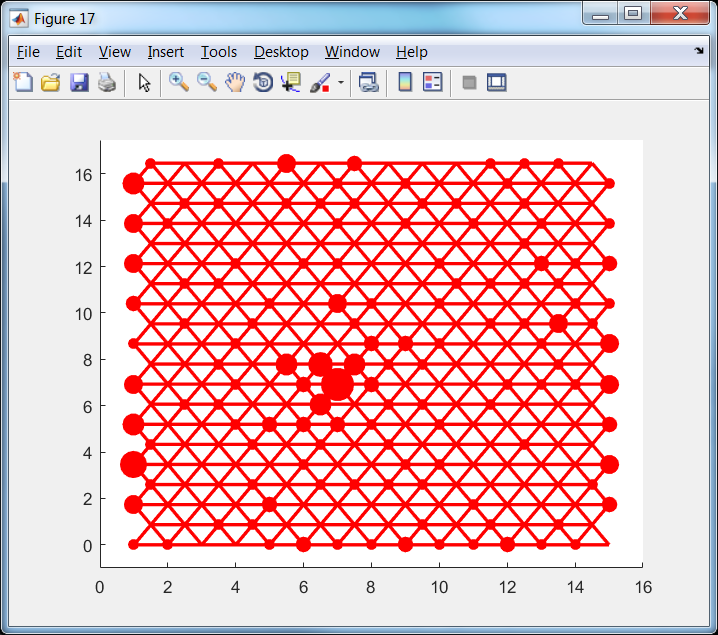
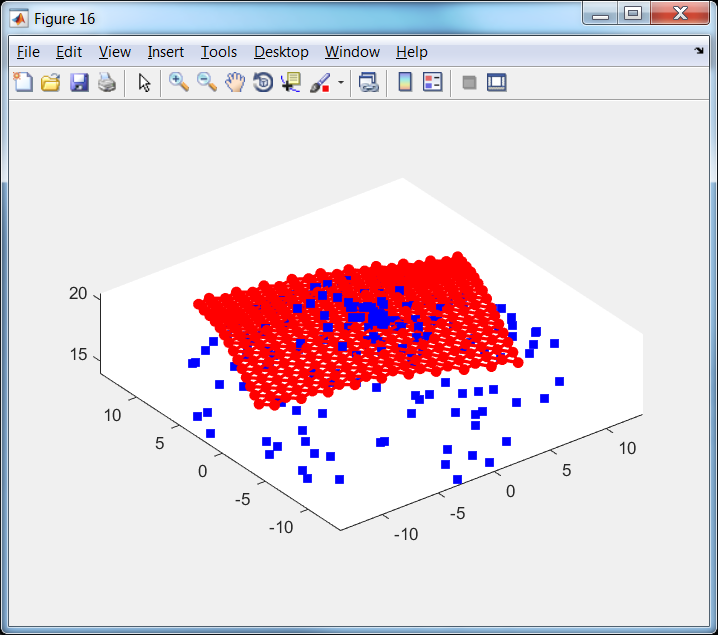
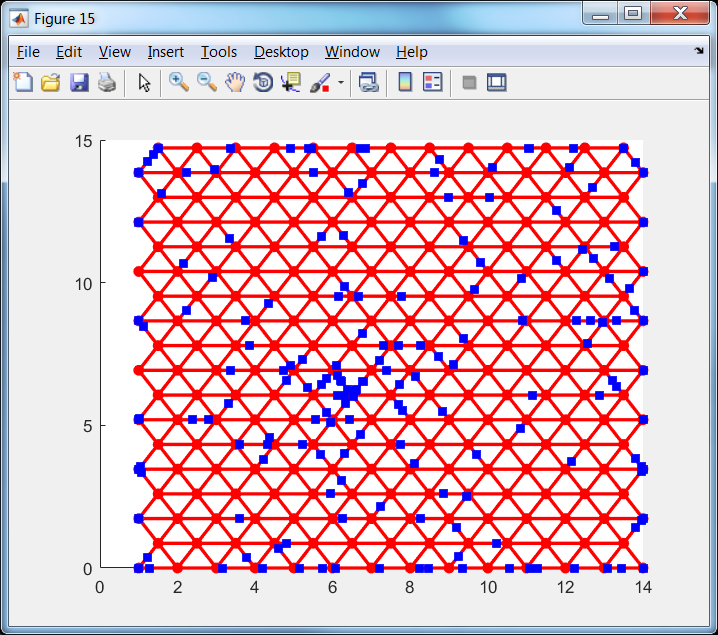
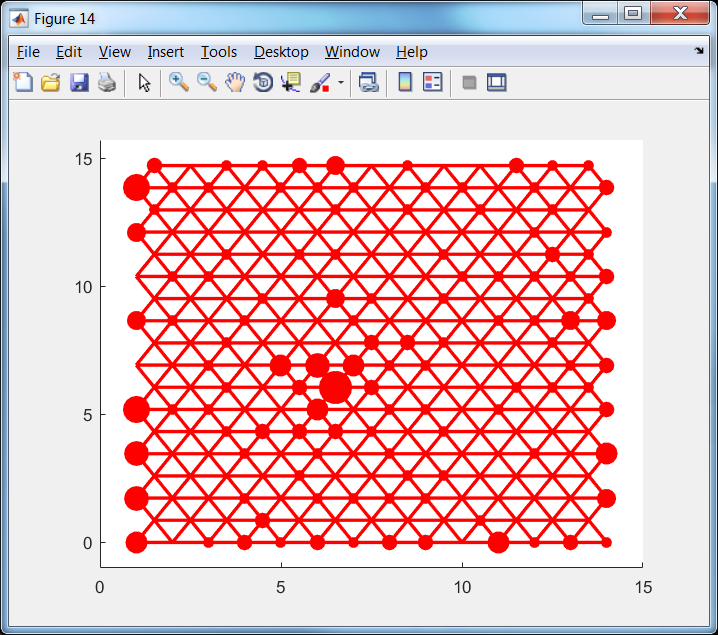
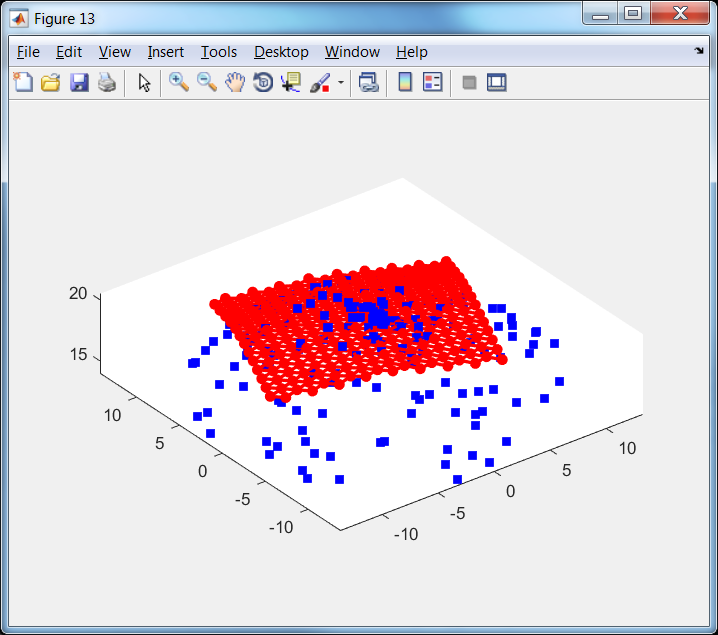
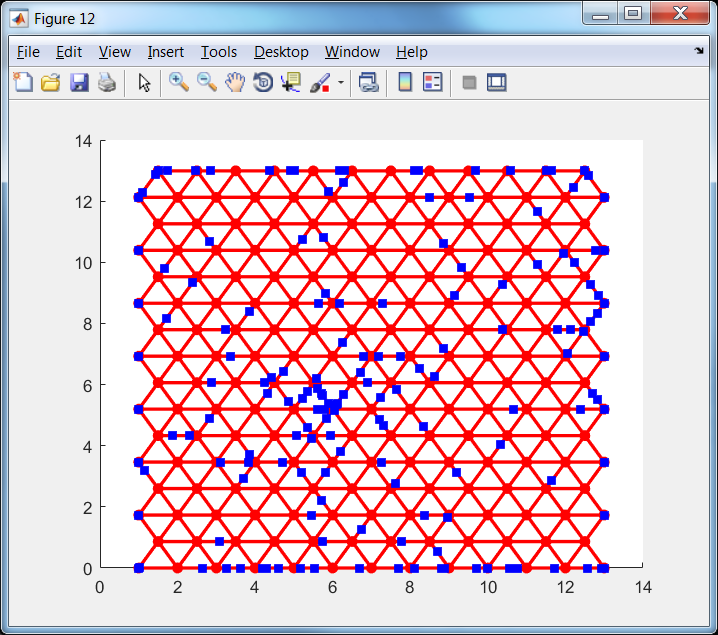
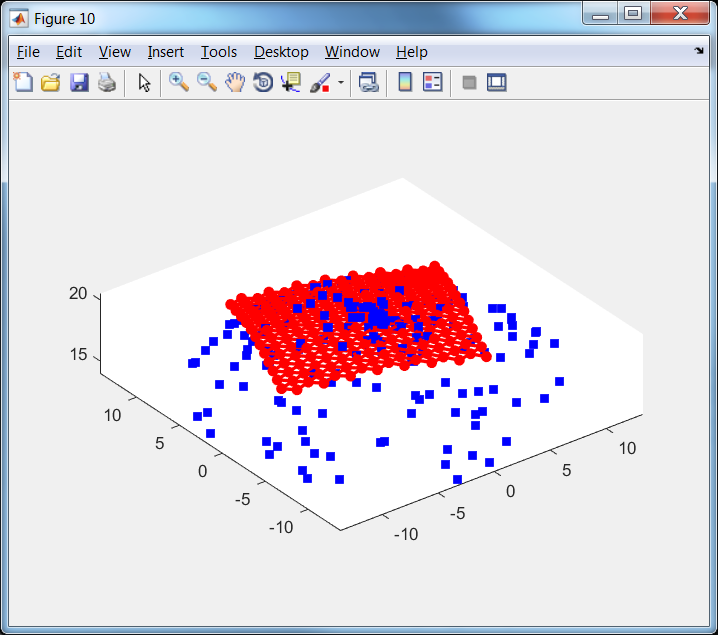
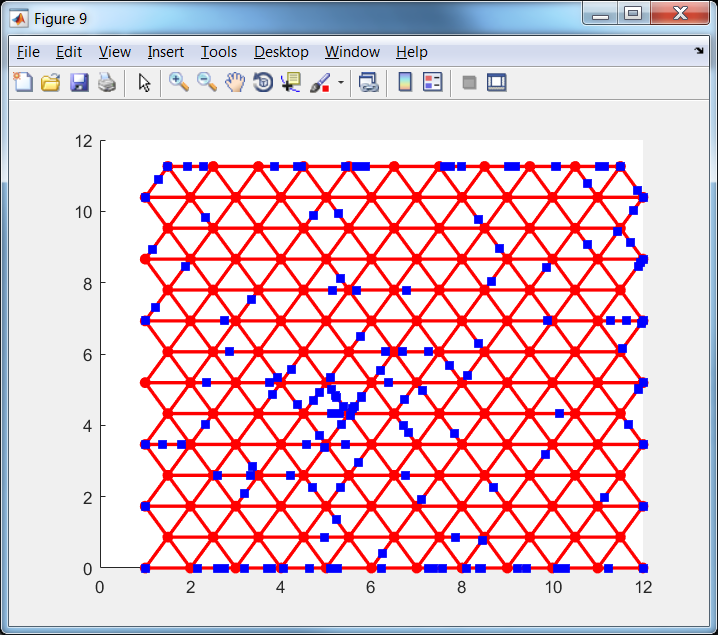
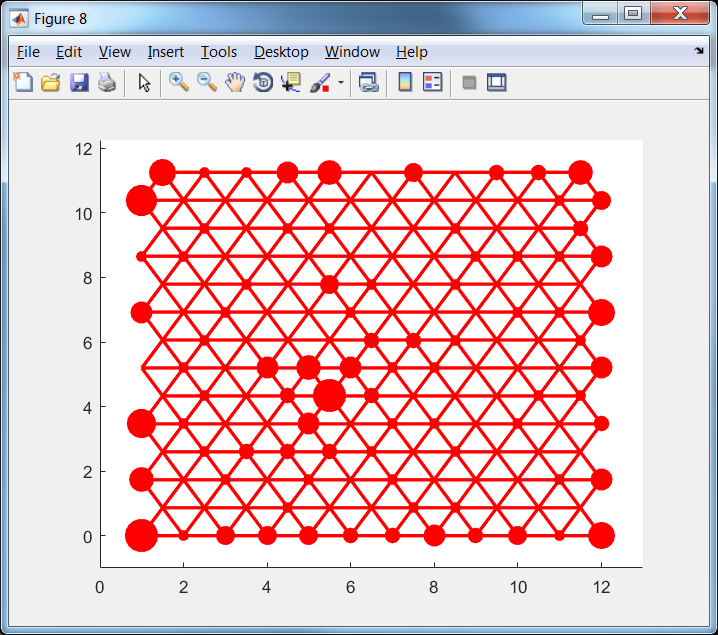
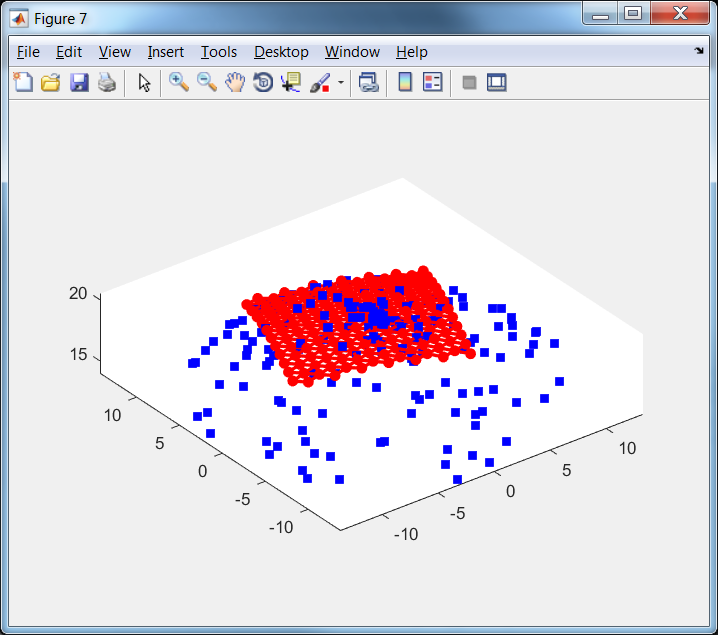
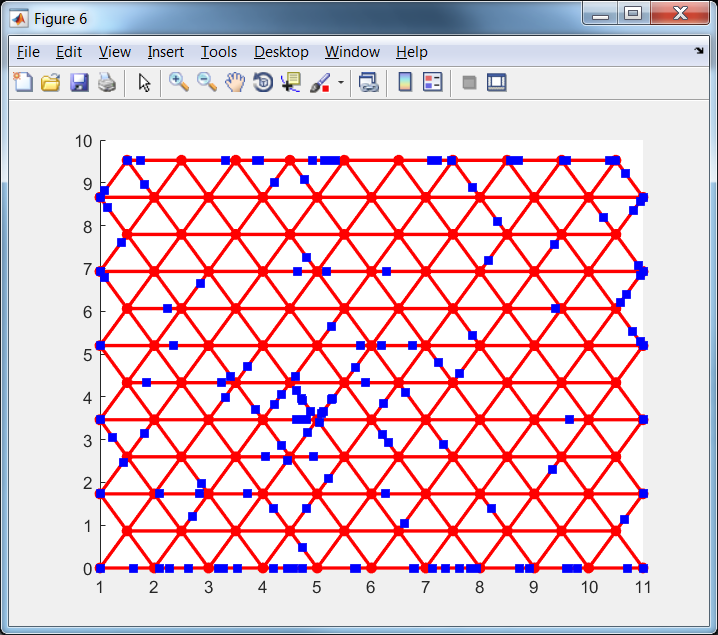
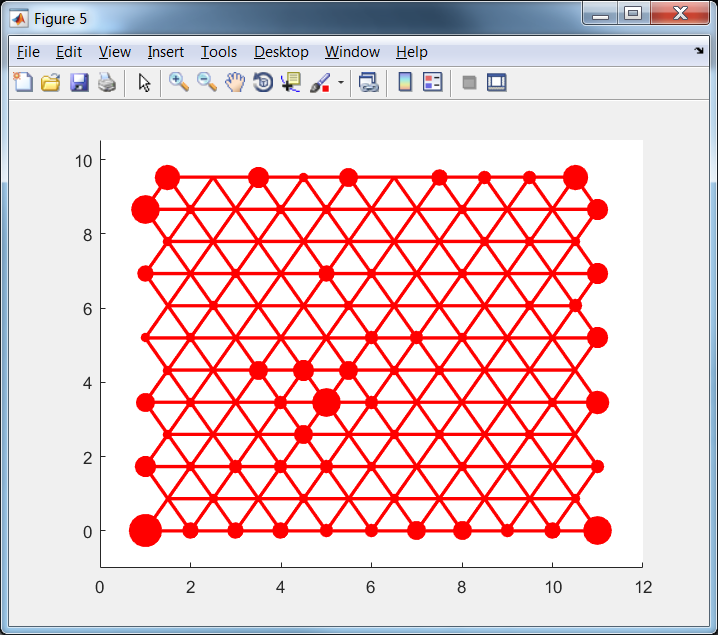
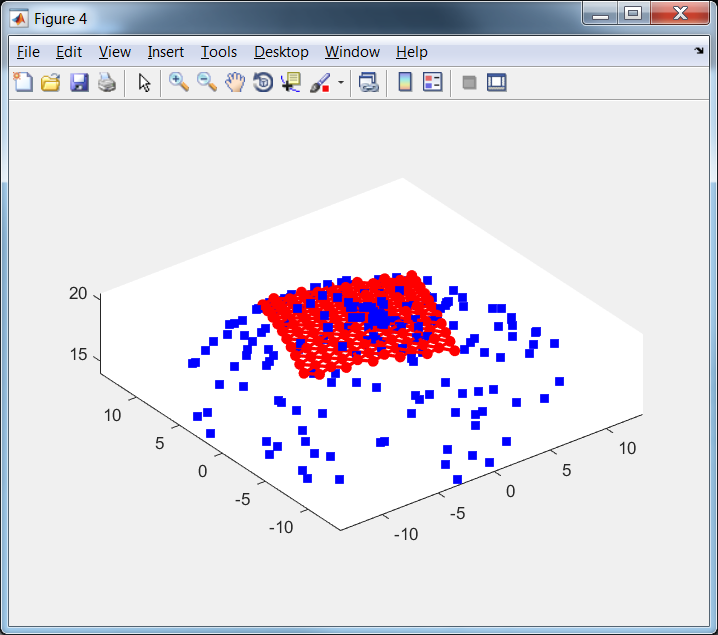
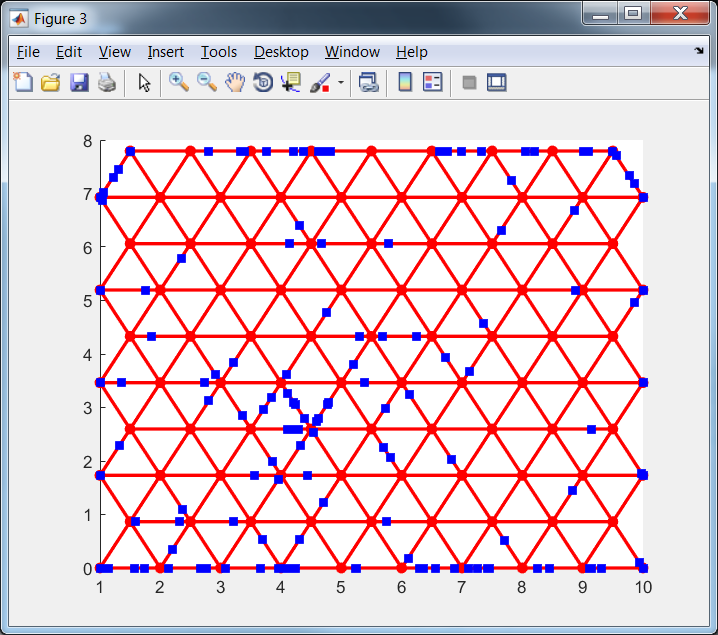
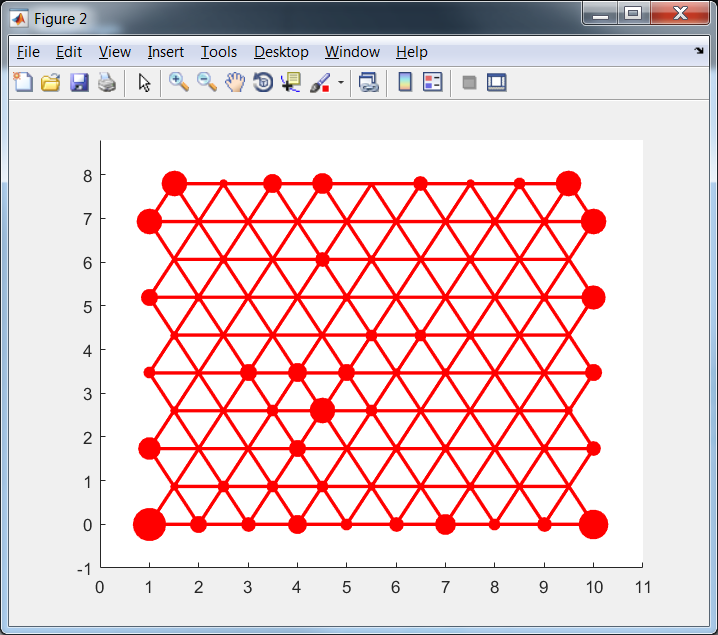
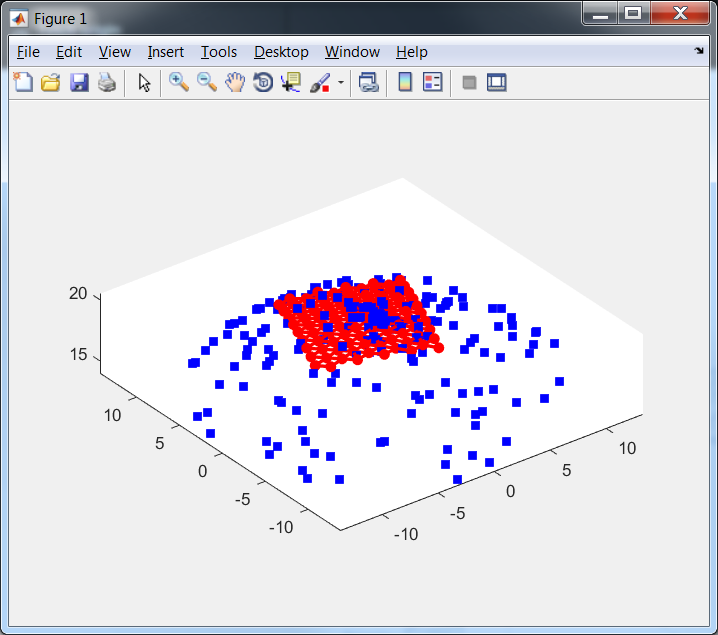


Figure 13. Extensions of tri2DMap in 3D projection (left), in the internal coordinates with projection onto node (centre) and with projection onto edges (right)

Fractions of border cases are presented in Table 2.

Table 2. Fraction of border cases for example of tri2DMap

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Number of extension | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
| Fraction of border cases | 0.69 | 0.62 | 0.58 | 0.50 | 0.42 | 0.36 | 0.29 | 0.22 | 0.14 | 0.10 | 0.08 | 0.04 |

It is very intrigue that for triangular map number of extension to provide the same fraction of border cases is significantly greater. This is effect of different extension rate in different directions: the first map in Figure 13 has 10 rows and 10 nodes in the bottom row but the last map has 32 rows and 21 nodes in the bottom row.

# EM (Elastic map)

EM is function which fit parameters of elastic map. Elastic map is introduced in paper [1] and detailed description is contained in [2]. The main idea of this approach is to search map as solution of optimization problem.

## Elastic energy

Let us have set of data points in dimensional space and map which contains nodes which connected by edges , where are number of node. Edges form ribs , where are number of node.

Our goal is to find map which (i) accurately approximate data and (ii) is smooth. To formalize the first requirement we can consider projection of each data point into nearest node and require the minimization of sum of squared distances between data points and projection of data points:

|  |  |
| --- | --- |
|  | (17) |

where is the node which is nearest to point .

To formalize the requirement for map to be ‘smooth’ we can penalize disturbance of smoothness. Let us use metaphor of elasticity: we have to forbid ‘big’ tension of edges and ‘big’ bending of ribs. For the first purpose we introduce stretching energy term:

|  |  |
| --- | --- |
|  | (18) |

where is stretching modulo. To prevent the big bending we introduce bending energy term:

|  |  |
| --- | --- |
|  | (19) |

where is bending modulo. Combination of data approximation term (17), stretching term (18) and bending term (19) forms elastic energy of map:

|  |  |
| --- | --- |
|  | (20) |

To find the best map we need to find minimum of function (20). Minimum of function (20) can be found by two step procedure

1. Association. Find the nearest nodes for each point for fixed nodes.
2. Minimization. Minimize energy (20) for fixed set of .

These two steps are repeated several times until the set of for two association steps become the same. It simple to prove that algorithm converges. The energy (20) is nonnegative. Let us consider the values of energy for two steps. Let us have value and set of after the association step. The following step is minimization of energy. It means that value of energy after this step can be less or equal to . If then location of nodes does not change and set of new nearest nodes is the same. It means that algorithm converged. If then part of new nearest nodes is different. Let us compare the energy value after the finding of new set of nearest nodes with . Step of association of data points with nearest nodes does not change terms and . It means that difference between and is in the term only. Let us select data point such that . Since is nearest node to the point we have The same inequality holds for all such that . For all data points such that we have equality . contains summands and contains summands . It means that if at least for one data point we have . If there is no points which change nearest node then algorithm converged. Finally we proved that for each step of algorithm value of energy becomes less or equal to the value of energy before this step and equality of energy values before and after step means that algorithm converged. Value of energy is restricted by zero. Number of possible sets is finite. It means that algorithm will be stopped after finite number of steps.

Step of data points association with nearest nodes requires calculation of distances between each data point and each nodes and selection of minimum for each data point.

The minimum of energy for fixed association can be found by differentiation of energy (20) with respect to each coordinate of each node:

|  |  |
| --- | --- |
|  | (21) |

Let us consider each term separately. For data approximation term we have

|  |  |
| --- | --- |
|  | (22) |

For stretching term we have

|  |  |
| --- | --- |
|  | (23) |

For bending term we have

|  |  |
| --- | --- |
|  | (24) |

We can see that equation (21) can be written as system of linear algebraic equations

|  |  |
| --- | --- |
|  | (25) |

where by matrices correspond to data approximation, stretching and bending terms correspondingly, is matrix each row of which is coordinates of one node and is matrix with rows and columns. It is important that matrices and are data independent and can be calculated once. Matrix is data dependent and must be recalculated after each association step. All coordinates of nodes are independent.

Let us denote the number of data points which are associated with node . Then we can write matrix :

|  |  |
| --- | --- |
|  | (26) |

Matrix can be written as

|  |  |
| --- | --- |
|  | (27) |

Matrix can be calculated iteratively. Put . For each edge perform modification of matrix:

Matrix can be calculated iteratively. Put . For each rib perform modification of matrix:

## Weighted version

Data points can have weights. In this case data term of energy has kind:

|  |  |
| --- | --- |
|  | (28) |

Derivative of this function is

|  |  |
| --- | --- |
|  | (29) |

Matrices and can be rewritten in the form

|  |  |
| --- | --- |
|  | (30) |
|  | (31) |

## and choice

Selection of appropriate values of and is very important problem.

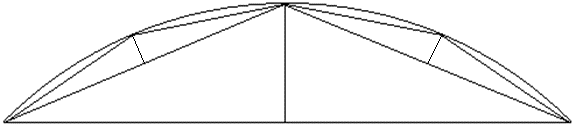
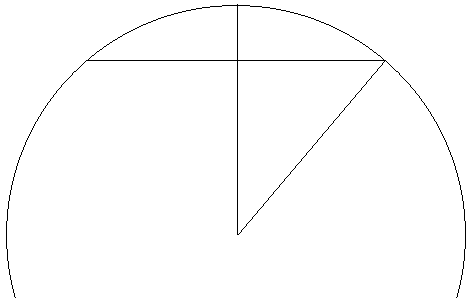
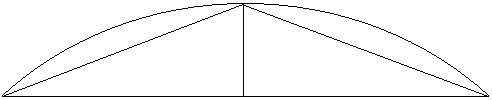
It looks like reasonable to have the same stretching energy for maps with different number of nodes. Let us consider map with one edge of length. Stretching energy of this map is

Let us split this edge into smaller equal edges. Then we can write

Since we want to have the same energy we can write:

This means that if we define stretch modulo for one fragment as then for chain of edges we have to use modulo.

Let us require to have the same bending energy for maps with different number of nodes. Let us consider two maps for the circle (see Figure 14).



a

b

c

Figure 14. Two maps for fragment of circle with two (a and c) and four (b) edges

Let us denote the coordinates of three points in the left figure as . In this case bending elastic energy is

Let us calculate the altitude of left (right) triangle in the right figure. To calculate it we can use formula of length of chord:

where is central angle for chord and is radius. For chord from the first point to the last point we can write

Then we can write . Since we know that angle is positive and small enough we can find and :

Required altitude can be found as

We also know that

Now we can calculate

We know that for any reasonable situations. This means that we can estimate Now we can write

For the second map we have three equal summands in bending energy:

We want to have equal energies. This means

Let us generalise the last formula. First of all, formula is correct for all case of which is small relatively length of edge. Now let us calculate energies for chain of edges of equal length. This chain contains ribs and has energy

Now let us split each edge in two equal edges by analogy of Figure 14. In this case we have chain with ribs end energy

Since we want to have equal energy we can write

## PQSQR for data

In the elastic energy formula (17) the sum of squared distances is used for description of data approximation. Sometimes such approach is too sensitive to outliers and usage of Manhattan distance is recommended by many researchers. Unfortunately, usage of any metrics exclude Euclidean requires huge amount of computations. From the other side recently [6] was developed method allows quickly search minimum of arbitrary sub quadratic function.

We have following values specified by user:

1. Set of intervals
2. Majorante function

For each interval it is necessary to calculate coefficients of sub quadratic function with property To minimise difference of functions and under condition it is necessary to put .

Let us calculate coefficients for arbitrary interval :

For the border cases we have

Now we can rewrite data term (17) of energy function as

|  |  |
| --- | --- |
|  | (32) |

Let us calculate derivative

|  |  |
| --- | --- |
|  | (33) |

where is defined by inequality

Now we can rewrite matrix A as

|  |  |
| --- | --- |
|  | (34) |

Matrix can be written as

|  |  |
| --- | --- |
|  | (35) |

## Weighted PQSQ version

According to subsections “Weighted version” and “PQSQR for data” we can rewrite matrices and as

|  |  |
| --- | --- |
|  | (36) |
|  | (37) |

As we can see these formulas are almost the same as (30) and (31) with recalculation of weights as but without recalculation of normalisation term .

# Maps colouring

2D maps both in the internal coordinates and in the three dimensional space can be coloured according with one of selected function. Flat map (map in the internal coordinates) can also be flat (analogue of contour surface representation) or 3D where the height represents value of function.

Examples of map colouring are presented in Demo.docx and <https://github.com/Mirkes/ElMap/wiki>. This section contains only description of used methods. There are three types of colouring specification in ElMap:

1. Function defined in nodes only (values of function in nodes known only).
2. Function defined in any points (we have expression to calculate function).
3. Function defined in data points only

## Grid of colouring

We use faces presented by map for colouring. For smoothness we split each face (triangle) to four sub faces by dividing of each side into two equal fragments (see Figure 15). Values of function are calculated in vertex of each triangle and then linearly interpolated inside triangle.

Figure . Grid detailing

## Function defined in nodes and in arbitrary points

In this case we can directly calculate values of function in nodes of new grid. Example of function defined in nodes can be coordinate of original attributes. Example of function defined in arbitrary point can be Fisher’s discriminant value or logistic regression value.

## Function defined in data points

For function defined in data points we use special procedure to estimate value of function in nodes. There are two different ways of such calculations.

The first way is to put kernel function to each data point and then calculate value of function in each node.

The alternative way is initially project each data point onto map and then apply the same procedure.

One of the properties of high dimension is almost Gaussian form of any distributions [???]. As a result density distribution, estimated by the first way usually looks like Gaussian. Example of density of breast cancer estimated by the first approach is presented in Figure 16 left. It is necessary also note that for such map it is necessary to drastically change value smoothness parameter: in the presented example this value is 30 instead of default value 0.15, used for projected density depicted in Figure 16 right.

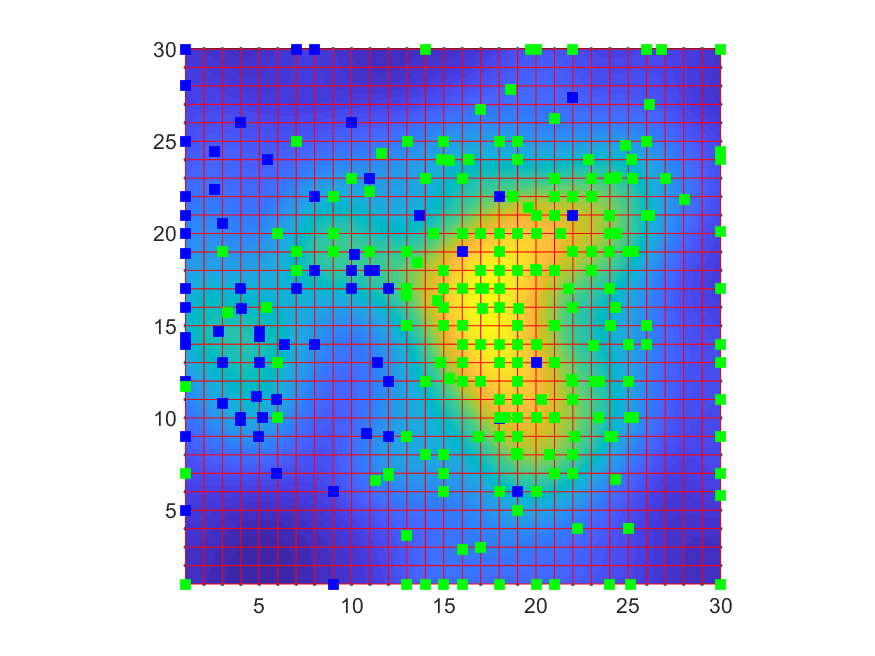
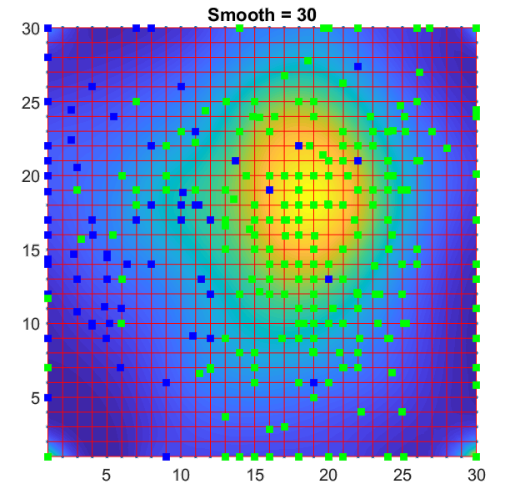


Figure . Density map colouring for density estimated in space (left) and in projection (right)

There are many types of kernel functions for such usage [9]. For map colouring we decided to use Gaussian kernel in form

|  |  |
| --- | --- |
|  | (38) |

where is smoothness parameter, is node to estimate and is mean of individual attribute variances:

|  |  |
| --- | --- |
|  | (39) |

where is attribute. For projected version of colouring we have two attributes only.

Since we are interested in the relative values of density value of multiplier does not matter.

# Tests

This section contains several tests of map for artificial datasets. Figure caption contains commands to form map.

## Two arcs

Data set contains two arcs with shift 4 in direction. The top arc contains 100 points and the bottom arc contains 400 points.

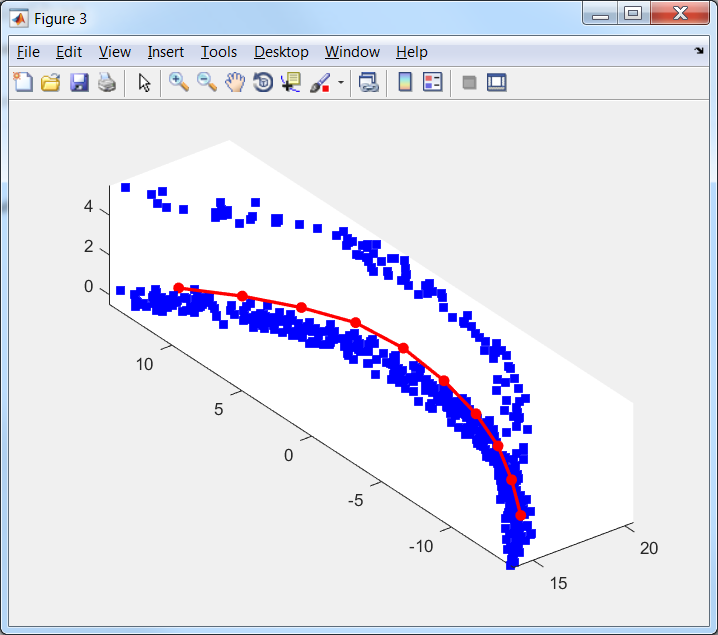
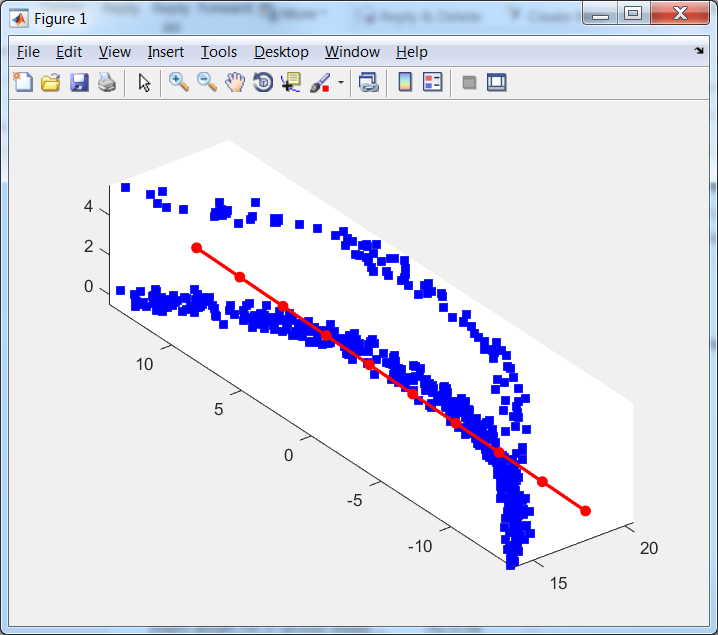


Figure 17. Original and EM(map, data, 'stretch', 0.01, 'bend', 0.1 );

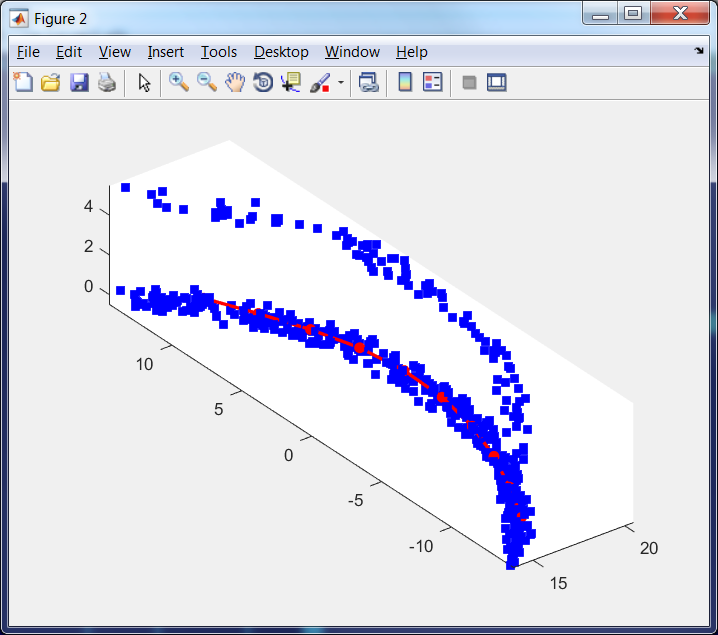
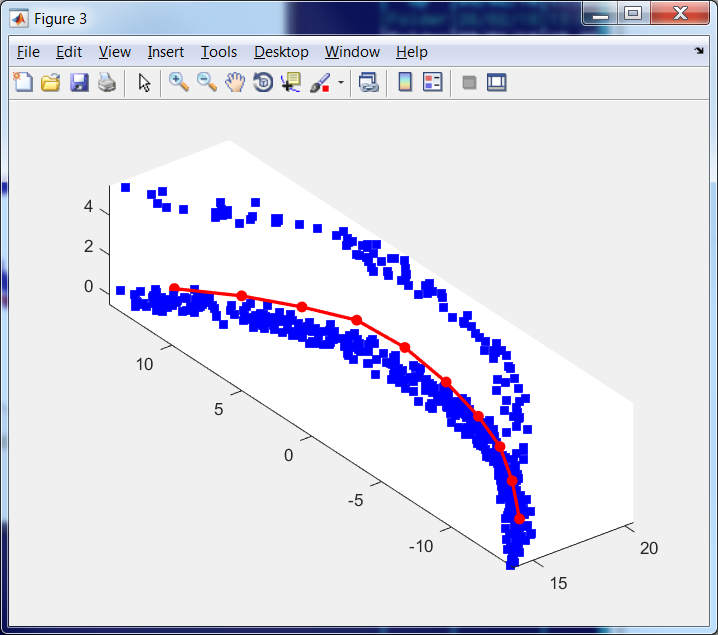


Figure 18. EM(map, data, 'stretch', 0.01, 'bend', 0.1, 'potential', @L2, 'Number\_of\_intervals', 2); left

EM(map, data, 'stretch', 0.01, 'bend', 0.1, 'potential', @L1, 'Number\_of\_intervals', 5); right

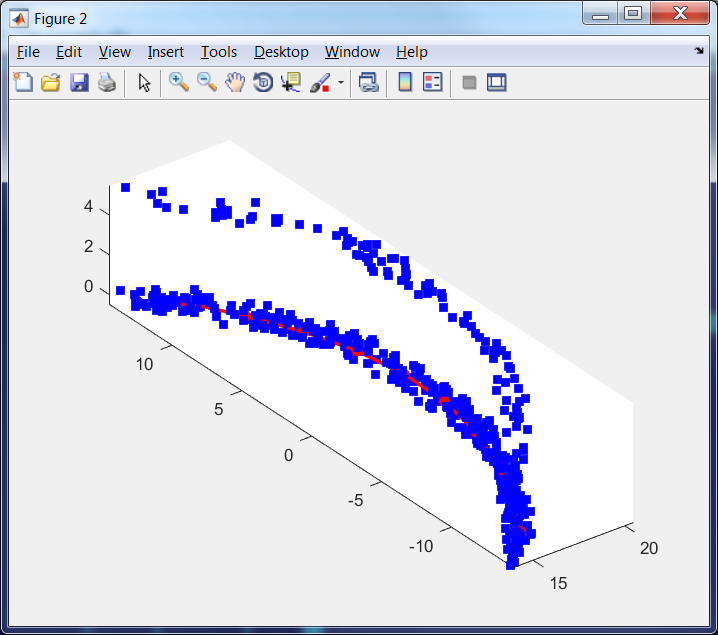


Figure 19. EM(map, data, 'stretch', 0.01, 'bend', 0.1, 'potential', @LLog, 'Number\_of\_intervals', 5); left

EM(map, data, 'stretch', 0.01, 'bend', 0.1, 'potential', @L2, 'Number\_of\_intervals', 2, 'intshrinkage', 0.7); right

We can see that LLog or L1 norm have approximately the same robustness property as L2 with trimming.

## Two arcs with x and y shift

Data set contains two arcs with shift 5 in direction. The shifted arc contains 100 points and the bottom arc contains 400 points.

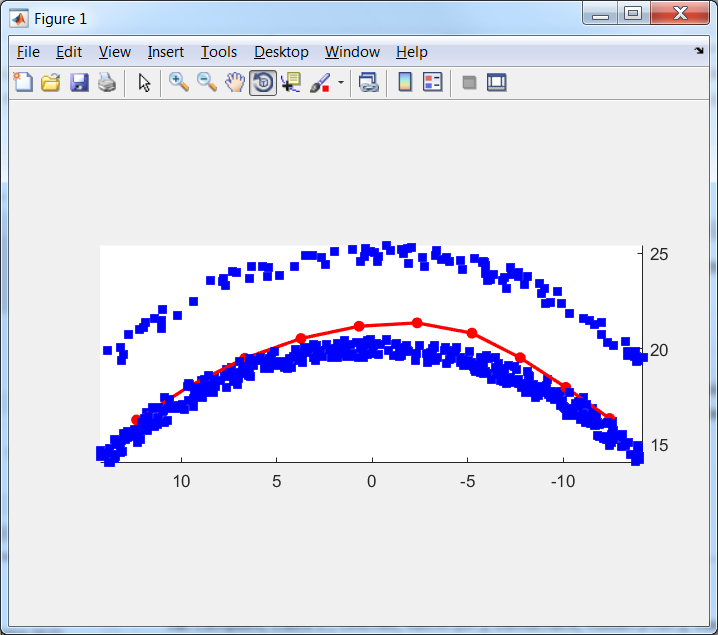
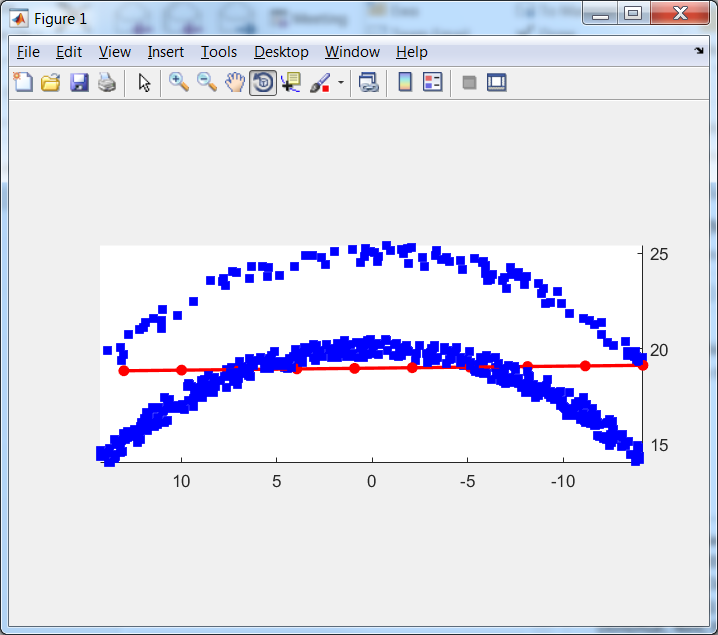


Figure 20. Original and EM(map, data, 'stretch', 0.01, 'bend', 0.1 );

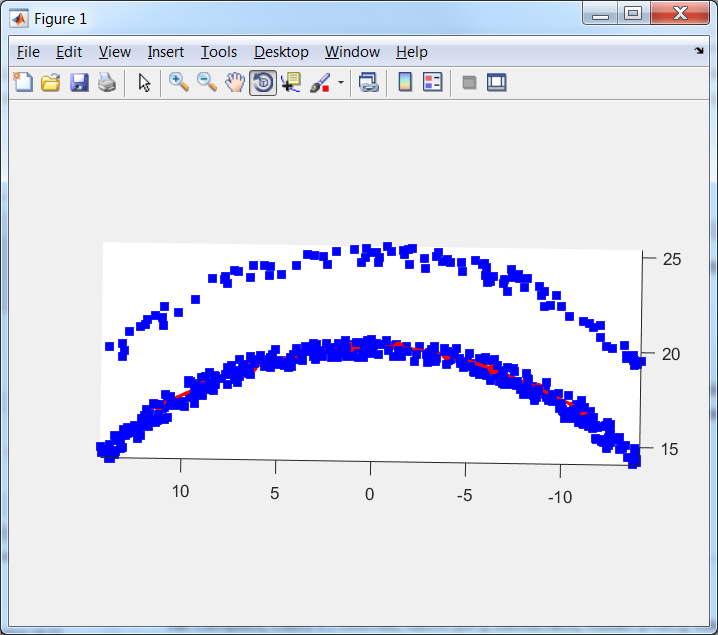
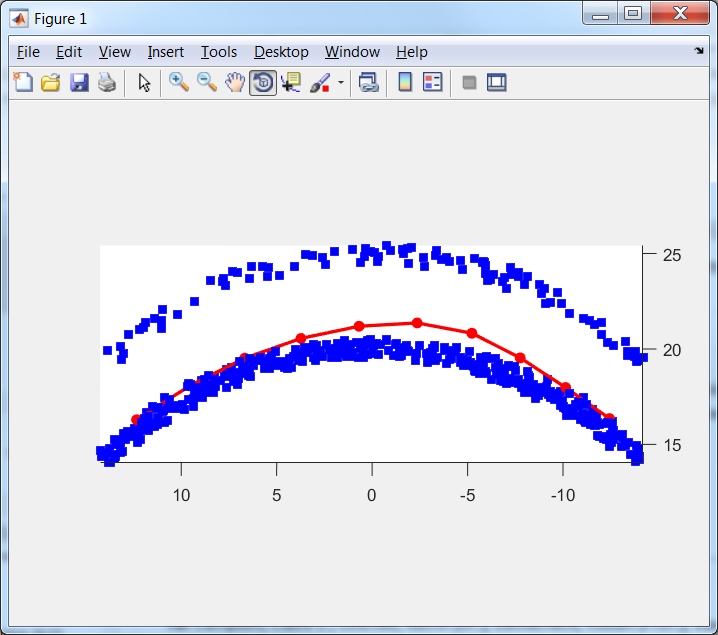


Figure 21. EM(map, data, 'stretch', 0.01, 'bend', 0.1, 'potential', @L2, 'Number\_of\_intervals', 2); left

EM(map, data, 'stretch', 0.01, 'bend', 0.1, 'potential', @L1, 'Number\_of\_intervals', 5); right

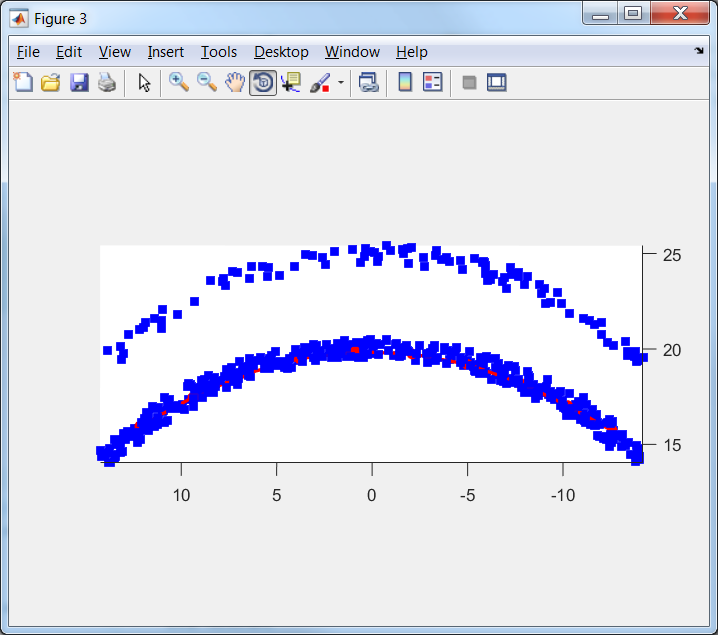
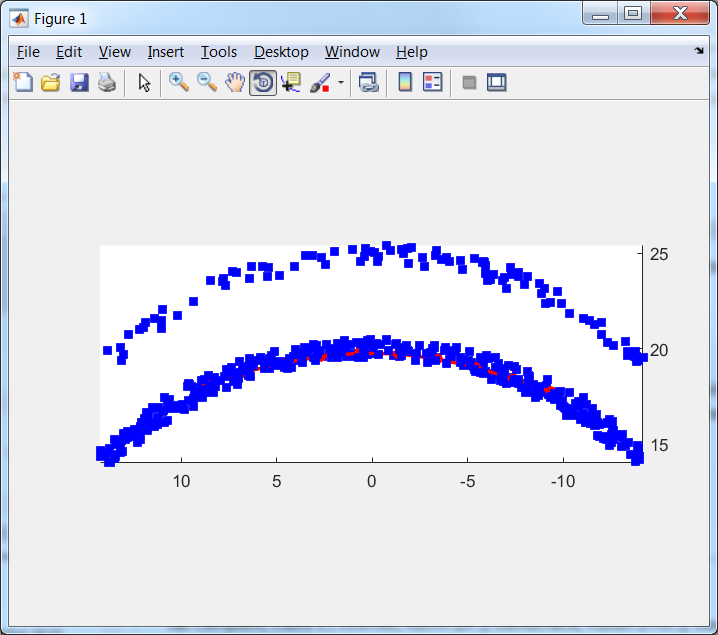
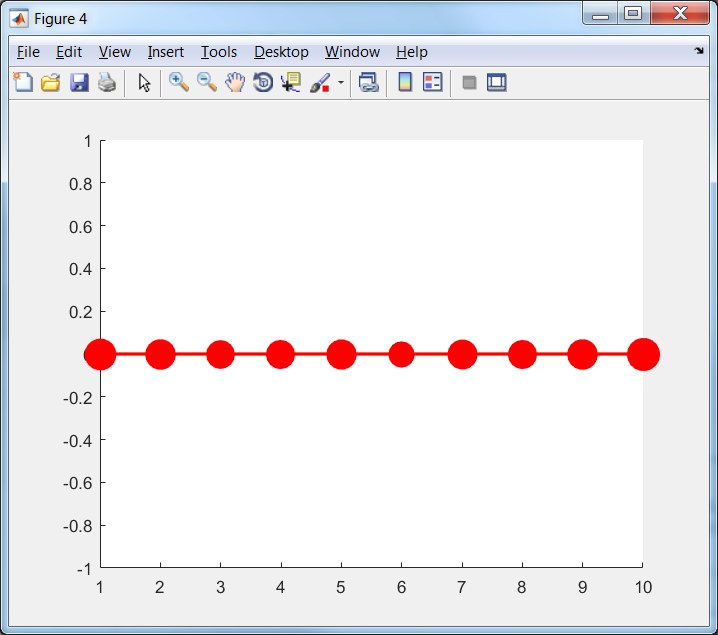
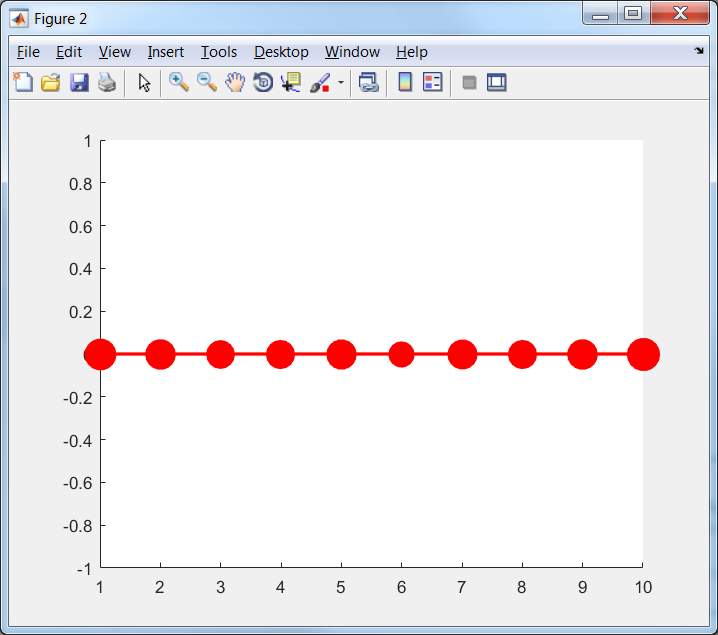


Figure 22. EM(map, data, 'stretch', 0.01, 'bend', 0.1, 'potential', @LLog, 'Number\_of\_intervals', 5); left

EM(map, data, 'stretch', 0.01, 'bend', 0.1, 'potential', @L2, 'Number\_of\_intervals', 2, 'intshrinkage', 0.5);

We can see that LLog or L1 norm have approximately the same robustness property as L2 with trimming. We can also estimate uniformity of nodes loading.



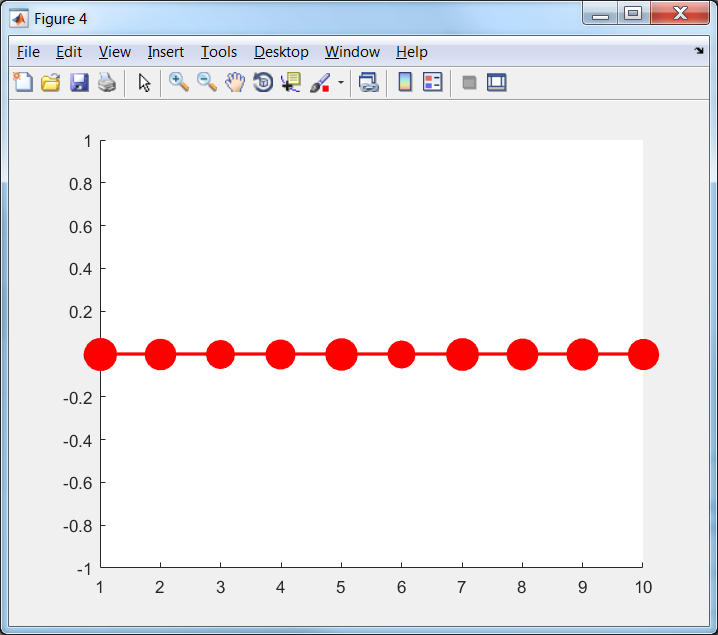


Figure 23. Left to right< top – down:

EM(map, data, 'stretch', 0.01, 'bend', 0.1 );

EM(map, data, 'stretch', 0.01, 'bend', 0.1, 'potential', @L2, 'Number\_of\_intervals', 2);

EM(map, data, 'stretch', 0.01, 'bend', 0.1, 'potential', @L1, 'Number\_of\_intervals', 5);

EM(map, data, 'stretch', 0.01, 'bend', 0.1, 'potential', @LLog, 'Number\_of\_intervals', 5);

EM(map, data, 'stretch', 0.01, 'bend', 0.1, 'potential', @L2, 'Number\_of\_intervals', 2, 'intshrinkage', 0.5);

We can see that all based graphs have approximately uniform distribution of number of points per node (approximately the same size of all nodes). and based maps are considerably less uniform. This effect can be compensated by decreasing of stretch modulo (see Figure 24).

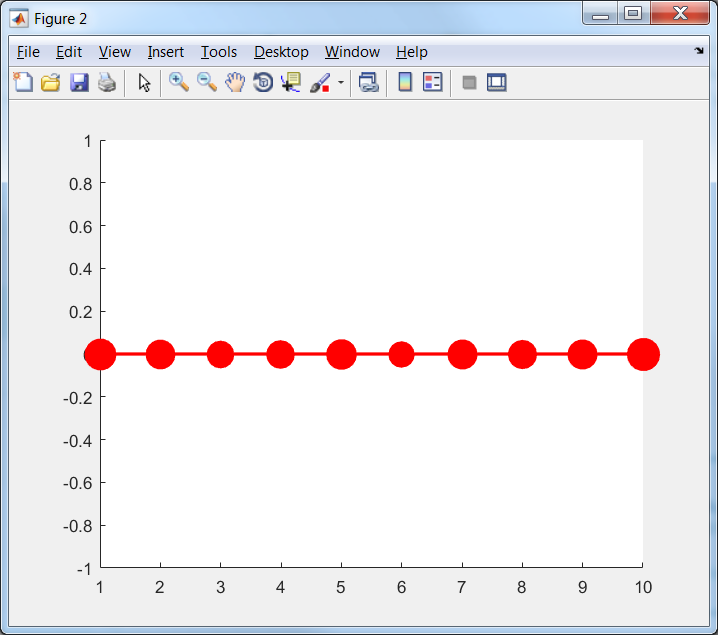


Figure 24. EM(map, data, 'stretch', 0.01, 'bend', 0.1, 'potential', @L1, 'Number\_of\_intervals', 5); left

EM(map, data, 'stretch', 0.001, 'bend', 0.1, 'potential', @L1, 'Number\_of\_intervals', 5); right

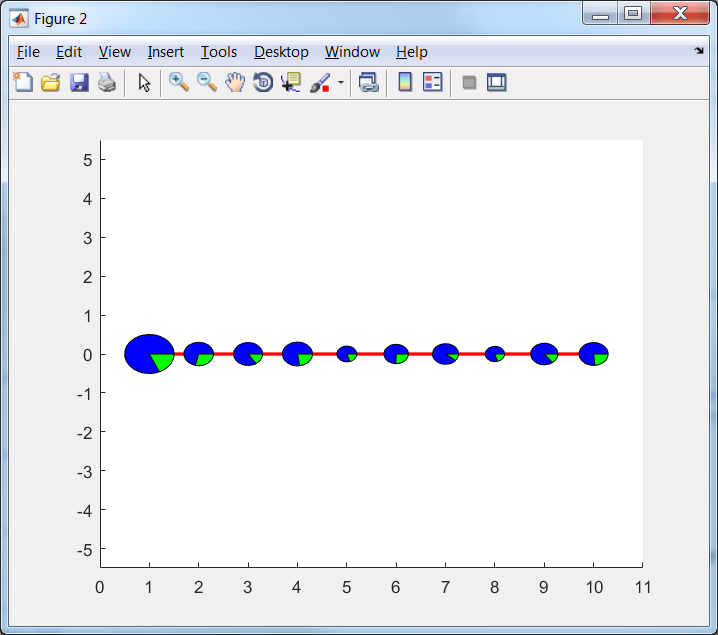


Figure 25. EM(map, data, 'stretch', 0.001, 'bend', 0.1, 'potential', @L1, 'Number\_of\_intervals', 5);

## Fragment of sphere

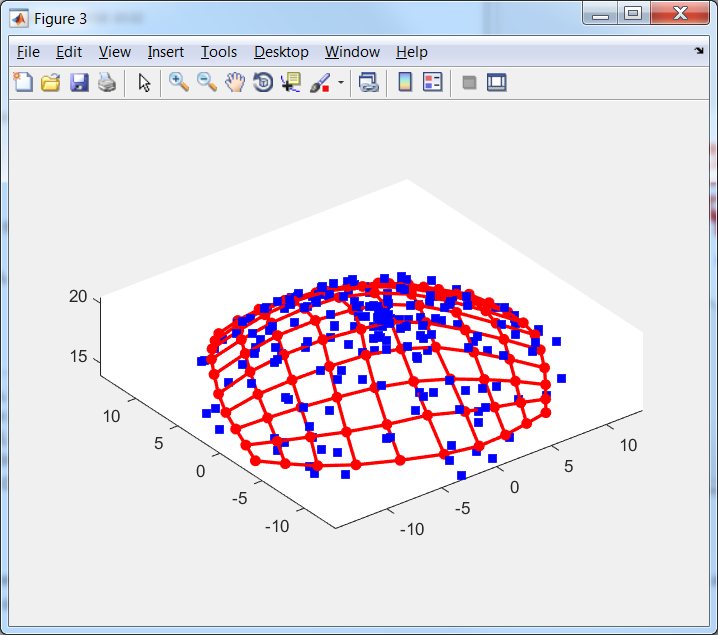
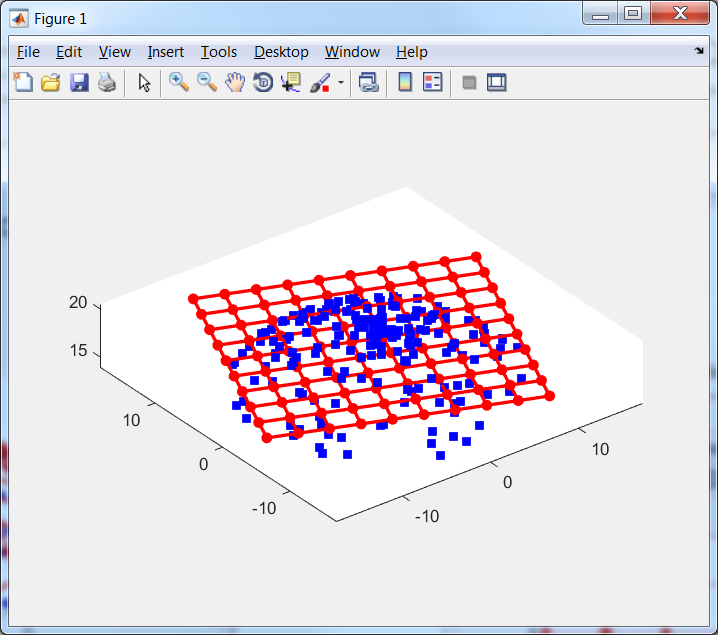


Figure 26. Original and EM(map, data, 'stretch', 0.001, 'bend', 0.01);

## Two fragments of sphere

Data set contains two fragments of sphere with shift 5 in direction. The shifted sphere contains 100 points (green) and the bottom fragment of sphere contains 400 points.

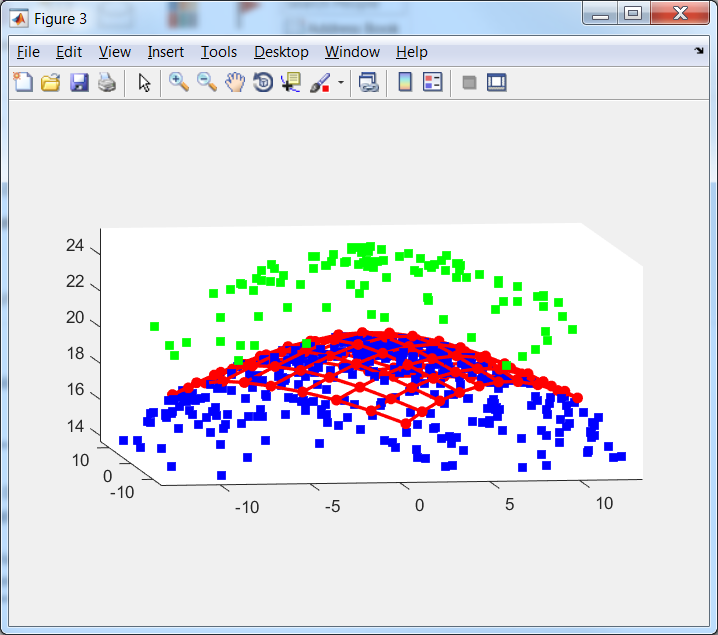
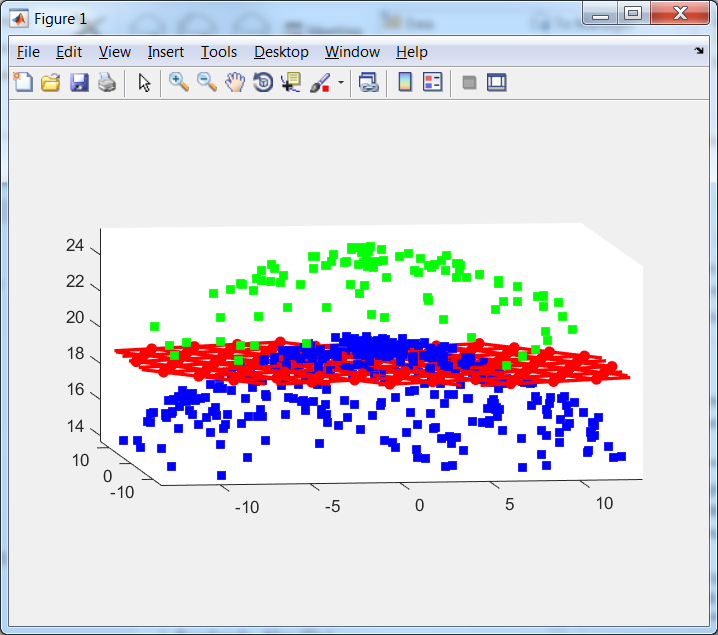


Figure 27. original and EM(map, data, 'stretch', 0.01, 'bend', 0.1);

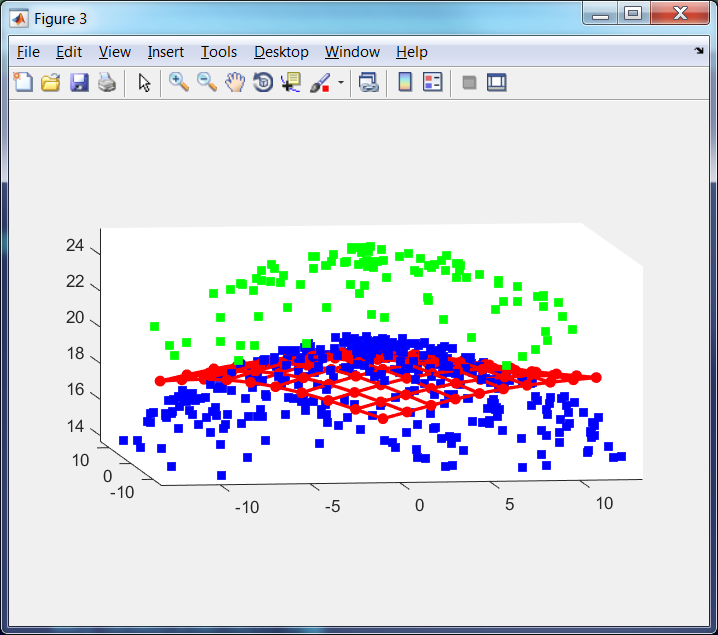
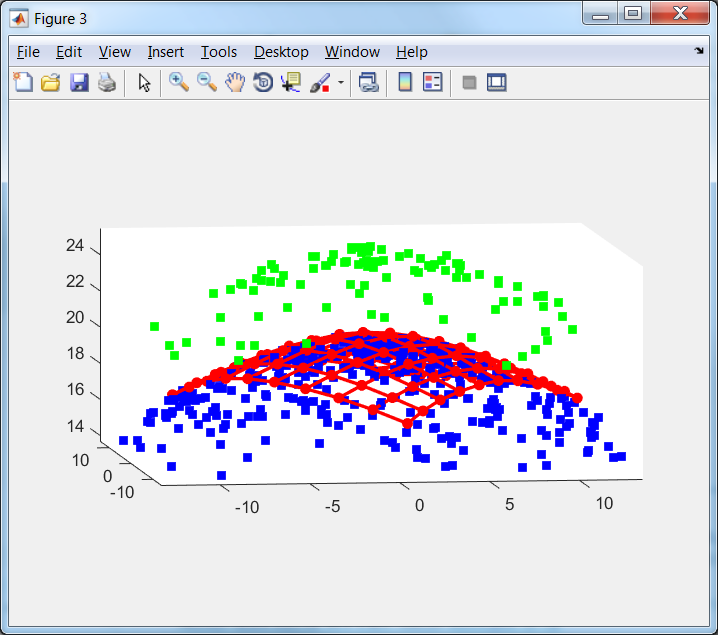


Figure 28. EM(map, data, 'stretch', 0.01, 'bend', 0.1, 'potential', @L2, 'Number\_of\_intervals', 2); left

EM(map, data, 'stretch', 0.001, 'bend', 0.1, 'potential', @L1, 'Number\_of\_intervals', 5); right

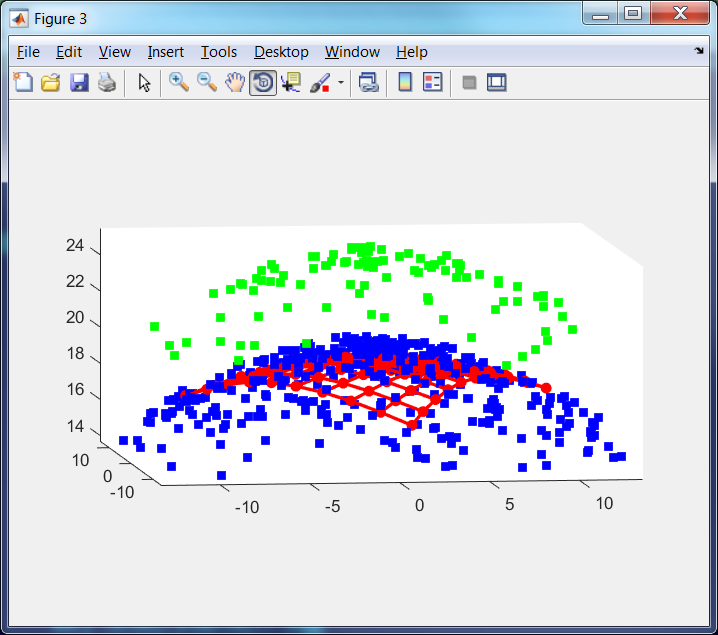
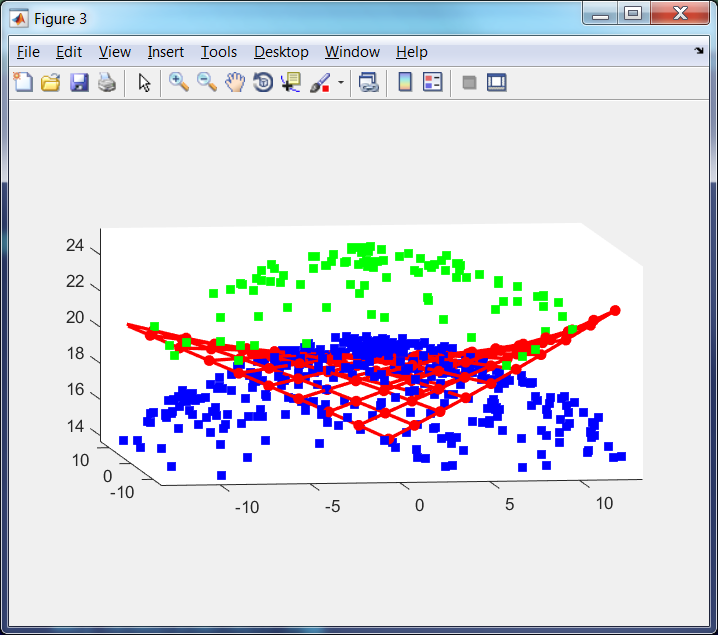


Figure 29. EM(map, data, 'stretch', 0.0001, 'bend', 0.1, 'potential', @LLog, 'Number\_of\_intervals', 5);left

EM(map, data, 'stretch', 0.01, 'bend', 0.1, 'potential', @L2, 'Number\_of\_intervals', 2, 'intshrinkage', 0.5);

We can see that for this data robustness of and based maps are considerably worse. Moreover to avoid of almost collapse of maps we decrease stretch modulo 10 times for based map and 100 times for based map. Both and based maps are far from desirable shape. This mean that for such data the based map with trimming is preferable.

# Appendix A. Formulas derivation

For formula (12):

Formula (22):

For formula (23):

For formula (24):

# Appendix B. Examples of map descriptions

This appendix contains examples of complete descriptions of small maps as example for simple understanding.

## OneDMap

2

3

2

1

4

3

2

1

1

Figure 30. Example of the one dimensional map with four nodes: nodes’ numbers are located above the node, edges’ numbers (blue) are located below edge, ribs are depicted by red lines and its numbers (green) are located below rib.

Let us consider OneDMap with four nodes.

Map creation:

oneMap = OneDMap(4);

Initialization:

init(oneMap, data, 'pci');

or

oneMap.init(data, 'pci');

where data is matrix of data points.

Figure of map in the internal coordinates is presented in Figure 30.

Internal coordinates of nodes are presented in Table 3

Table 3. Internal coordinates of oneMap

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Node number | 1 | 2 | 3 | 4 |
| X coordinate | 1 | 2 | 3 | 4 |

List of edges is presented in Table 4. List of ribs is presented in Table 5.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Table 4. List of edges of oneMap   |  |  |  | | --- | --- | --- | | Edge # | Node 1 | Node 2 | | 1 | 1 | 2 | | 2 | 2 | 3 | | 3 | 3 | 4 | | Table 5. List of ribs of oneMap   |  |  |  |  | | --- | --- | --- | --- | | Rib # | Node 1 | Node 2 | Node 3 | | 1 | 1 | 2 | 3 | | 2 | 2 | 3 | 4 | |

Matrices and for EM (25) are presented below

## rect2DMap

Let us consider rect2DMap with four nodes in each row and column.

13

14

15

16

9

10

11

12

5

6

7

8

3

2

1

4

3

2

1

Figure 31. Example of the rectangular 2D map: nodes’ numbers are located left above the nodes; edges’ numbers (blue) are located below horizontal edge and at right side of vertical edges.

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

Map creation:

rectMap = rect2DMap (4,4);

Initialization:

init(rectMap, data, 'pci');

or

rectMap.init(data, 'pci');

where data is matrix of data points.

Figure of map in the internal coordinates is presented in Figure 31. Numbers of ribs and faces are not presented in Figure 31. Dotted lines depict the faces borders.

Internal coordinates of nodes are presented in Table 6. Lists of edges, ribs and faces are presented in Table 7, Table 8 and Table 9.

Table 6. Internal coordinates of rectMap

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Node number | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 |
| X coordinate | 1 | 1 | 1 | 1 | 2 | 2 | 2 | 2 | 3 | 3 | 3 | 3 | 4 | 4 | 4 | 4 |
| Y coordinate | 1 | 2 | 3 | 4 | 1 | 2 | 3 | 4 | 1 | 2 | 3 | 4 | 1 | 2 | 3 | 4 |

Table 7. List of edges of rectMap

| Edge # | Node 1 | Node 2 |  | Edge # | Node 1 | Node 2 |  | Edge # | Node 1 | Node 2 |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 1 | 2 |  | 9 | 11 | 12 |  | 17 | 5 | 9 |
| 2 | 2 | 3 |  | 10 | 13 | 14 |  | 18 | 6 | 10 |
| 3 | 3 | 4 |  | 11 | 14 | 15 |  | 19 | 7 | 11 |
| 4 | 5 | 6 |  | 12 | 15 | 16 |  | 20 | 8 | 12 |
| 5 | 6 | 7 |  | 13 | 1 | 5 |  | 21 | 9 | 13 |
| 6 | 7 | 8 |  | 14 | 2 | 6 |  | 22 | 10 | 14 |
| 7 | 9 | 10 |  | 15 | 3 | 7 |  | 23 | 11 | 15 |
| 8 | 10 | 11 |  | 16 | 4 | 8 |  | 24 | 12 | 16 |

Table 8. List of ribs of rectMap

| Rib # | Node 1 | Node 2 | Node 3 |  | Rib # | Node 1 | Node 2 | Node 3 |  | Rib # | Node 1 | Node 2 | Node 3 |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 1 | 2 | 3 |  | 7 | 13 | 14 | 15 |  | 12 | 4 | 8 | 12 |
| 2 | 2 | 3 | 4 |  | 8 | 14 | 15 | 16 |  | 13 | 5 | 9 | 13 |
| 3 | 5 | 6 | 7 |  | 9 | 1 | 5 | 9 |  | 14 | 6 | 10 | 14 |
| 4 | 6 | 7 | 8 |  | 10 | 2 | 6 | 10 |  | 15 | 7 | 11 | 15 |
| 5 | 9 | 10 | 11 |  | 11 | 3 | 7 | 11 |  | 16 | 8 | 12 | 16 |
| 6 | 10 | 11 | 12 |  |  |  |  |  |  |  |  |  |  |

Table 9. List of faces of rectMap

| Rib # | Node 1 | Node 2 | Node 3 |  | Rib # | Node 1 | Node 2 | Node 3 |  | Rib # | Node 1 | Node 2 | Node 3 |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 1 | 2 | 5 |  | 7 | 9 | 10 | 13 |  | 13 | 9 | 6 | 10 |
| 2 | 2 | 3 | 6 |  | 8 | 10 | 11 | 14 |  | 14 | 10 | 7 | 11 |
| 3 | 3 | 4 | 7 |  | 9 | 11 | 12 | 15 |  | 15 | 11 | 8 | 12 |
| 4 | 5 | 6 | 9 |  | 10 | 5 | 2 | 6 |  | 16 | 13 | 10 | 14 |
| 5 | 6 | 7 | 10 |  | 11 | 6 | 3 | 7 |  | 17 | 14 | 11 | 15 |
| 6 | 7 | 8 | 11 |  | 12 | 7 | 4 | 8 |  | 18 | 15 | 12 | 16 |

Matrices and for EM (25) are presented below

## tri2DMap

Let us consider tri2DMap with four rows four nodes in each odd row and three nodes in each even row.

13

14

9

10

11

12

5

6

7

8

3

2

1

4

3

2

1

Figure 32. Example of the triangular 2D map: nodes’ numbers are located above the nodes; edges’ numbers (blue) are located near edges and faces’ numbers (green) are located in the centres of triangles.

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

10

25

26

27

28

1

3

2

4

5

6

7

8

9

10

11

12

13

14

15

Map creation:

triMap = tri2DMap (4,4);

Initialization:

init(triMap, data, ‘pci’);

or

triMap.init(data, ‘pci’);

where data is matrix of data points.

Figure of map in the internal coordinates is presented in Figure 32. Numbers of ribs are not presented in Figure 32.

Internal coordinates of nodes are presented in Table 10. Lists of edges, ribs and faces are presented in Table 11, Table 12 and Table 13.

Table 10. Internal coordinates of triMap

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Node number | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
| X coordinate | 1.00 | 2.00 | 3.00 | 4.00 | 1.50 | 2.50 | 3.50 | 1.00 | 2.00 | 3.00 | 4.00 | 1.50 | 2.50 | 3.50 |
| Y coordinate | 0.00 | 0.00 | 0.00 | 0.00 | 0.87 | 0.87 | 0.87 | 1.73 | 1.73 | 1.73 | 1.73 | 2.60 | 2.60 | 2.60 |

Table 11. List of edges of triMap

| Edge # | Node 1 | Node 2 |  | Edge # | Node 1 | Node 2 |  | Edge # | Node 1 | Node 2 |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 1 | 2 |  | 11 | 1 | 5 |  | 20 | 2 | 5 |
| 2 | 2 | 3 |  | 12 | 2 | 6 |  | 21 | 3 | 6 |
| 3 | 3 | 4 |  | 13 | 3 | 7 |  | 22 | 4 | 7 |
| 4 | 8 | 9 |  | 14 | 8 | 12 |  | 23 | 9 | 12 |
| 5 | 9 | 10 |  | 15 | 9 | 13 |  | 24 | 10 | 13 |
| 6 | 10 | 11 |  | 16 | 10 | 14 |  | 25 | 11 | 14 |
| 7 | 5 | 6 |  | 17 | 5 | 9 |  | 26 | 5 | 8 |
| 8 | 6 | 7 |  | 18 | 6 | 10 |  | 27 | 6 | 9 |
| 9 | 12 | 13 |  | 19 | 7 | 11 |  | 28 | 7 | 10 |
| 10 | 13 | 14 |  |  |  |  |  |  |  |  |

Table 12. List of ribs of triMap

| Rib # | Node 1 | Node 2 | Node 3 |  | Rib # | Node 1 | Node 2 | Node 3 |  | Rib # | Node 1 | Node 2 | Node 3 |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 1 | 2 | 3 |  | 7 | 1 | 5 | 9 |  | 12 | 2 | 5 | 8 |
| 2 | 2 | 3 | 4 |  | 8 | 2 | 6 | 10 |  | 13 | 3 | 6 | 9 |
| 3 | 8 | 9 | 10 |  | 9 | 3 | 7 | 11 |  | 14 | 4 | 7 | 10 |
| 4 | 9 | 10 | 11 |  | 10 | 5 | 9 | 13 |  | 15 | 6 | 9 | 12 |
| 5 | 5 | 6 | 7 |  | 11 | 6 | 10 | 14 |  | 16 | 7 | 10 | 13 |
| 6 | 12 | 13 | 14 |  |  |  |  |  |  |  |  |  |  |

Table 13. List of faces of triMap

| Rib # | Node 1 | Node 2 | Node 3 |  | Rib # | Node 1 | Node 2 | Node 3 |  | Rib # | Node 1 | Node 2 | Node 3 |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 1 | 2 | 5 |  | 6 | 10 | 11 | 14 |  | 11 | 9 | 13 | 12 |
| 2 | 2 | 3 | 6 |  | 7 | 5 | 6 | 9 |  | 12 | 10 | 14 | 13 |
| 3 | 3 | 4 | 7 |  | 8 | 6 | 7 | 10 |  | 13 | 5 | 9 | 8 |
| 4 | 8 | 9 | 12 |  | 9 | 2 | 6 | 5 |  | 14 | 6 | 10 | 9 |
| 5 | 9 | 10 | 13 |  | 10 | 3 | 7 | 6 |  | 15 | 7 | 11 | 10 |

Matrices and for EM (25) are presented below

# References

1. 5. Gorban, A.N., Zinovyev, A.Y. and Wunsch, D.C., 2003, July. Application of the method of elastic maps in analysis of genetic texts. In *Proceedings of the International Joint Conference on Neural Networks, 2003.* (Vol. 3, pp. 1826-1831). IEEE. <https://doi.org/10.1109/ijcnn.2003.1223685>
2. 6. Gorban, A. and Zinovyev, A.Y., 2001. Visualization of data by method of elastic maps and its applications in genomics, economics and sociology. <http://www.ihes.fr/~/zinovyev/papers/M01-36.pdf>
3. 13. Gorban, A.N. and Zinovyev, A.Y., 2008. Elastic maps and nets for approximating principal manifolds and their application to microarray data visualization. In Principal manifolds for data visualization and dimension reduction (pp. 96-130). Springer, Berlin, Heidelberg. <https://doi.org/10.1007/978-3-540-73750-6_4>
4. 14. Gorban, A.N. and Zinovyev, A.Y., 2001. Method of elastic maps and its applications in data visualization and data modeling. International Journal of Computing Anticipatory Systems, CHAOS, 12, pp.353-369.
5. 12. Shaban, H. and Tavoularis, S., 2014. Identification of flow regime in vertical upward air–water pipe flow using differential pressure signals and elastic maps. International Journal of Multiphase Flow, 61, pp.62-72. <https://doi.org/10.1016/j.ijmultiphaseflow.2014.01.009>
6. Gorban, A.N., Mirkes, E.M. and Zinovyev, A., 2018, July. Data analysis with arbitrary error measures approximated by piece-wise quadratic PQSQ functions. In *2018 International Joint Conference on Neural Networks (IJCNN)* (pp. 1-8). IEEE. <https://doi.org/10.1109/ijcnn.2018.8489568>
7. 1. Akinduko, Ayodeji A., Evgeny M. Mirkes, and Alexander N. Gorban. "SOM: Stochastic initialization versus principal components." *Information Sciences*(2015), <http://dx.doi.org/10.1016/j.ins.2015.10.013>
8. 2. Dobson, Annette J., and Adrian Barnett. *An introduction to generalized linear models*. CRC press, 2008. <https://doi.org/10.1002/pst.486>
9. Kernel (statistics) <https://en.wikipedia.org/wiki/Kernel_(statistics)>
10. 2. Dobson, Annette J., and Adrian Barnett. *An introduction to generalized linear models*. CRC press, 2008.
11. 3. PCA
12. 4. PCA
13. 7. <https://en.wikipedia.org/wiki/Distance#Distances_between_sets_and_between_a_point_and_a_set>
14. 8. <https://en.wikipedia.org/wiki/Metric_space#Distance_between_points_and_sets;_Hausdorff_distance_and_Gromov_metric>
15. 9. Gorban & A Rossiev
16. 10. Brand M. (2002) Incremental Singular Value Decomposition of Uncertain Data with Missing Values. In: Heyden A., Sparr G., Nielsen M., Johansen P. (eds) Computer Vision — ECCV 2002. ECCV 2002. Lecture Notes in Computer Science, vol 2350. Springer, Berlin, Heidelberg
17. 11.Kurucz, M., Benczúr, A.A. and Csalogány, K., 2007, August. Methods for large scale SVD with missing values. In *Proceedings of KDD cup and workshop* (Vol. 12, pp. 31-38).