Elastic maps

Contents

[1 Introduction 2](#_Toc516129139)

[2 Map geometry 2](#_Toc516129140)

[3 Projection data points onto map 3](#_Toc516129141)

[3.1 Projection of data points to node 4](#_Toc516129142)

[3.2 Projection of data onto nearest edge 4](#_Toc516129143)

[3.3 Projection of data onto nearest face 5](#_Toc516129144)

[4 Fraction of variance unexplained 7](#_Toc516129145)

[5 EM (Elastic map) 8](#_Toc516129146)

[5.1 Elastic energy 8](#_Toc516129147)

[5.2 Function description 11](#_Toc516129148)

[6 SOM 11](#_Toc516129149)

[7 Appendix A. Formulas derivation. 11](#_Toc516129150)

[8 Appendix B. Examples of map descriptions 13](#_Toc516129151)

[8.1 OneDMap 13](#_Toc516129152)

[8.2 rect2DMap 14](#_Toc516129153)

[8.3 tri2DMap 15](#_Toc516129154)

[9 References 17](#_Toc516129155)

# Introduction

Presented package ElMap is developed to form nonlinear manifolds by two techniques: Self-Organizing Map (SOM) and Elastic Map (EM). Main purpose is EM implementation and SOM is added as auxiliary feature.

This document contains theoretical description, user guide and technical description.

# 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 and SOM 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 . 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).

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

***Dimension*** is number of internal coordinates.

***Internal coordinates*** is the set 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 SOM or 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.

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 defines an initial mapped coordinates. In accordance of results of paper [1] 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.

***Distance*** is method to calculate distances between two nodes in the internal coordinates. This method is useful for SOM fitting procedure.

***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.

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.

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

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

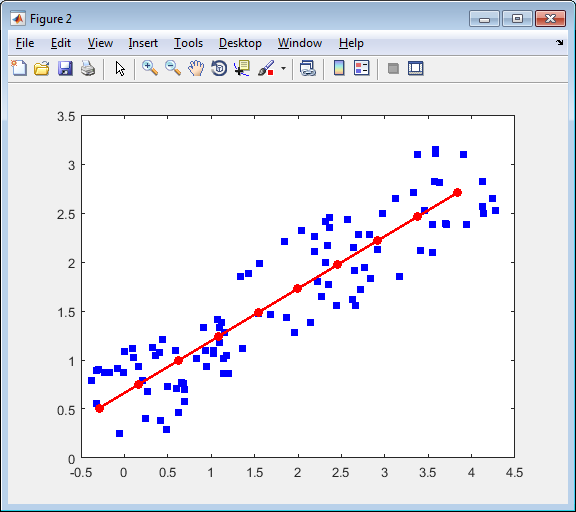
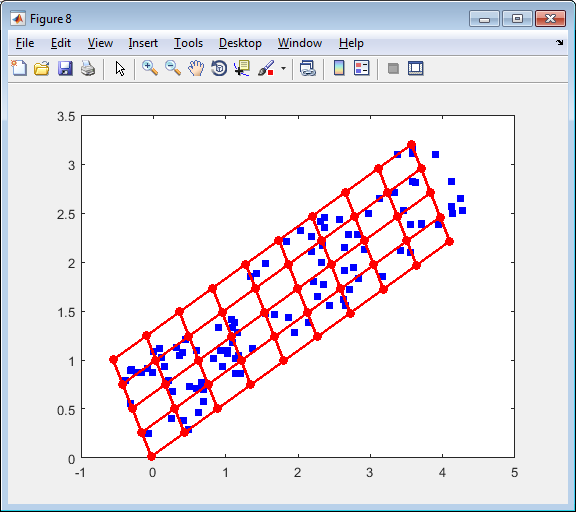
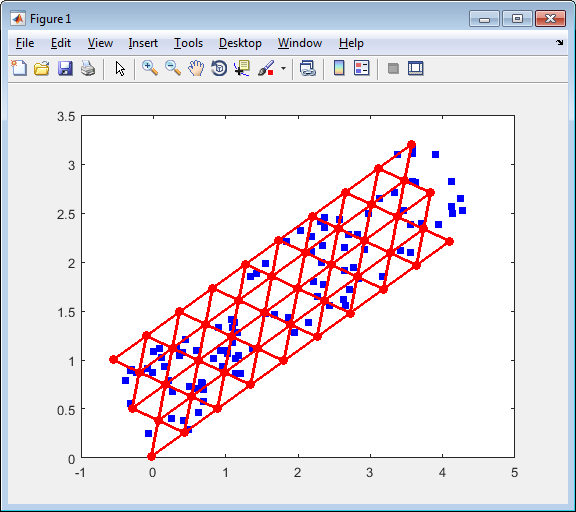
***putMapped*** is method to put fitted mapped coordinates to the map.

# 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 is 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.

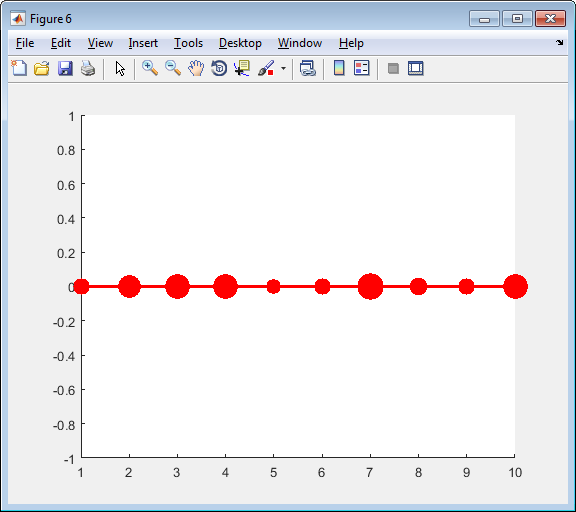
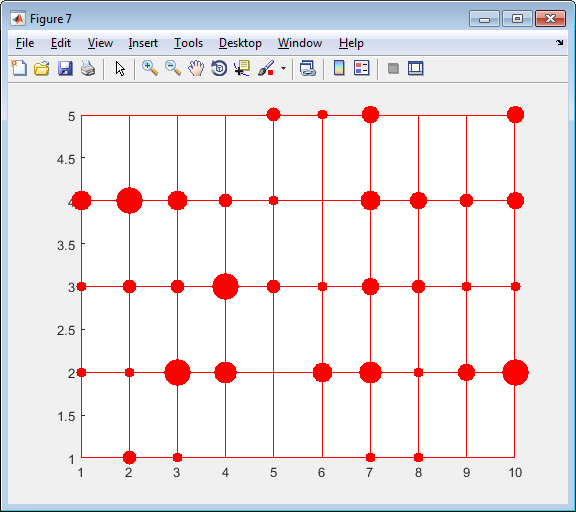
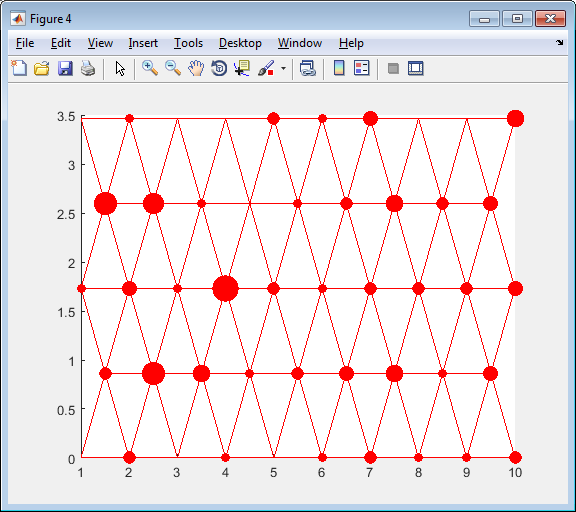
  

Figure . Examples of data points projection to the nearest node: top row contains graphs in the original space and bottom row contains corresponding graphs in the internal coordinates; left column presents the one dimensional map, central column presents rectangular 2D map and right column presents the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

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

Figure

a

b

c

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

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

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:

or

or

|  |  |
| --- | --- |
|  | (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:

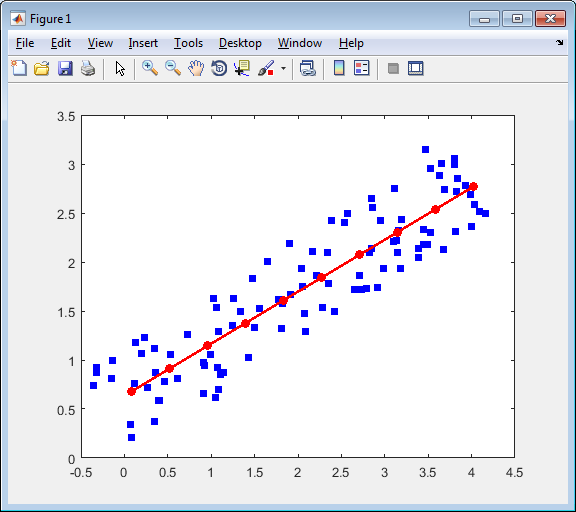
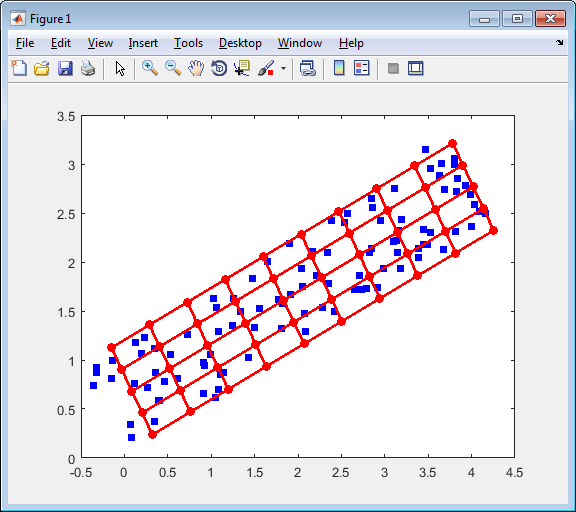
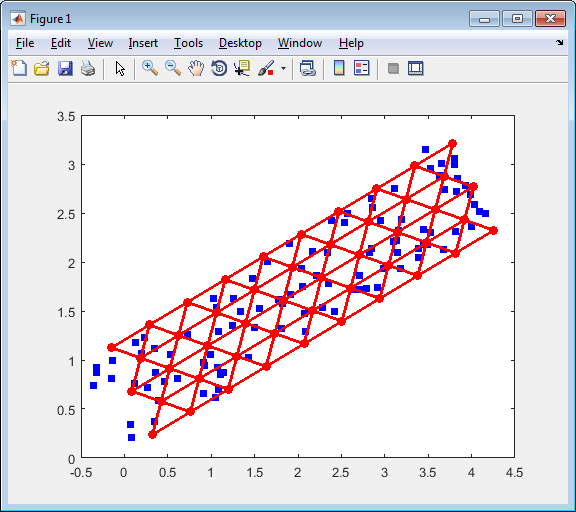
|  |  |
| --- | --- |
|  | (3) |

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).

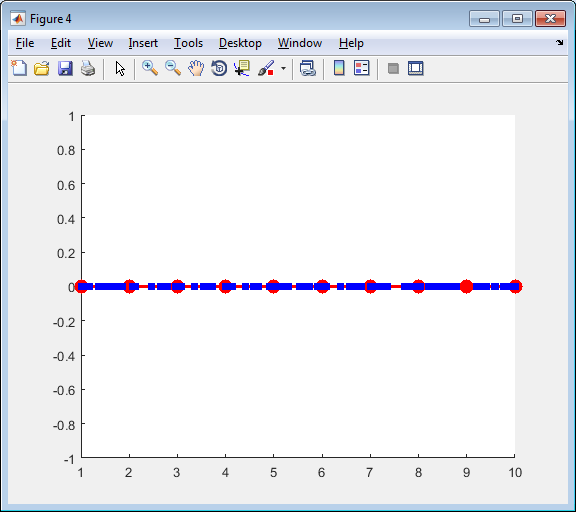
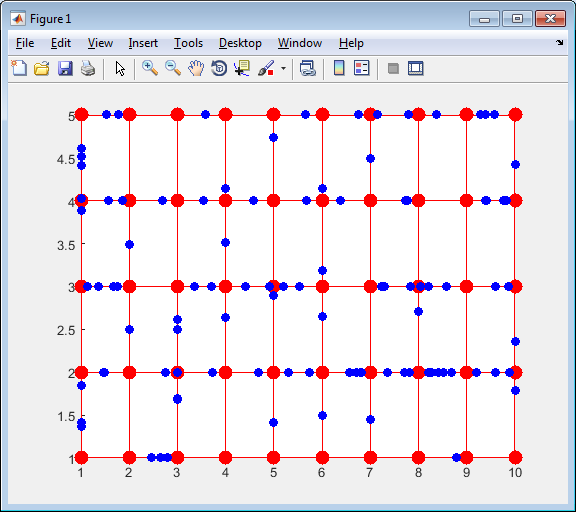
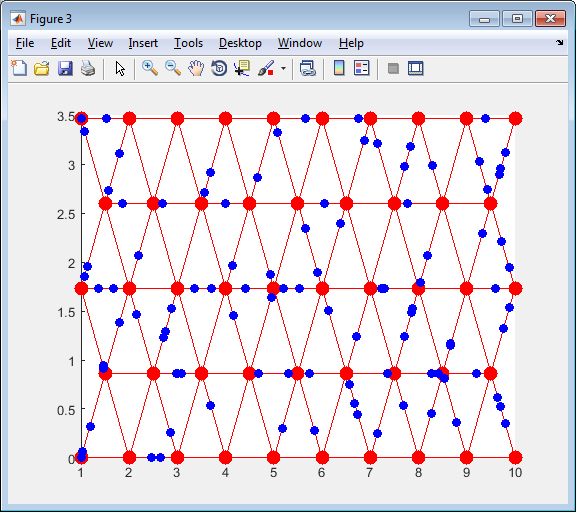
  

Figure . Examples of data points projection to the nearest node: top row contains graphs in the original space and bottom row contains corresponding graphs in the internal coordinates; left column presents the one dimensional map, central column presents rectangular 2D map and right column presents the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

## Projection of data onto nearest 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

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

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 distance. To do it differentiate 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) |

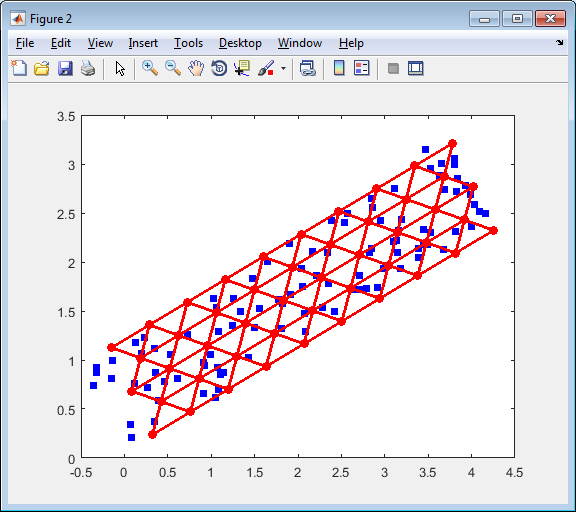
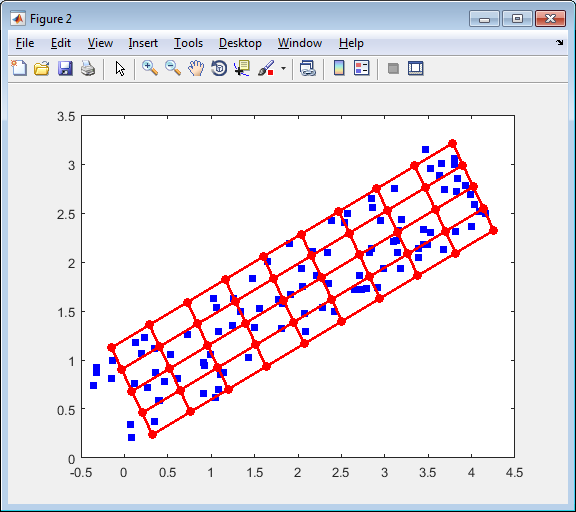
and

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

Now we can calculate the distance:

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

Examples of data points projection onto nearest face are presented in Figure 7.



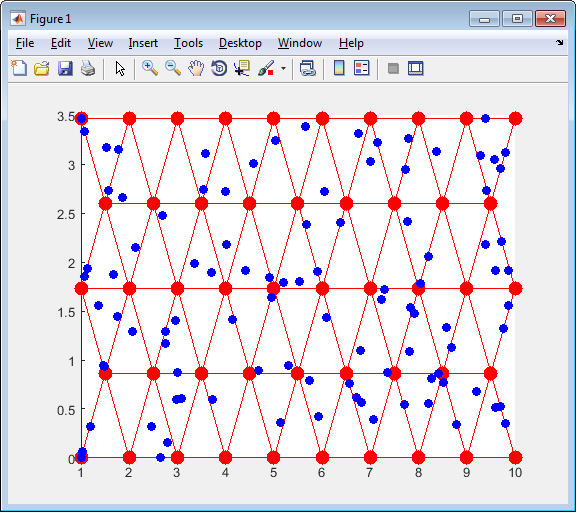
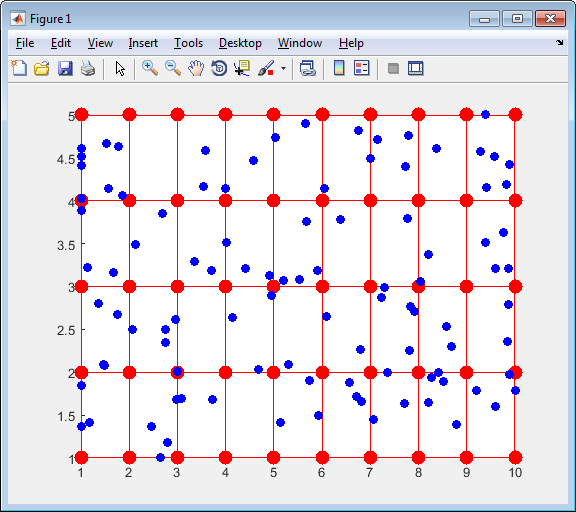


Figure . Examples of data points projection to the nearest face: top row contains graphs in the original space and bottom row contains corresponding graphs in the internal coordinates; left column presents the rectangular 2D map and right column presents the triangular 2D map.

# Fraction of variance unexplained

Fraction of Variance Unexplained (FVU) [2] 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 [2]. 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 [3, 4]. 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 and SOM.

# EM (Elastic map)

EM is function which fit parameters of elastic map. Elastic map is introduced in paper [5] and detailed description is contained in [6]. 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 nodes. Edges form ribs , where are number of nodes.

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 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 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:

## Function description

function EM( map, data, stretch, bend )

%EM is function to fit map to data

% Syntax

% EM( map, data )

% EM( map, data, type )

% EM( map, data, stretch, bend )

%

% map is an object of MapGeometry class or subclass of this class.

% data is n-by-m matrix which is the data set to fit. Each row contains

% m coordinates of one data point.

% type is one of the following strings:

% 'hard' is hard map with stretch = 1 and bend = 1

% 'medium' is more flexible map with stretch = 0.7 and bend = 0.7

% 'soft' is soft map with stretch = 0.5 and bend = 0.5

% For syntax EM( map, data ) the 'medium' is used.

% stretch is a positive numeric value which is represent the value of

% stretching modulo or a function with syntax

% val = stretch( iteration )

% where iteration is number of iteration and val is the

% stretch1ng modulo to use on specified iteration. Iterations

% are numerated from 1.

% bend is a positive numeric value which is represent the value of

% bending modulo or a function with syntax

% val = bend( iteration )

% where iteration is number of iteration and val is the bending

% modulo to use on specified iteration. Iterations are

% numerated from 1.

# SOM

# 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

Let us consider OneDMap with four nodes.

2

3

2

1

4

3

2

1

1

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

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 8.

Internal coordinates of nodes are presented in Table 1

Table . Internal coordinates of oneMap

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

List of edges is presented in Table 2. List of ribs is presented in Table 3.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Table . List of edges of oneMap   |  |  |  | | --- | --- | --- | | Edge # | Node 1 | Node 2 | | 1 | 1 | 2 | | 2 | 2 | 3 | | 3 | 3 | 4 | | Table . 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 . Example of the rectangular 2D map: nodes’ numbers are located left above the nodes; edges’ numbers 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 9. Numbers of ribs and faces are not presented in Figure 9. Dotted lines depict the faces borders.

Internal coordinates of nodes are presented in Table 4. Lists of edges, ribs and faces are presented in Table 5, Table 6 and Table 7.

Table . 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 . 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 . 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 . 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 . Example of the triangular 2D map: nodes’ numbers are located above the nodes; edges’ numbers are located near edges and faces’ numbers 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 10. Numbers of ribs are not presented in Figure 10.

Internal coordinates of nodes are presented in Table 8. Lists of edges, ribs and faces are presented in Table 9, Table 10 and Table 11.

Table . 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 . 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 . 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 . 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. 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>
2. Dobson, Annette J., and Adrian Barnett. *An introduction to generalized linear models*. CRC press, 2008.
3. PCA
4. PCA
5. EM first
6. EM detailed