CLASSE – Synthetic Classification for Industrial Supervision

VISRED

Data Visualization by Space Reduction

Version 3.0

User's Guide

António Dourado
Edgar Ferreira
Paulo Barbeiro
Sara Silva

FCTUC

October 10, 2008

UNIAO EUROPEIA
FSE/FEDER

Supported by FCP Project POSC/EIA/58162/2004

TABLE OF CONTENTS

Page

Authors:
Edgar da Silva Ferreira – edgar.bajouca@gmail.com
Paulo Filipe Domingues Barbeiro – paulobarbeiro@gmail.com
Sara Guilherme Oliveira da Silva – sara@dei.uc.pt

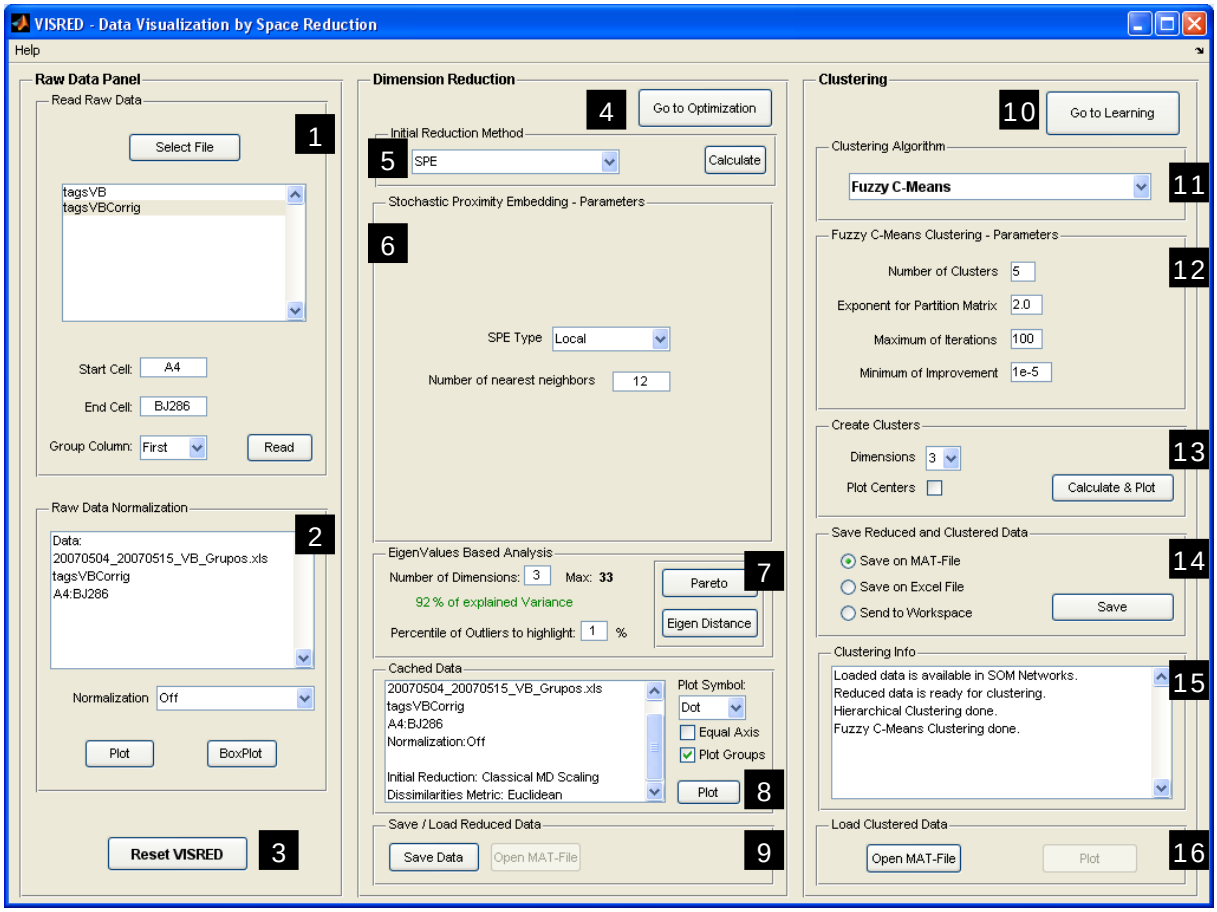
Supervisor:
António Dourado Correia (PhD) – dourado@dei.uc.pt

SOFWARE REQUIREMENTS

Software requirements for VisRed 3.0:

- MATLAB 7.5.0 (R2007b) or later
- Statistics Toolbox 6.1 or later
- Optimization Toolbox 3.1.2 or later
- Fuzzy Logic Toolbox 2.2.6 or later
- Neural Network Toolbox 5.1 or later
- Genetic Algorithm and Direct Search Toolbox 2.2 or later
- Curve Fitting Toolbox 1.2 or later
- Spline Toolbox 3.3.3 or later
- Mapping Toolbox 2.6 or later

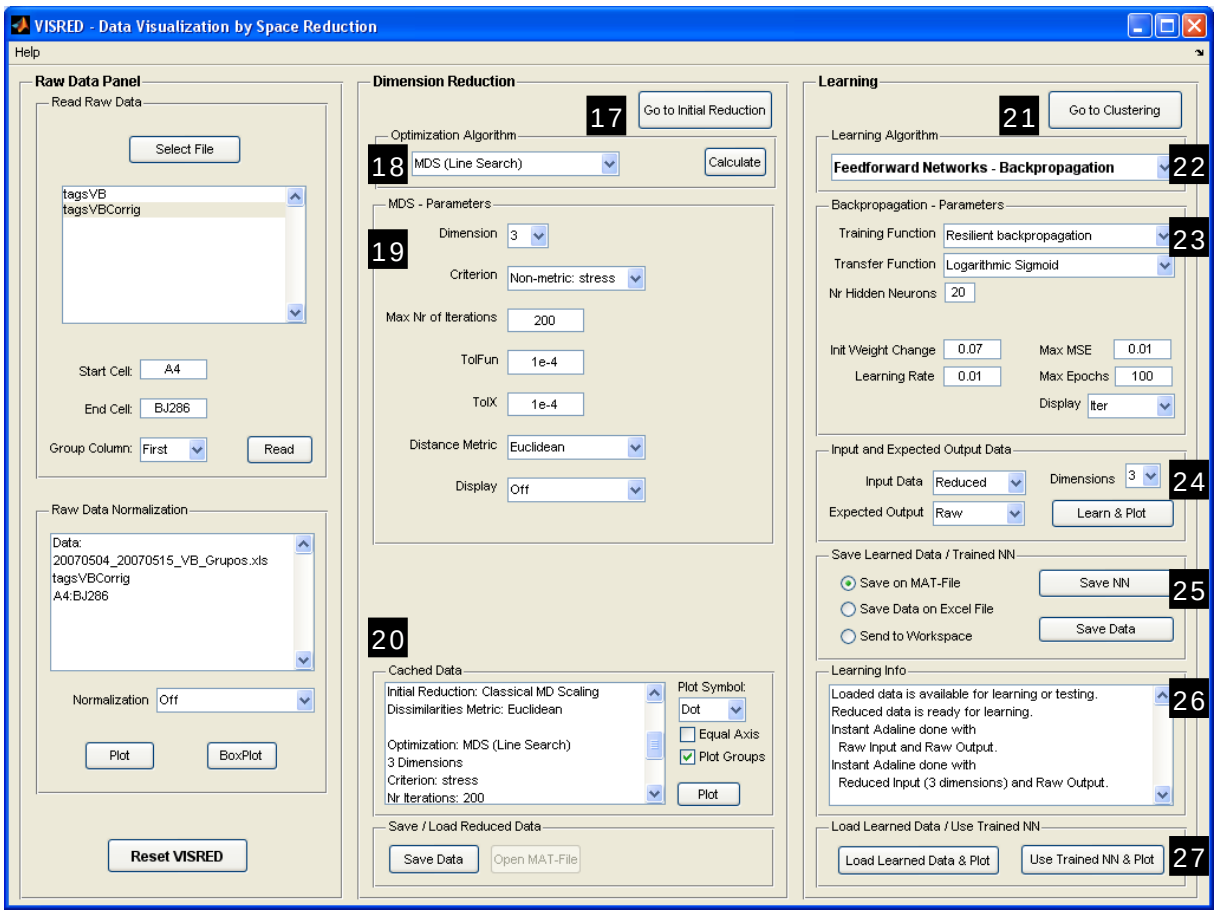
OVERVIEW



Description:

1. *Read Raw Data*: Reads data from an Excel file
2. *Raw Data Normalization*: Provides normalization and visualization of imported data
3. *Reset VISRED*: Restarts all components and variables
4. *Go to Optimization*: Hides the initial reduction panel and shows the dimensionality reduction optimization methods
5. *Initial Reduction Method*: Methods to perform initial dimensionality reduction
6. *Classical Multidimensional Scaling / Principal Component Analysis / Random Dimensionality Reduction / Non Linear PCA / Isomap / Stochastic Proximity Embedding / Factor Analysis / Stochastic Neighbour Embedding / Sammon Mapping / Laplacian Eigenmaps – Parameters*: Setting of initial reduction parameters
7. *EigenValues Based Analysis*: Several visualization methods based on existing Eigenvalues

8. *Cached Data*: Displays information on data currently being reduced, and provides its visualization
9. *Save / Load Reduced Data*: Saves a reduced matrix / Loads a file containing a previously saved reduced matrix
10. *Go to Learning*: Hides the clustering panel and shows the learning methods
11. *Clustering Algorithm*: Methods to perform data clustering
12. *Hierarchical Clustering / K-Means Clustering / Fuzzy C-Means Clustering / Fuzzy Subtractive Clustering / Dignet Clustering / SOM Network – Parameters*: Setting of clustering parameters
13. *Create Clusters*: Computes and plots clusters
14. *Save Reduced and Clustered Data*: Saves the clustering results to an Excel or MAT file, or to Matlab workspace
15. *Clustering Info*: Displays clustering related information
16. *Load Clustered Data*: Loads and plots clustered data



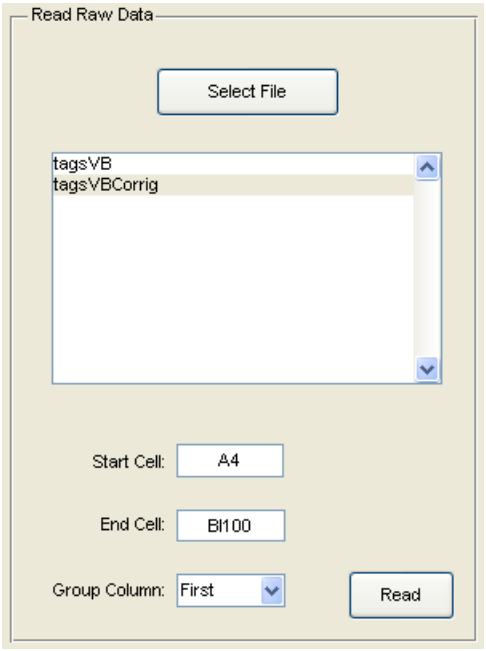
17. *Go to Initial Reduction*: Hides the optimization methods and shows the initial dimensionality reduction panel
18. *Optimization Algorithm*: Methods to perform dimensionality reduction optimization
19. *MDS / Genetic Algorithm / Simulated Annealing / Non Linear PCA – Parameters*: Setting of dimensionality reduction optimization parameters
20. *Cached Data*: Displays information on data previously reduced and currently being optimized, and provides its visualization
21. *Go to Clustering*: Hides the learning panel and shows the clustering methods
22. *Learning Algorithm*: Methods to perform data learning
23. *Adaline / Backpropagation / Focused Time-Delay / Distributed Time-Delay / NARX / Layer-Recurrent / Radial Basis / Probabilistic / Generalized Regression / Support Vector Machines – Parameters*: Setting of learning parameters
24. *Input and Expected Output Data*: Selection of input and expected output data, computes and plots classification
25. *Save Learned Data / Trained NN*: Saves the learning results to an Excel or MAT file, or to Matlab workspace
26. *Learning Info*: Displays learning related information
27. *Load Learned Data / Use Trained NN*: Loads learned data and plots classification, loads trained neural network to be used in a different data set

Note: plots are made on separate windows.

DATA IMPORT

Excel File Reading

Data that the user intends to subject to dimension reduction must be read from a sheet in an Excel file. If this is the case, the first step to follow in the application is to import the data, in the *Read Raw Data* panel. For such, the user must choose a file by pressing the *Select File* button. After this, all Excel sheets in the file are listed, and the user must choose which one contains the data to analyze.



By default, values that appear in the cells refer to data with 47 variables, the study target in this project. The *Start Cell* (top left corner of data) corresponds to the variable names line (if it exists, else it is the first row of data), and to the first column of the event labels (if there are no event labels, first column of data). The *End Cell* is the bottom right cell with data. The *Group Column* is a column defined by the user, containing a numeric classification that will allow plotting the data in colored groups, and applying learning methods. Rows with the same number belong to same group, or same class. This column must appear after all event labels, before or after the rest of the data (i.e., it must be the first or last column of data). To read the data, the user must press the *Read* button.

Header1 1 (1)	Header 2 (1)	…	Header m (1)	Variable1	Variable2	…	VariableM
Header1 1 (2)	Header 2 (2)	…	Header m (2)				
…	…	…	…				
Header 1 (N)	Header 2 (N)	…	Header m (N)				
1st eventLabel_1	1st eventLabel_2	…	1st eventLabel_m	11	12	13	14
2nd eventLabel_1	2nd eventLabel_2	…	2nd eventLabel_m	21	22	23	24
3rd eventLabel_1	3rd eventLabel_2	…	3rd eventLabel_m	31	32	33	34
4th eventLabel_1	4th eventLabel_2	…	4th eventLabel_m	41	42	43	44
…	…	…	…	51	52	53	54
…	…	…	…	61	62	63	64
nth eventLabel_1	nth eventLabel_2	…	nth eventLabel_m	71	72	73	74

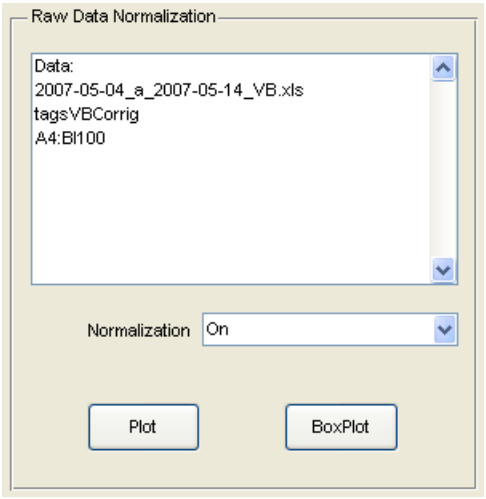
Multiple labels for headers, variables and events are accepted. Also, if there are no labels at all (or labels are incomplete), automatic ones will be generated. If there are several rows for *Headers* and *Variables* labels, they will all be concatenated to just one row. The last column of labels must contain text, but the other columns of labels don't have this restriction, as the application is able to convert them to text. If data is incomplete (numeric cells without values), a warning message will appear and the loading process will be halted.

NORMALIZATION

Normalization of Data

Generally speaking, collected data for data-mining processes refers to variables with different measurement units (or even with no unit). The absence of homogeneity on the data scale may induce errors on the analyzing process. To prevent this from happening, normalization is required, although some important information may be lost during this process.

The normalization method is chosen in the *Raw Data Normalization* panel.



- *Normalization* can be one of four options:

  - Off
    Data is used as read.

  - Mean Value Only
    Calculates and subtracts from each variable their mean value. By doing so, the mean of each variable becomes zero.

  - Standard Deviation Only
    This method sets the maximum standard deviation (STD) of each variable to be one. The regular STD based normalization set all STDs to 1, which in case of constant (or near constant) variables results in a division by zero (or near zero). When this happens, those variables are amplified to ∞, becoming much more influent than the others, and unbalancing the proportion between data. By allowing variables to have STD≤1, this situation is corrected – those who have already a low STD remain unchanged.

  - On
    This option is equivalent to '*Mean Value Only*' and '*Standard Deviation Only*', and the resulting data has mean = 0, and STD≤1.

Plot and Boxplot Buttons

Boxplot button – Draws a box and whisker plot with one box for each variable.

Plot button – Plots read data, with events on X axis and variable values on Y axis. Clicking on the data represented on the plot will tell the user which variable and event correspond to the selected point.

This point selection option to see information about the data is available throughout most of the application, and can be enabled or disabled by pressing the *data cursor button* anytime:

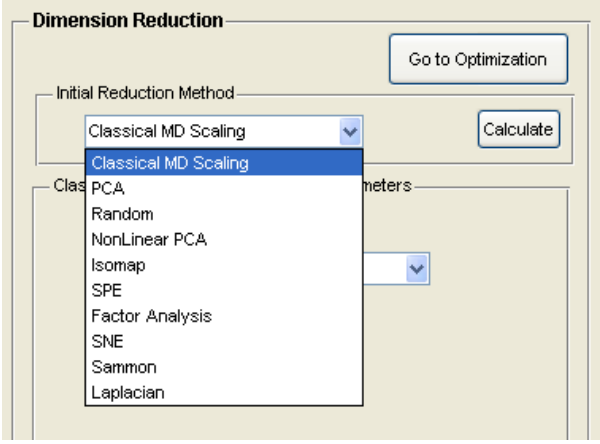
When active, the mouse pointer will became a small white cross. Keyboard arrows can be used to cycle along the data points, and multiple tags will appear by pressing *Alt* key while selecting a new point with the mouse.

DIMENSIONS REDUCTION

Dimensions Reduction Method

Note: Before starting calculus it may be useful to activate CPU monitoring since, depending on the amount of information, it may take from 10 seconds up to 2 days for calculus to be complete. CPU monitoring can be activated by pressing *Ctrl+Alt+Del*, and run in the background by minimizing the console that appears.

The method for the initial dimensionality reduction can be selected on the *Dimension Reduction Method* panel, *Initial Reduction Method* drop-down list.



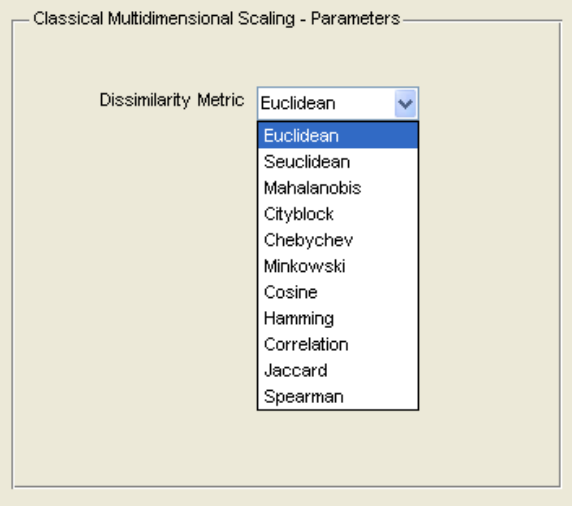
- Classical Multidimensional Scaling (CMD)

  This method maps the high-dimensional data to a low-dimensional representation while attempting to preserve the pairwise distances between the data points. It takes an interpoint distance matrix and returns a configuration matrix in a lower dimensional space. It also returns an Eigenvalues vector, where each value corresponds to the representativeness of each new dimension. The quality of the mapping is expressed in the stress function, a measure of the error between the pairwise distances in the low-dimensional and high-dimensional representation of the data.

  (For further information, see *cmdscale* in Matlab Help, from Statistics Toolbox)

  There is one parameter associated with the CMD method, which is the *Dissimilarity Metric*. Dissimilarity is a measurement of disparity between multidimensional points. In this application there are several different methods for calculating distances. In most of our experiments the best results were obtained with the Euclidean, SEuclidean, CityBlock and Correlation metrics.

  (For further information, see *pdist* in Matlab Help, from Statistics Toolbox)



- Principal Component Analysis (PCA)

  Principal Component Analysis is a statistics based method to reduce the number of dimensions of a data matrix. After calculation, a window appears with the results on 3 plots. In the first plot the scales of the axes are adjusted; in the second plot the 'equal axes' mode is on; in the third plot there is a projection of the original variables, in order to allow a visualization of the contribution of each of the original variables for each new principal component.

  (For further information, see *princomp* in Matlab Help)

- Random

  This method creates a random matrix based on raw data statistics.

- Non Linear Principal Component Analysis (nlPCA)

  This method calls a Graphic User Interface (GUI) for adjusting the parameters and calculating the several components of the nlPCA. Since this is also an optimization algorithm, variables and operation flow are explained on the corresponding later section.

- Isomap

  This method is an extension of multidimensional scaling. Instead of using Euclidean or other common metrics, it calculates pairwise geodesic distances on a weighted graph, where each point is connected to all its neighbors. Each geodesic distance is calculated as the sum of edge weights along the shortest path between two nodes.

  There is a parameter associated with the Isomap method, which is the *Number of nearest neighbors* considered to calculate distances.

  (For further information, see *compute_mapping* in Matlab Help, from Dimensionality Reduction Toolbox by L.J.P. van der Maaten)

- Stochastic Proximity Embedding (SPE)

  This method minimizes the MDS raw stress function. It differs from MDS in the computationally-efficient rule it uses to update the current estimate of the low-dimensional data representation. SPE can also be applied in order to retain only local distances in a neighborhood graph, leading to a behavior that is comparable to Isomap.

  In this application, both *Global* and *Local* types of SPE can be used, the latter requiring a parameter specifying the *Number of nearest neighbors* considered to calculate distances.

  (For further information, see *compute_mapping* in Matlab Help, from Dimensionality Reduction Toolbox by L.J.P. van der Maaten)

- Factor Analysis

  This method models observed variables as linear combinations of fewer unobserved variables called factors, plus error terms. Factor Analysis is related to PCA, and becomes essentially the same if the errors are all assumed to have the same variance.

  (For further information, see *compute_mapping* in Matlab Help, from Dimensionality Reduction Toolbox by L.J.P. van der Maaten)

- Stochastic Neighbor Embedding (SNE)

  This method, also related to MDS, converts each high-dimensional similarity into the probability that one data point will pick the other data point as its neighbor. It provides a low-dimensional data representation that preserves mainly local relations, as nearby points contribute more than points farther away.

  There is a parameter associated with the SNE method, which is the *Perplexity*, a measure related to entropy that sets the number of effective nearest neighbors.

  (For further information, see *compute_mapping* in Matlab Help, from Dimensionality Reduction Toolbox by L.J.P. van der Maaten)

- Sammon Mapping

  This method is a variant of MDS where more emphasis is put on retaining distances that were originally small.

  (For further information, see *compute_mapping* in Matlab Help, from Dimensionality Reduction Toolbox by L.J.P. van der Maaten)

- Laplacian Eigenmaps

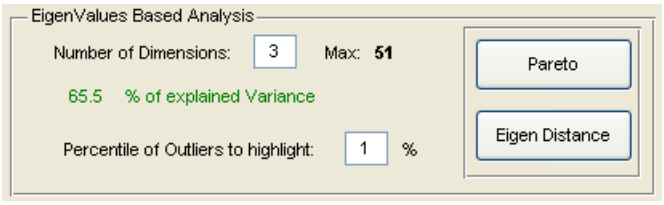
This method finds a low-dimensional data representation by preserving local properties, based on the pairwise distances between near neighbors. Using spectral graph theory, the problem of finding the low-dimensional data representation can be defined as a generalized Eigenvector problem.

There are two parameters associated with the Laplacian method, which are the *Number of nearest neighbors* and the *Sigma* value used to solve the Eigenvector problem.

(For further information, see *compute_mapping* in Matlab Help, from Dimensionality Reduction Toolbox by L.J.P. van der Maaten)

Eigen Values Based Analysis

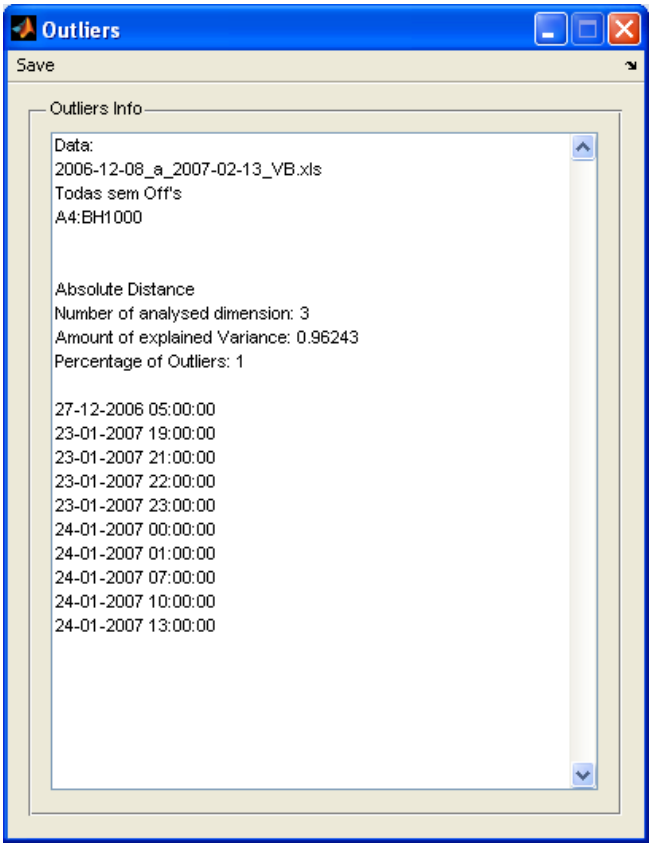
This panel allows several operations based on the existence of Eigenvalues:



The *Pareto* button draws two plots on a window: a bar graph for each of the first five Eigenvalues, with a cumulative curve for them, and a curve for all Eigenvalues. The datacursor mode (explained on the *Plot and Boxplot Buttons* section) provides the principal component index and value. It is advisable to watch carefully for mean negative values of Eigenvalues, since that is an indicator of poor quality of the performed reduction.

The *Eigen Distance* calculates the absolute distance from points to origin, based on the principle that distances along each of the dimensions are proportional to the amount of variance that those dimensions represent. For this calculation, '*Percentile of Outliers to highlight*' will set the percentage of existing further points to detect, and '*Number of Dimensions*' defines the number of dimensions that will be analysed.

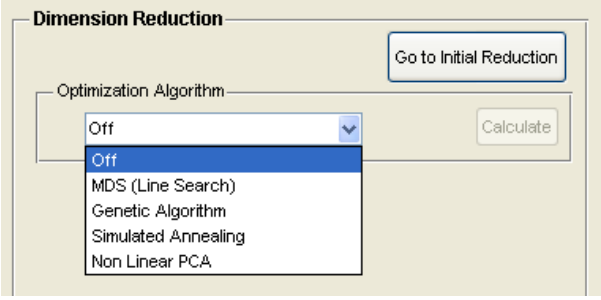
The output is presented in a window as depicted below, and can be saved to a file. The amount of explained variance is normalized between 0 and 1.


Outliers
Save
Outliers Info
Data:
2006-12-08_a_2007-02-13_VB.xls
Todas sem Off's
A4:BH1000
Absolute Distance
Number of analysed dimension: 3
Amount of explained Variance: 0.96243
Percentage of Outliers: 1
27-12-2006 05:00:00
23-01-2007 19:00:00
23-01-2007 21:00:00
23-01-2007 22:00:00
23-01-2007 23:00:00
24-01-2007 00:00:00
24-01-2007 01:00:00
24-01-2007 07:00:00
24-01-2007 10:00:00
24-01-2007 13:00:00

OPTIMIZATION ALGORITHMS

Several optimization techniques can be used to improve the initial dimensionality reduction results. These methods are available on the Dimension Reduction panel, *Optimization Algorithm* drop-down list, and use data from the initial reduction as a starting reference, in order to improve defined criterion.

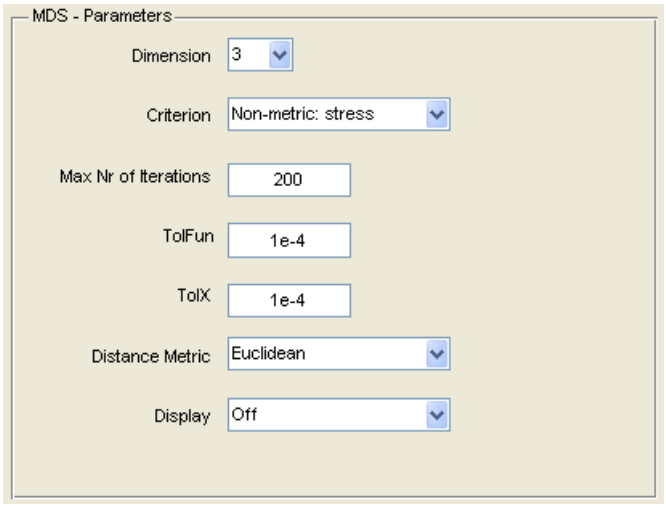
In every optimization algorithm, the ideal configuration of points would be the one whose distance between points is the same as a reference one. This reference configuration is calculated when the dimensionality reduction is made, with the metric and normalization selected at the time.



Optimization methods can be performed sequentially, allowing many criterion improvements to the same data sample.

Multi Dimensional Scaling (MDS)

Multidimensional scaling is an optimization method for dimensions reduction that tries to represent data on a fixed number of dimensions, by equalizing the dissimilarities maps of both matrixes.



There are several parameters that can be chosen to optimize performance, and a short description of them is provided below.

Parameters:

Dimension: Desired number of dimensions for data after optimization process.

Criterion: The goodness-of-fit criterion to minimize. This also determines the type of scaling, either non-metric or metric, that *mdscale* performs.

Max Number of Iterations: Maximum number of iterations allowed.

TolFun: Termination tolerance for the stress criterion and its gradient.

TolX: Termination tolerance for the configuration location step size.

Distance Metric: Metric for calculation of dissimilarities during the attempt to minimize criterion. For coherence, one should use the same metric on dissimilarities and distance, although it is possible to try combinations of metrics.

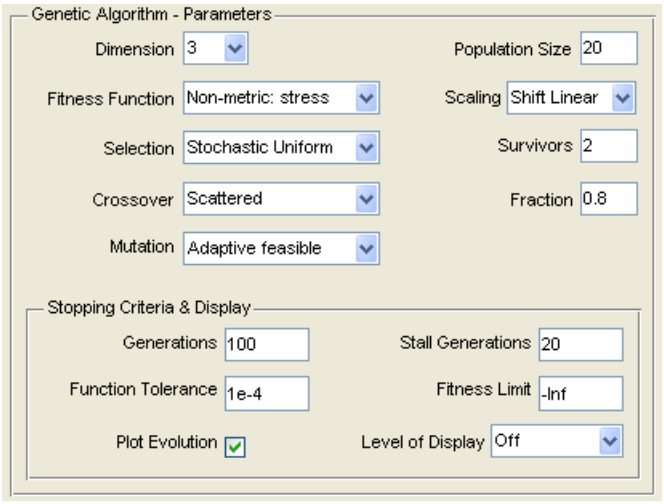
Display: Amount of displayed output on Matlab's console.

(For further information, see *mdscale* in Matlab Help, from Statistics Toolbox)

Genetic Algorithm

Genetic algorithms are a paradigm for solving optimization problems that is based on natural selection, the process that drives biological evolution. The genetic algorithm repeatedly modifies a population of individual solutions. At each step, it selects individuals from the current population to be parents and uses them to produce the children for the next generation. Over successive generations, the population evolves towards an optimal solution[1].

The Genetic Algorithm panel is enabled by selecting the *Genetic Algorithm* option on the *Optimization Algorithm* panel:



This method is based on the *ga* function from Genetic Algorithm and Direct Search Toolbox. Tens of variables can be modified using this function; however, only the most significant parameters (those that produce the highest behavior changes) can be changed on the *Genetic Algorithm – Parameters* panel, as described below:

- *Dimension*: defines the number of dimensions for data after optimization process.
- *Population Size*: defines how many individuals there are in each generation.
- *Fitness Function*: defines the goodness-of-fit criterion the algorithm will try to minimize on each generation.

[1] *Genetic Algorithm and Direct Search Toolbox User's Guide*, The MathWorks, Inc. 2007

- *Scaling*: converts raw fitness scores returned by the fitness function to values in a range that is suitable for the selection function.

- *Selection*: chooses parents for the next generation based on their scaled values from the fitness scaling method.

- *Survivors*: specifies the number of individuals that are guaranteed to survive to the next generation.

- *Crossover*: specifies how to perform the combination of two individuals, or parents, to form a new individual, or child, for the next generation.

- *Fraction*: specifies the fraction of the next generation that is produced by crossover. The remaining offspring in the next generation is produced by mutation.

- *Mutation*: specifies how to make small random changes in the individuals in the population, which provide genetic diversity.

- *Generations*: specifies the maximum number of iterations the genetic algorithm performs.

- *Stall Generations*: specifies the maximum computing generations without significant fitness improvement.

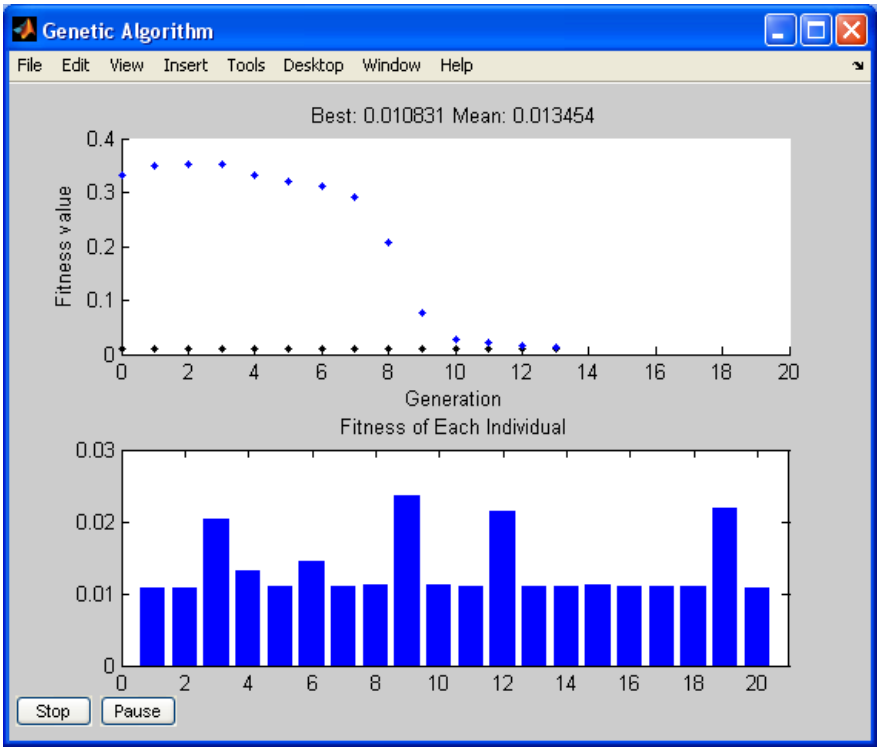
- *Function Tolerance*: specifies the minimum cumulative change in the fitness function value over stall generations.

- *Fitness Limit*: specifies the minimum achievable fitness value.

- *Plot evolution*: allows the user to plot genetic algorithm evolution as it is executing – if selected, the user can stop the process by clicking the *Stop* button.

- *Level of Display*: amount of information displayed on Matlab's console.

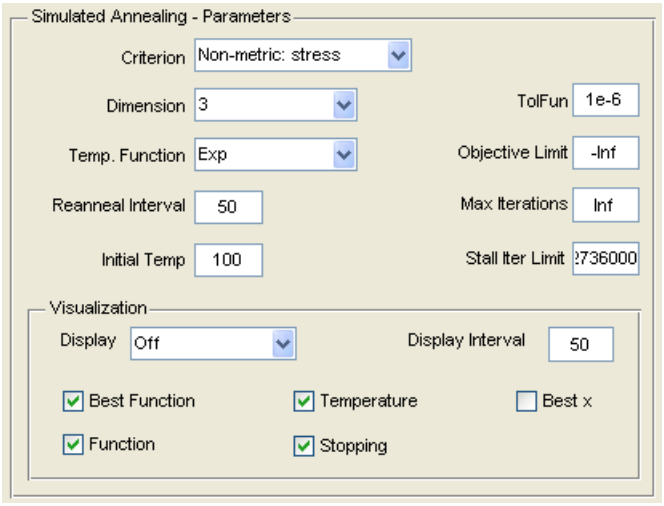
While the genetic algorithm is executing, if *Plot Evolution* is selected an evolutionary plot will be shown (overall evolution on top and iterative behavior on bottom):



When the optimization is completed, a dialog box shows the improvement achieved by this algorithm and results are saved on the *reducedData* structure.

Simulated Annealing

The name and inspiration for this technique came from annealing in metallurgy, a technique involving heating and controlled cooling of a material to increase the size of its crystals and reduce their defects. The heat causes the atoms to become unstuck from their initial positions (a local minimum of the internal energy) and wander randomly through states of higher energy; the slow cooling gives them more chances of finding configurations with lower internal energy than the initial one[2].



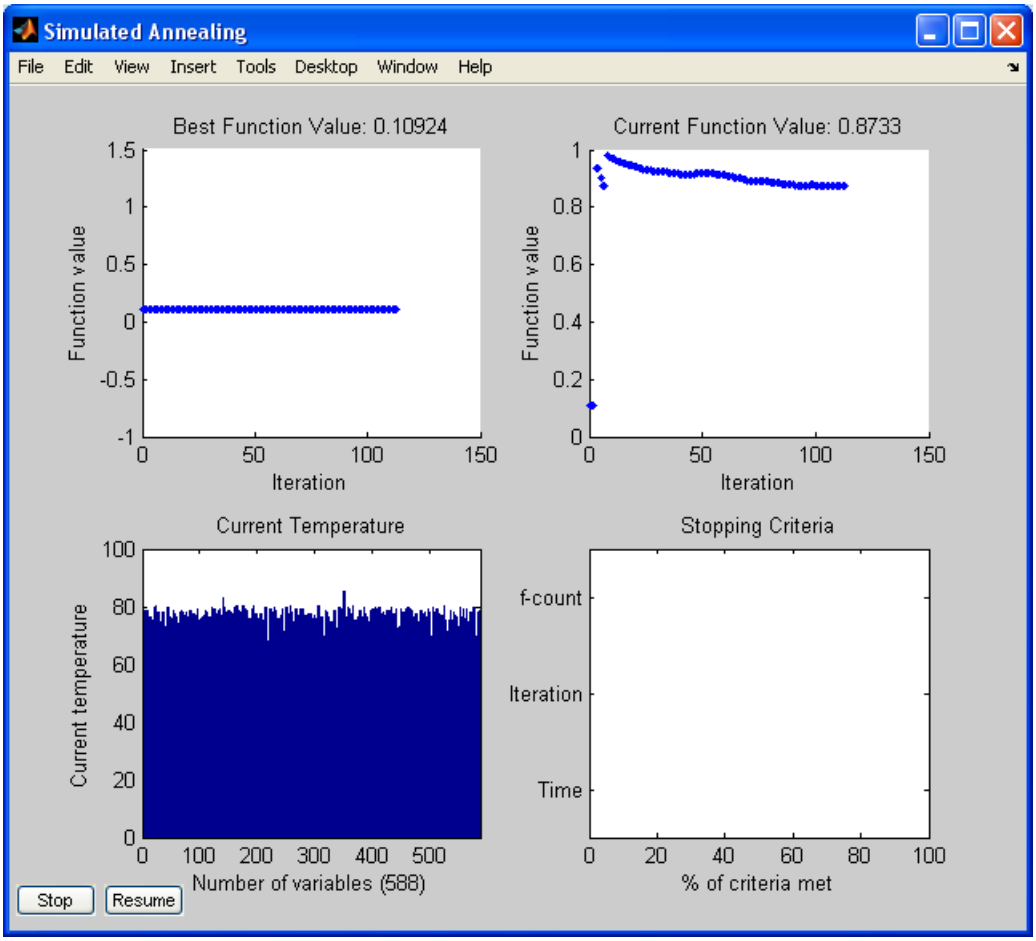
This method is based on the *simulannealbnd* function from Genetic Algorithm and Direct Search Toolbox. From all the variables that can be modified, the ones thought as more

[2] http://en.wikipedia.org/wiki/Simulated_annealing

important were selected and can be changed on the *Simulated Annealing – Parameters* panel, as listed below:

- *Dimension*: defines the number of dimensions for data after optimization process.
- *Criterion*: defines the goodness-of-fit criterion the algorithm will try to minimize on each generation.
- *Temperature Function*: function used to update the temperature schedule.
- *Initial Temperature*: initial temperature at the start of the algorithm.
- *Reanneal Interval*: number of points accepted before re-annealing (increasing temperature).
- *Function Tolerance (TolFun)*: specifies the minimum cumulative change in the fitness function value over stall generations.
- *Objective Limit*: minimum objective function value desired.
- *Maximum Iterations*: maximum number of iterations to execute before stopping.
- *Stall Iteration Limit*: number of iterations over which average change in objective function value at current point is less than *Function Tolerance*.
- *Display*: specifies the amount of information displayed on Matlab's console.
- *Display Interval*: specifies the interval for iterative display.
- *Plot Functions*
  - *Best Function*: best function value obtained.
  - *Temperature*: current temperature for each variable.
  - *Best X* : best point found so far (best X).
  - *Function*: current function value.
  - *Stopping*: percentage of completed stopping criteria.

For the default selected plot functions, the plot that will appear is depicted below:

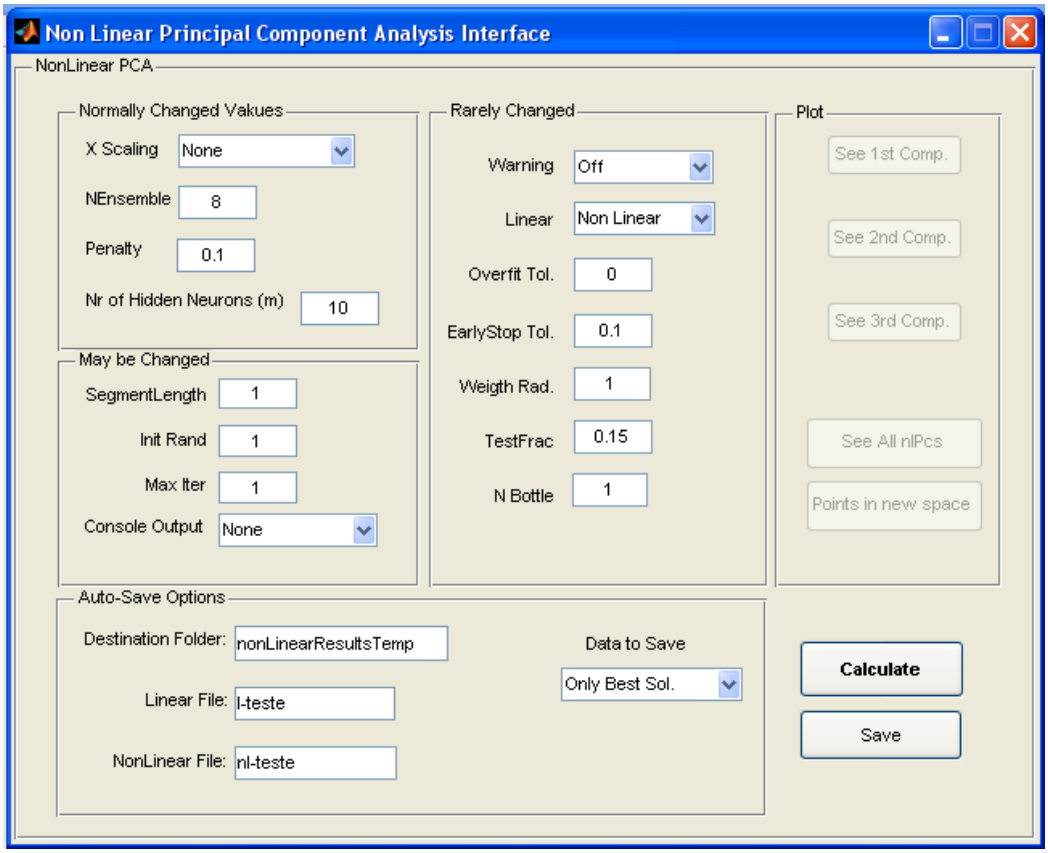


When optimization is completed, a dialog box shows the improvement achieved by this algorithm and results are saved on the *reducedData* structure.

Non Linear PCA

Non Linear PCA opens a second GUI that allows calculating nonlinear principal components accordingly to the principle of bottleneck neural networks used for dimensions reduction.

Nonlinear principal component analysis optimization is applied to the *n* first principal components, where *n* is a number or variables which sum of normalized Eigenvalues is above 0.995 (this means that data explains at least 99.5 % of variance). The developed interface implements the existing code of William W. Hsieh on his Neuralnets for Multivariate and Time Series Analysis (NeuMATSA) work. Below is depicted a figure of the developed GUI:



Here the user can easily manipulate several parameters of the bottle-neck neural network which is used to reduce dimensions:

- *X Scaling* – Defines the scaling of each variable, which can be none, or removal the mean and division by standard deviation.
- *NEnsemble* – Number of retraining of the NN. Since the sample to train and test is acquired randomly, increasing this value will increment the possibility of find a global minimum.
- *Penalty* – Scales the weight penalty term in the cost function. Increasing penalty leads to less nonlinearity.
- *Nr of Hidden Neurons* – Number of neurons in the hidden layer. Increasing this number will increase the nonlinearity of the possible solution.

- *Segment Length* – Data for training and testing are chosen in segments with length equal to segmentlength (increase this value if correlation of data is a problem).
- *Init Rand* – Initializes random numbers generator.
- *Max Iterations* – Scales the maximum number of iterations allowed.
- *Console Output* – Amount of information that will be displayed in the console.
- *Warning* – Allows user to choose whether internal warnings will be shown.
- *Linear* – Define if the calculated PCA will be Linear or NonLinear.
- *Overfit Tolerance* – Tolerance for overfitting. When this value is zero, the program automatically chooses an overfitting tolerance level.
- *Early Stop Tolerance* – Should be around 0.05 to 0.15, and avoids overfitting.
- *Weight Radius* – Controls the initial random weight radius for the ensemble member, and must be adjusted if there are convergence problems.
- *Test Fraction* – Fraction of data that will be reserved to NN testing.
- *N bottle* – Number of neurons in the bottle-neck layer. Although it is possible to use more than one neuron for calculation, visualization of this data must be hand-made by the user, since the interface only covers the possibility of using 1 neuron in the bottleneck layer (in order to obtain orthogonal components).

For a more detailed description of each variable you should read the original user guide by William W. Hsieh.

Each time the *Calculate* button is pressed, a new NLPC is calculated, up to a maximum of 3 of them. Components can be views regarding the data used to retrieve them – *See n^{th} Comp.* –, the initial data that's being analyzed – *See all nlPcs* –, and the mapping of the original data to the new space.

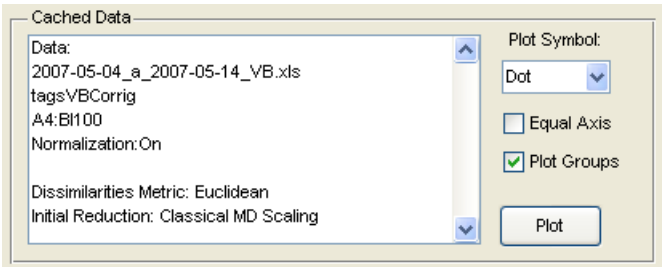
Due to the existing implementation, each of the resulting principal components is automatically stored with a distinct name on the designated destination folder, which allows the data to be reviewed. As these files are also needed to plot the components in the interface, it's not advisable to change them while running the interface.

After finishing all calculations, pressing the save button will close the interface and send the points in the new space back to VisRed main interface. Closing the interface by other means will simply return to VisRed.

DATA VISUALIZATION AND MANAGEMENT

Plotting Data

The *Cached Data* panel shows information regarding the data that is currently being analysed.



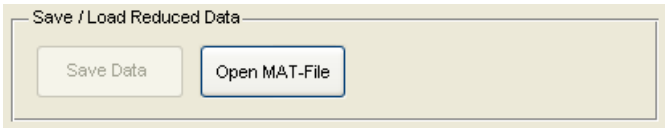
By pressing the *Plot* button, it is possible to view data represented by points in 2D or 3D (it depends on the amount of dimensions that were generated). The user may also choose the plot symbol, keeping in mind that symbols with minus signal (-) will have a line connecting the points sequentially. The *Equal Axis* option allows the disabling of auto sizing of figure axis. The *Plot Groups* option is enabled when categorized data are loaded (as referred in the *Data Import* chapter). When this option is checked, the user can plot each group on a different color.

Save and Load Reduced Matrices

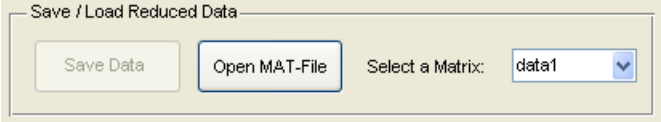
It is possible to save processed data (with respective data labels, if existing) into a file, along with the description present at the time in the *Cached Data* window. The user can choose the name and location of the destination file in the pop-up window that appears. This file contains the structure that holds all labels, the originally imported data and the reduced data (with Eigenvalues if they exist).

Reading from a MAT file

One can import matrices that have already been reduced and saved with this application, avoiding the necessity of recalculating the reduced matrix which, depending on the amount of data and used methods, can take from a few minutes to several days. This can be done by just pressing the *Open MAT-File* button and selecting a *.mat* file. If the *.mat* file was not saved by this application (does not contain some specified fields), an error message will appear.



Labels and information of data and processes applied to the analysis were also stored during the save process, and are now read and made available. When the user opens a file that contains more than one object, the first matrix is loaded and a list with the existing variables in the file appears (which means this file was not created by this application).

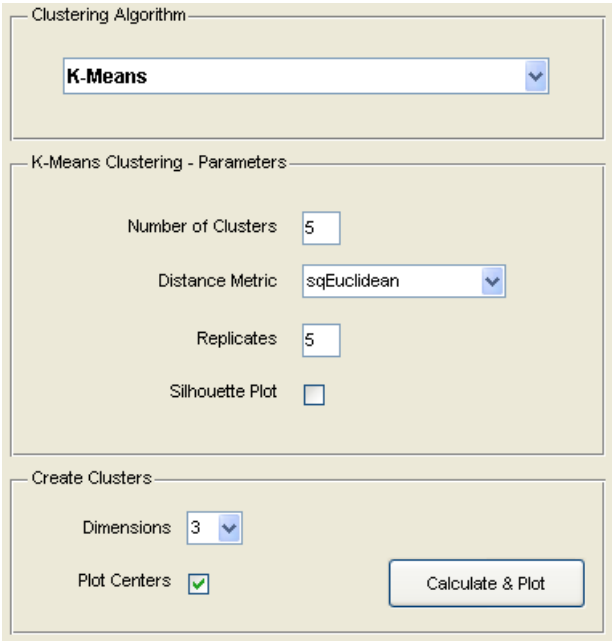


Matrices with only one dimension or other types of unrecognized variables will not be loaded and an error message will appear.

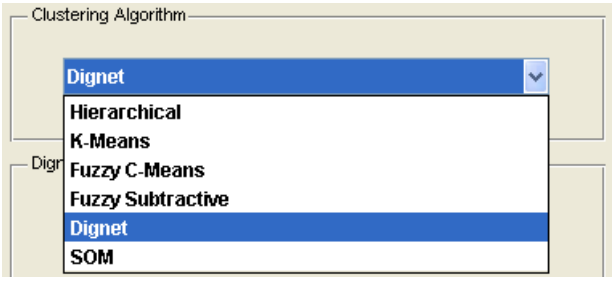
CLUSTERING

Clustering Algorithm

Clustering is enabled after computing dimension reduction (*Calculate* button on *Dimensions Reduction Method* panel), or when a reduced matrix is loaded from a recognizable *.mat* file. In this stage the three panels that allow clustering control are enabled:



Clustering Algorithm panel allows choosing one of six available clustering methods:

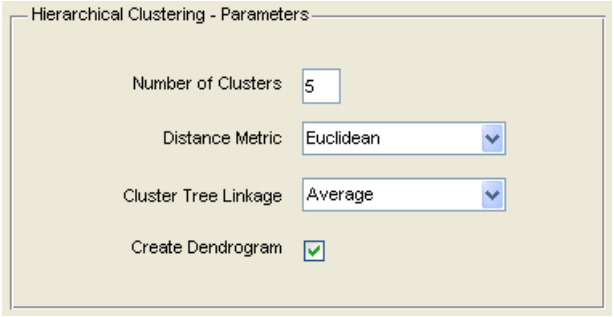


Depending on the choice of clustering algorithm, parameters panel presents variables related to current method, which are user controllable.

When reduced data is obtained by a reduction method or loaded from a file matrix with more than two dimensions, it is possible to set the number of dimensions to use in clustering process (in *Create Clusters* panel).

Next, a description of each method parameters is presented (by default, each method is set with values that produced the best clustering results).

1. Hierarchical Clustering



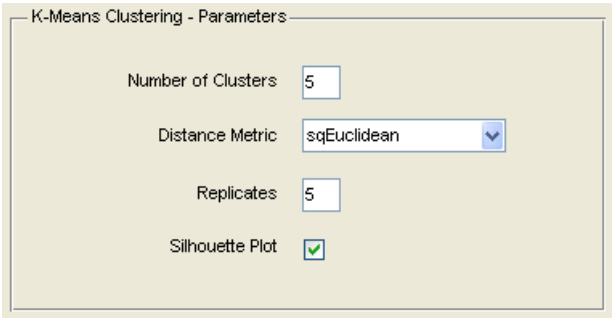
- *Number of Clusters*: number of groups to create.

- *Distance Metric*: metric used to compute distance between points. It can assume one of the six following values (for more information, refer to *pdist* function in Matlab Help):
  - Euclidean
  - Seuclidean
  - Mahalanobis
  - Cityblock
  - Chebychev
  - Minkowski (p=9)

- *Cluster Tree Linkage*: algorithm used to compute the hierarchical cluster tree. It can assume one of two values (for more information, refer to *linkage* function in Matlab Help):
  - Average
  - Complete

- *Create Dendrogram*: allows user to create a dendrogram plot. A dendrogram consists of many U-shaped lines connecting objects in a hierarchical tree. The height of each U represents the distance between the two objects being connected (for more information, refer to *dendrogram* function in Matlab Help).

2. K-Means Clustering



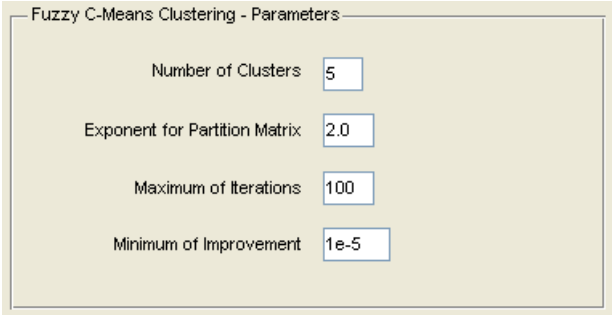
- *Number of Clusters*: number of groups to create.

- *Distance Metric*: defines distance measure to compute distance. This algorithm computes centroid clusters differently for the different supported distance measures. It can assume one of the three following values (for more information, refer to *kmeans* function in Matlab Help):
  - sqEuclidean
  - Cityblock
  - Cosine

o *Replicates*: number of times to repeat the clustering, each with a new set of initial cluster centroid positions. It is returned the solution with the lowest centroid distance sum (for more information, refer to *kmeans* function in Matlab Help).

o *Silhouette Plot*: allows user to create a silhouette plot. Silhouette plot focus the distance between neighbor clusters, setting lower values to points closer to neighbor clusters (for more information, refer to *silhouette* function in Matlab Help). Silhouette is plotted using the interpoint distance measure specified in *Distance Metric*.

3. Fuzzy C-Means Clustering



o *Number of Clusters*: number of groups to create.

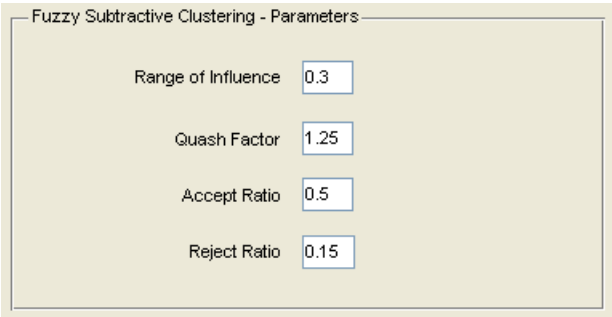
o *Exponent for Partition Matrix*: defines exponent for the partition matrix U.

o *Maximum of Iterations*: maximum number of iterations; computing is stopped when centroids distance sum doesn't decrease significantly between iterations.

o *Minimum of Improvement*: defines algorithm sensibility; clustering is stopped when the objective function improvement between two consecutive iterations is less than the minimum amount of improvement specified.

For more information on this parameters, refer to *fcm* function in Matlab Help.

4. Fuzzy Subtractive Clustering



o *Range of Influence*: variable between 0 and 1 that specifies cluster center's range of influence, assuming the data falls within a unit hyper box. Small values commonly result in finding a few large clusters. Good values are usually between 0.2 and 0.5.

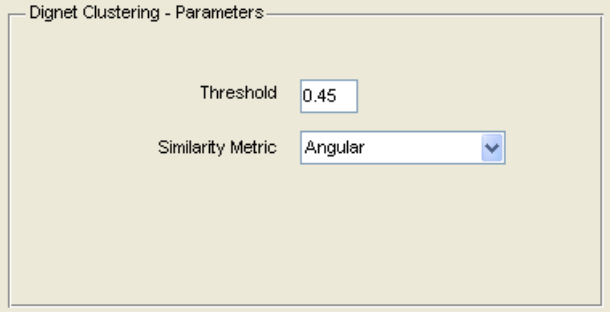
o *Quash Factor*: factor used to multiply the radii values that determine the neighborhood of a cluster center.

o *Accept Ratio*: potential, as a fraction of the potential of the first cluster center, above which another data point will be accepted as a cluster center.

o *Reject Ratio*: potential, as a fraction of the potential of the first cluster center, below which a data point will be rejected as a cluster center.

For more information on this parameters, refer to *subclust* function in Matlab Help.

5. Dignet Clustering



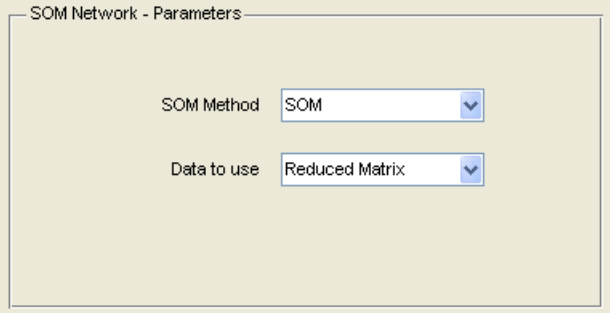
o *Threshold*: defines noise threshold used to compute clusters centers; small values result in finding a large number of clusters.

o *Similarity Metric*: defines distance measure to compute distance between cluster centers and data points. It can assume one of four values:

- Angular: $S_{j,n} = \arccos\left(\frac{\langle e_{j,n-1}; x_n \rangle}{\|e_{j,n-1}\| \cdot \|x_n\|}\right)$
- Euclidean: $S_{j,n} = \sqrt{\sum_k \left(e_{j_k,n-1} - x_{k,n}\right)^2}$
- Chebychev: $S_{j,n} = \max_k \left|e_{j_k,n-1} - x_{k,n}\right|$
- Cityblock: $S_{j,n} = \sum_k \left|e_{j_k,n-1} - x_{k,n}\right|$

In this method, $S_{j,n}$ defines similarity, e defines the cluster center that is being computed and x defines the point that is being measured.

6. SOM Clustering



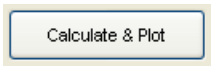
o *SOM Method*: defines SOM network method. Classical SOM and Recursive SOM are available.

o *Data to use*: allows user to choose which matrix to use (original or reduced data).

Note: these clustering methods work on a different way when compared to classical clustering methods: a new interface window where it is possible to control and to visualize training and classification process is opened.

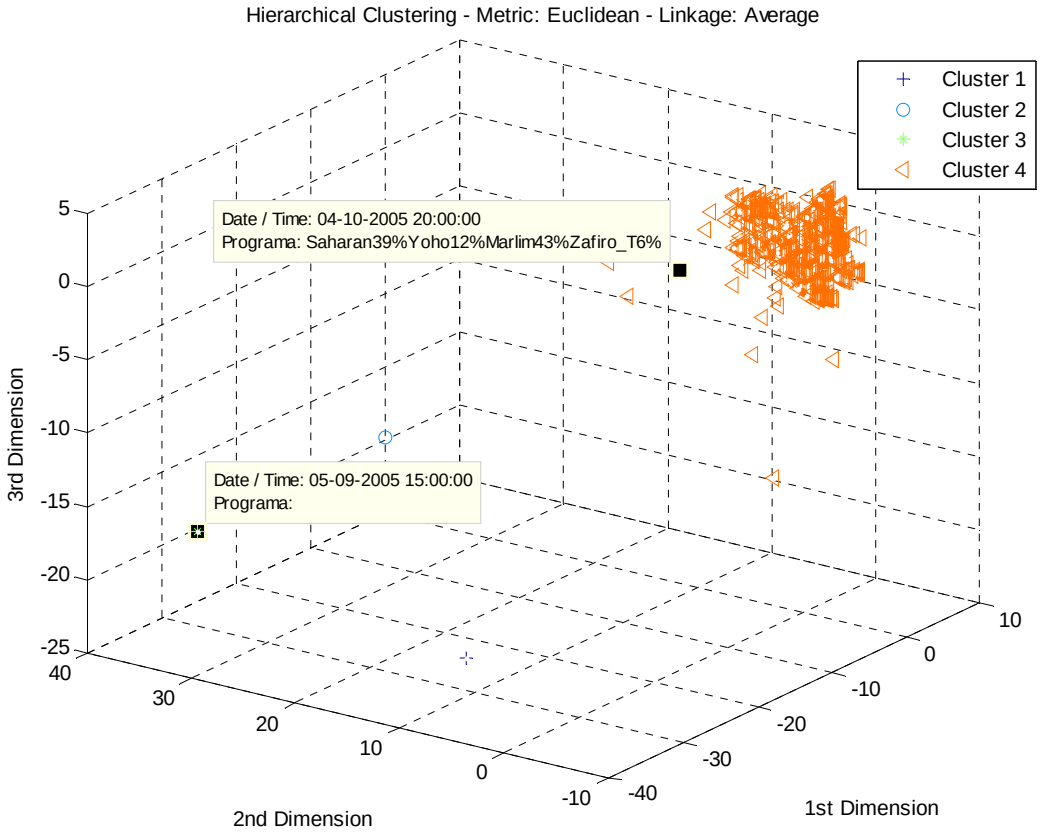
Clustering Method Calculation

Description in this section is valid for all clustering methods, except SOM methods, once these are computed on specific windows. Clustering is computed by clicking *Calculate & Plot* button on *Create Clusters* panel:



Thus, clustering is done and the respective plot is generated (3D or 2D, as defined on this panel or on *MDS – Parameters* panel). In this graph each cluster is represented with a distinct color and symbol.

Created plots have custom titles, illustrating some important aspects of applied clustering method, as well as cluster legend. Using 'Data Cursor' button (enabled by default), it is possible to show each point date and time (and industrial program, if available on read data); these two labels are only possible examples for this study case because application loads all label columns that are available on selected Excel file. Besides 'Zoom', another interesting tool on 3D plots is 'Rotate 3D' that allows rotating plot and analyzing it according to coordinate pairs.



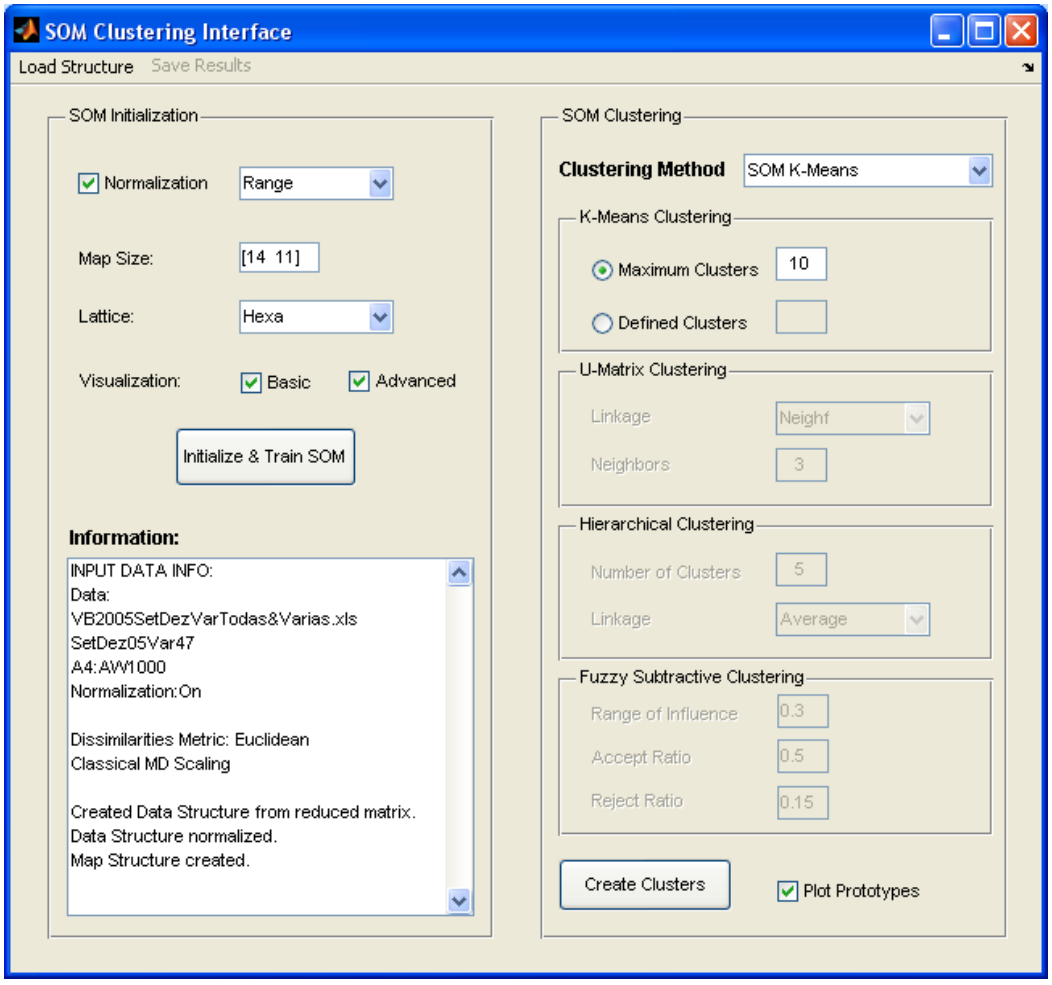
Example of a Hierarchical Clustering generated plot

Plot title is *Hierarchical Clustering – Metric: Euclidean – Linkage: Average*, which shows applied clustering method and the most important defined parameters.

In this case there are two visible point labels (pushing Alt key allows showing several point labels at a time).

SOM CLUSTERING INTERFACE

SOM Clustering Interface window opens when user clicks *Open SOM Guide* button on *Create Clusters* panel, after selecting SOM Clustering method:



Self-Organizing Map (SOM), proposed by T. Kohonen, is an unsupervised neural network method which has properties of both vector quantization and vector projection algorithms. The prototype vectors are positioned on a regular low-dimensional grid in an ordered fashion, making the SOM a powerful visualization tool[3].

The SOM can be thought of as a net which is spread to the data cloud (as prototype vectors). The SOM training algorithm moves the weight vectors so that they span across the data cloud and so that the map is organized: neighboring neurons on the grid get similar weight vectors.

SOM Initialization

On *SOM Initialization* panel it is possible to control several SOM train parameters:

- *Normalization*: when selected, normalization is applied to given original or reduced data, using one of three methods: *Range* (values are normalized between 0 and 1), *Var* (variance is normalized to one), *Log* (natural logarithm is applied to the values: $x = \log(x - \min(x) + 1)$).

[3] Vesanto, J., Himberg. J, Alhoniemi, E. and Parhankangas, J. (2000), SOM Toolbox for Matlab 5, Helsinki University of Technology

- *Map Size*: map grid size. This parameter is normalization dependent but it can be set by user.

- *Lattice*: topology, or structure, of the SOM. It can be defined by: *Hexa* (hexagonal shape) or *Rect* (rectangular shape).

- *Visualization*: it is possible to select two kinds of SOM visualization: *Basic* (shows the *U-Matrix*, a unified distance matrix of a SOM that focus the distance between map units) and *Advanced* (shows a four subplots window: a *U-Matrix* with map units sized according to the number of matching points, a surface distance matrix, a prototype plot and a prototype/data mixed plot).

- *Initialize & Train SOM* button: SOM train is performed, generating selected visualization methods and enabling clustering methods in *SOM Clustering* panel.

Note: *Information* field displays important info on performed actions.

SOM Clustering

SOM clustering is applied to the prototype set computed in SOM train. Then, each original point is matched to the closest prototype cluster index.

Clustering Method allows four different clustering algorithms:

1. SOM K-Means

   - *Maximum Clusters*: maximum number of clusters. Algorithm selects the clustering number with smallest Davies-Bouldin index.

   - *Defined Clusters*: number of clusters to compute, regardless Davies-Bouldin index.

2. SOM U-Matrix

   - *Linkage*: defines distance method to compute distance between *U-Matrix* elements.

   - *Neighbors*: number of neighbors to extend distance measure.

3. Hierarchical Clustering

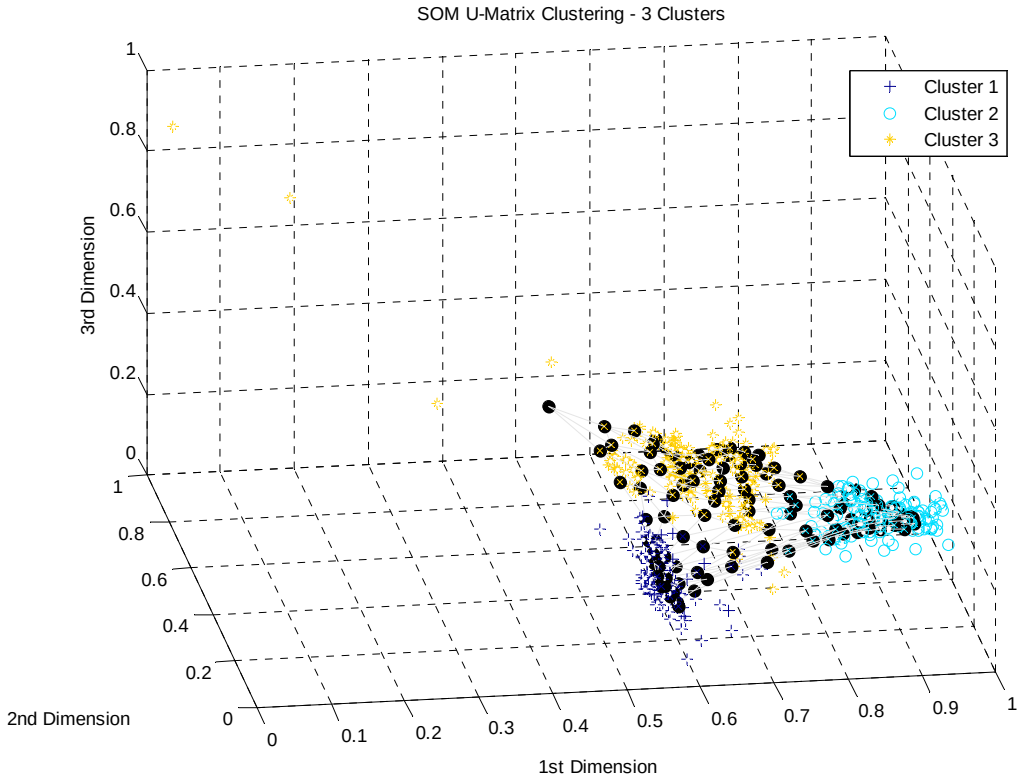
   Parameters described in *CLUSTERING* section.

4. Fuzzy Subtractive Clustering

   Parameters described in *CLUSTERING* section.

After choosing clustering method and respective parameters, user should click on *Create Clusters* button in order to proceed clustering and results plot.

Plot Prototypes selection box allows user to define if prototype vectors should be plotted. The final plot (including prototypes) looks like it follows:



Black linked points represent prototypes and color symbol points represent clustered original or reduced data. If original data has more than three dimensions, a 3D PCA projection of data is plotted.

Save SOM Results

User can save SOM clustering results on *.mat* and Excel files or send them to Matlab *Workspace*, by clicking on *Save Results* menu.

When *Save to MAT-File* submenu is clicked, user can choose name and path for the saving file, which will contain all the structures described on *Saving Results* section, including sD and sM structures, inside *clusteringData* structure. *Send to Workspace* method stores the same structures.

Clicking on *Save to Excel File* submenu, *Save Clustering results to Excel File* window is shown. This window was already described on *Saving Results* section.

Load SOM Structure

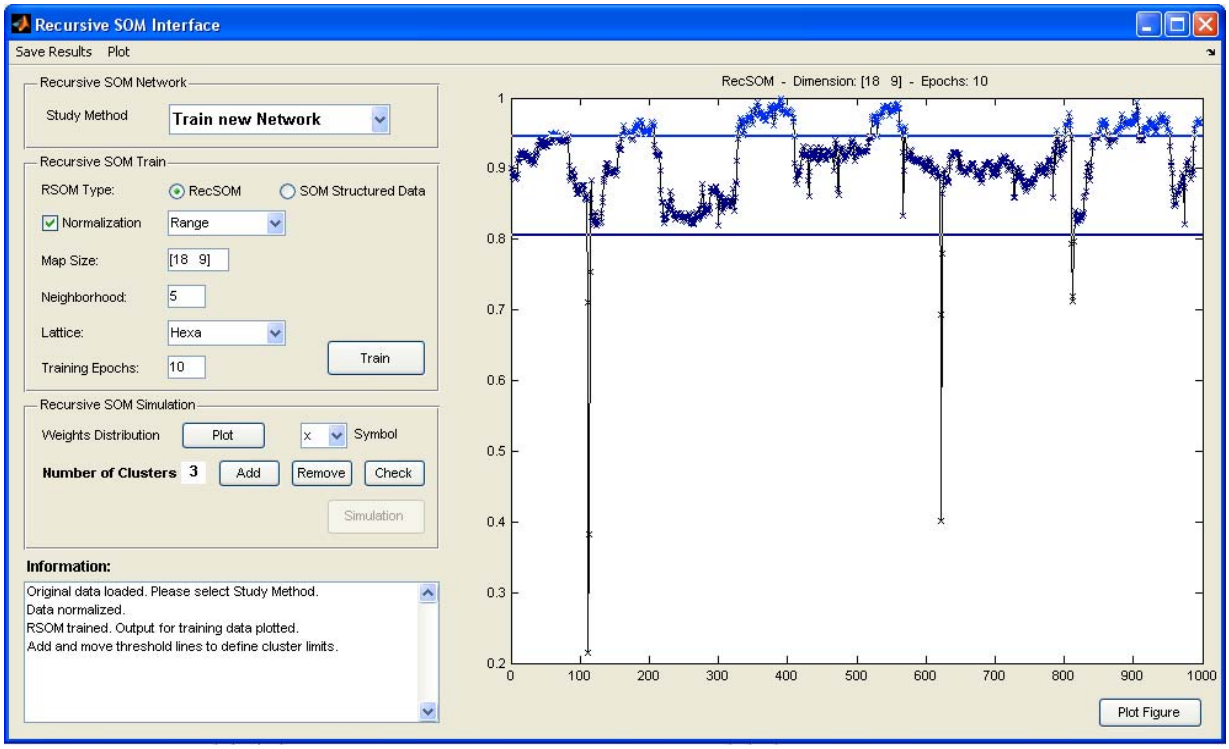
User can load *.mat* files containing SOM structures (*data structure* – sD – and *map structure* – sM) by clicking on *Load Structure* menu.

If an sM structure is loaded, *SOM Initialization* panel is disabled and SOM clustering methods become available.

If an sD structure is loaded, *SOM Clustering* panel is disabled and SOM initialization training methods become available.

RECURSIVE SOM INTERFACE

Recursive SOM Interface window opens when user clicks *Open RecSOM Guide* button on *Create Clusters* panel, after selecting Recursive SOM clustering method:



The Recursive SOM network is a generalization of SOM that learns to represent sequences recursively. Its resulting representations are adapted to the temporal statistics of the input series[4].

The SOM Structured Data network is also an extension of the standard SOM model, which allows the mapping of structured objects into a topological map. This mapping can then be used to discover similarities among the input objects[5].

Implementation of this interface is based on Recurrent Self-Organizing Map (RSOM) Toolbox for MATLAB, developed by Amir Reza Saffari Azar Alamdari, July 2005.

Study Method

Recursive SOM Network panel allows user to choose the intended study method to compute given interface data:

- *Train new Network*: allows user to train a SOM network using current data (original or reduced data);

- *Load trained Network*: a file selection window is shown, where one can load a network trained by this interface; loaded net can then be simulated using given interface data as its input. Net output for training data and cluster partition are plotted at interface plot area.

[4] Thomas Voegtlin (2002), Recursive self-organizing maps, Neural Networks 15 (2002) 979–991

[5] Hagenbuchner, M, Sperduti, A & Tsoi, AC (2003), A self-organizing map for adaptive processing of structured data, IEEE Transactions on Neural Networks, May 2003, 14(3), 491-505

Recursive SOM Train

Recursive SOM Train panel, only available on *Train new Network* study method, allows training parameters definition. Some of these parameters were already described on *SOM CLUSTERING INTERFACE* section, so following ones can be distinguished:

- *RSOM Type*: allows user to choose network type (Recursive SOM or SOM Structured Data, described above);

- *Training Epochs*: defines the number of network training iterations; regard that Recursive SOM is much slower than SOM Structured Data

When *Train* button is clicked, network train is performed and plot area shows the one-dimensional output for training data.

Note: *Information* field displays important info on performed actions.

Recursive SOM Simulation

Recursive SOM Simulation panel, only available after SOM training or loading, allows three main purposes:

- *Weights Distribution*: displays spatial sorting of trained (or loaded) network;

- *Number of Clusters*: allows setting cluster partition to apply to network output. One can add or remove clusters (between 1 and 8) and easily move colored cluster lines. *Check* button updates current cluster partition state by reformatting plot area. *Plot Figure* button plots the same results on a separated window;

- *Simulation* button: only available on *Load trained Network* method, computes net simulation for interface given data. Cluster partition is kept.

Note: *Plot* menu contains plot analysis tools that are similar to Matlab plot window tools.

Save Recursive SOM Results

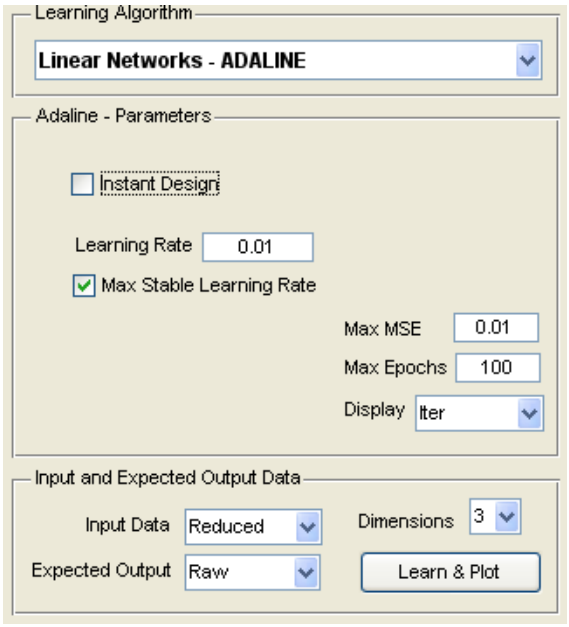
User can save SOM train and simulation results on *.mat* and Excel files or send them to Matlab *Workspace*, by clicking on *Save Results* menu.

When *Save to MAT-File* submenu is clicked, user can choose name and path for the saving file, which will contain all the structures described on *Saving Results* section, including network and cluster partition variables, inside *clusteringData* structure. *Send to Workspace* method stores the same structures.

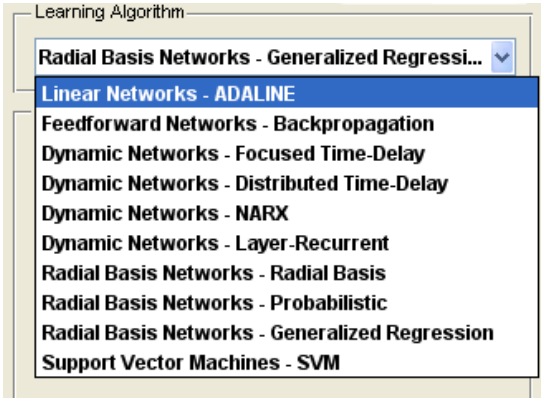
Clicking on *Save to Excel File* submenu, *Save Clustering results to Excel File* window is shown. This window was already described on *Saving Results* section.

LEARNING

Learning is enabled right after loading data, either raw (in the *Raw Data Panel*) or reduced (with the *Open MAT-File* button in the *Dimension Reduction* panel). The three panels that allow learning control then become enabled:



In the *Learning Algorithm* panel 10 different neural networks are available to perform supervised learning and classification of the data:



After selecting a learning algorithm from the list, the user is presented with a number of tunable parameters corresponding to the selected algorithm, in the *Parameters* panel. Each algorithm and its associated parameters will be described later, in the *Learning Algorithm* section.

Input and Target Output Data

The input data to the neural network can be either raw or reduced (when available), and the expected output data for training can be raw, binary (0,1) or bipolar (-1,+1) coded.

The user can choose any of these options in the *Input and Expected Output Data* panel, and if any of the pretended data sets is not available a warning message is issued.

Raw input data may be noisier than its reduced version, but data reduction usually results in some loss of information, so it's usually worth trying both sets and compare the results. The expected output is the data contained in the *Group Column* (see *Excel File Reading* section). This usually specifies the group, or class, that each sample belongs to, each class being identified by an integer. This raw representation may induce the learning algorithm into believing that some classes are more related than others, for example, that classes 1 and 2 are closer to each other than classes 1 and 3. This rarely is the case, so a binary or bipolar codification of the expected output is recommended, in same cases even mandatory (e.g. for Support Vector Machines). In binary or bipolar codification, as many outputs as the number of classes are used for each sample, where at each moment only one of them is on (i.e. contains the value 1), while the others are off (contain either 0 or -1, depending on the codification).

After selecting the input and output data formats, pressing the *Learn & Plot* button will execute the selected learning algorithm with its associated parameters.

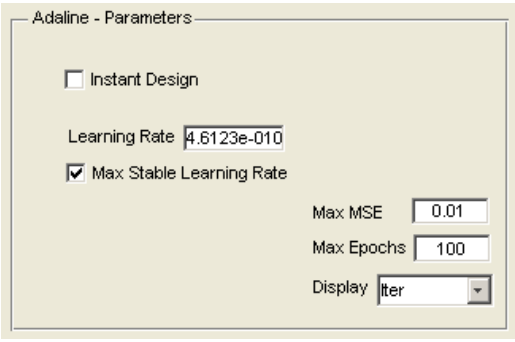
Learning Algorithm

What follows is a brief description of each of the available learning algorithms and their associated parameters.

1. ADALINE

This is a linear network and, as such, can only solve linearly separable problems. When given a set of input vectors, the network returns a set of output vectors, one for each input. The difference between an output vector and its target, or expected vector, is the error. The goal is to find values for the network weights and biases such that the sum of the squares of the errors is minimized or below a specific value. In most cases the linear network can be directly calculated so that its error is a minimum for the given input vectors and target vectors. In other cases, the network must be trained to have a minimum error by using the least mean squares (Widrow-Hoff) algorithm.

(For further information, see *newlind* and *newlin* in Matlab Help, from Neural Network Toolbox)



- *Instant Design*: when this option is selected, all others are disabled, and the ADALINE will be directly calculated from the input and target vectors.

- *Learning Rate*: rate used to update the network's weights and biases when using a training algorithm.

- *Max Stable Learning Rate*: maximum learning rate that will not cause the weights and biases to diverge – when selected, the application automatically calculates and uses this value for training.

- *Max MSE*: maximum mean squared error allowed before training stops (basically the goal to achieve it terms of error) – if *Max Epochs* (next parameter) is reached first, the training stops even if the error is higher.

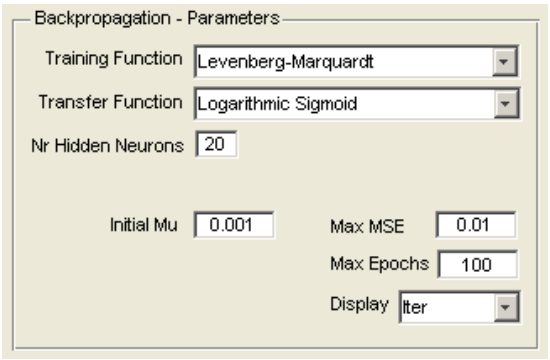
- *Max Epochs*: maximum number of training epochs.

- *Display*: periodicity of training information displayed on Matlab's console:
  - Off – no training information provided
  - Iter – training information provided every 25 epochs
  - Final – training information provided only after last epoch

2. Backpropagation

Backpropagation networks is the name usually given to feedforward neural networks trained with any variant of the backpropagation algorithm. Backpropagation is the generalization of the Widrow-Hoff learning rule to multiple-layer networks and nonlinear differentiable transfer functions. Backpropagation networks with biases, a sigmoid layer, and a linear output layer are capable of approximating any function with a finite number of discontinuities.

(For further information, see *newff* in Matlab Help, from Neural Network Toolbox)



In this application a number of parameters are associated to all backpropagation networks. Others only exist in the context of particular training functions.

- *Training Function*: several training functions are available (for further information, see the respective function name in Matlab Help, from Neural Network Toolbox):
  - Levenberg-Marquardt – *trainlm*
  - Gradient Descent – *traingd*
  - Gradient Descent with Momentum – *traingdm*
  - Gradient Descent with Adaptive Learning Rate – *traingda*
  - Gradient Descent with Momentum and Adaptive Learning Rate – *traingdx*
  - BFGS quasi-Newton – *trainbfg*
  - Resilient backpropagation (rProp) – *trainrp*

o *Transfer Function*: several transfer functions are available (for further information, see the respective function name in Matlab Help, from Neural Network Toolbox):
- Logaritmic Sigmoid – *logsig*
- Hyperbolic Tangent Sigmoid – *tansig*
- Linear – *purelin*

o *Nr Hidden Neurons*: number of neurons on the hidden layer.

o *Max MSE*: maximum mean squared error allowed before training stops – same behavior as in ADALINE.

o *Max Epochs*: maximum number of training epochs – same behavior as in ADALINE.

o *Display*: periodicity of displayed training information (same behavior as in ADALINE).

o *Initial Mu*: initial value of the adaptive μ parameter used in the Levenberg-Marquardt training function.

o *Learning Rate*: rate used to update the network's weights and biases in all Gradient Descent training functions and also in Resilient backpropagation (see parameter *Training Function*). This value is modified during learning in the training functions using an adaptive learning rate.

o *Momentum*: term used in the Gradient Descent with Momentum training functions (see parameter *Training Function*).

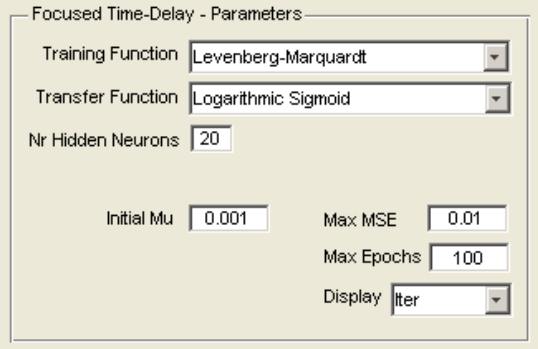
o *Search Routine*: search routine used in the BFGS quasi-Newton training function (for further information, see the respective function name in Matlab Help, from Neural Network Toolbox):
- Hybrid bisection-cubic search – *srchhyb*
- Golden section search – *srchgol*
- Charalambous method – *srchcha*
- Brents method – *srchbre*
- Backtracking – *srchbac*

o *Init Weight Change*: initial weight change used in the Resilient backpropagation training function (see parameter *Training Function*).

3. Focused Time-Delay

This is a dynamic network. Unlike static feedforward networks, dynamic networks can have feedback elements and contain delays. A Focused Time-Delay neural network is the most straightforward dynamic network, consisting of a feedforward network with a tapped delay line at the input. It is well suited for time-series prediction.

(For further information, see *newfftd* in Matlab Help, from Neural Network Toolbox)



As can be seen, the parameters associated to the Focused Time-Delay networks, as well as all other dynamic networks of this application, are exactly the same as the parameters of the previous backpropagation networks.

4. Distributed Time-Delay

This is another dynamic network. While the Focused Time-Delay network had the tapped delay line memory only at the input to the first layer, the Distributed Time-Delay can have the tapped delay lines distributed throughout the network. It was first designed for phoneme recognition problems.

(For further information, see *newdtdnn* in Matlab Help, from Neural Network Toolbox)

The parameters associated to the Distributed Time-Delay networks, as well as all other dynamic networks of this application, are the same as in the backpropagation networks.

5. NARX

The NARX (nonlinear autoregressive network with exogenous inputs) is a recurrent dynamic network, with feedback connections enclosing several layers of the network. The NARX model is based on the linear ARX model, which is commonly used in time-series modeling.

(For further information, see *newnarxsp* in Matlab Help, from Neural Network Toolbox)

The parameters associated to the NARX are the same used in the backpropagation networks.

6. Layer-Recurrent

First introduced by Elman in a simplified form, the Layer-Recurrent network contains a feedback loop, with a single delay, around each layer of the network except for the last layer.

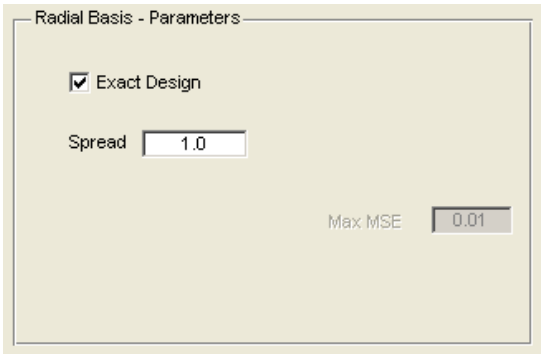
(For further information, see *newlrn* in Matlab Help, from Neural Network Toolbox)

The parameters associated to the Layer-Recurrent network are the same used in the backpropagation networks.

7. Radial Basis

In Radial Basis networks each output neuron receives the vector distance between its weight vector and the training input vector, multiplied by the bias, and applies a radial basis transfer function to the result. Radial basis networks can require more neurons than standard feedforward backpropagation networks, but they can often be designed in much less time than it takes to train standard feedforward networks. A Radial Basis network can be exactly designed to achieve null error on the input and target output data, or it can be built by iteratively adding hidden neurons until the mean squared error falls below a specified value.

(For further information, see *newrbe* and *newrb* in Matlab Help, from Neural Network Toolbox)

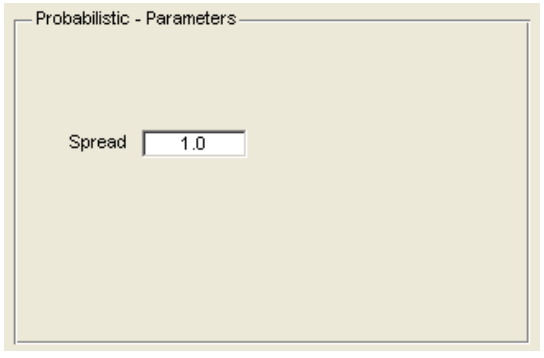


- *Exact Design*: when this option is selected, the *Max MSE* parameter is disabled and the network will be designed such that the mean squared error is null for the training data.
- *Spread*: parameter that controls the spread of the radial basis functions.
- *Max MSE*: maximum mean squared error allowed before training stops (basically the goal to achieve it terms of error).

8. Probabilistic

Probabilistic networks are radial basis networks containing two layers: the first is a radial basis layer that calculates distances between each input vector and previously seen training vectors; the second is a competitive layer that uses this information to produce a vector containing the probabilities of the input vector belonging to each training class. A final element selects the most probable class.

(For further information, see *newpnn* in Matlab Help, from Neural Network Toolbox)



There is only one tunable parameter for the Probabilistic networks:

- *Spread*: parameter that controls the spread of the radial basis functions.

9. Generalized Regression

Generalized Regression networks are also built with a radial basis layer, followed by a special linear layer. They perform regression, not classification like the Probabilistic networks. The output of a Generalized Regression network is simply a weighted average of the target values of training samples close to the given input.

(For further information, see *newgrnn* in Matlab Help, from Neural Network Toolbox)

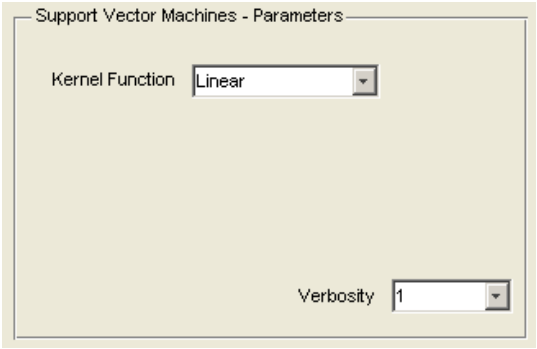
The only tunable parameter for Generalized Regression networks is *Spread*, the same as in Probabilistic networks.

10. Support Vector Machines (SVMs)

SVMs operate by finding a hyperplane in the space of training inputs. This hyperplane will attempt to split the positive examples from the negative examples, such that the distance between the hyperplane to the nearest of the positive and negative examples is the largest possible. This is particularly suited for binary classification problems.

To perform the separation between classes, the important concept is the relative position, or similarity, between the points. Such a similarity is computed by the kernel function. The simplest kernel function is the dot-product between two vectors (known as the linear kernel), which leads to a linear decision boundary between the two classes. Nonlinear kernels provide additional flexibility.

The implementation of SVMs is based on a MATLAB/MEX Interface to SVMlight by Tom Briggs, 2005. The SVMlight code is the intellectual property of Thorsten Joachims.



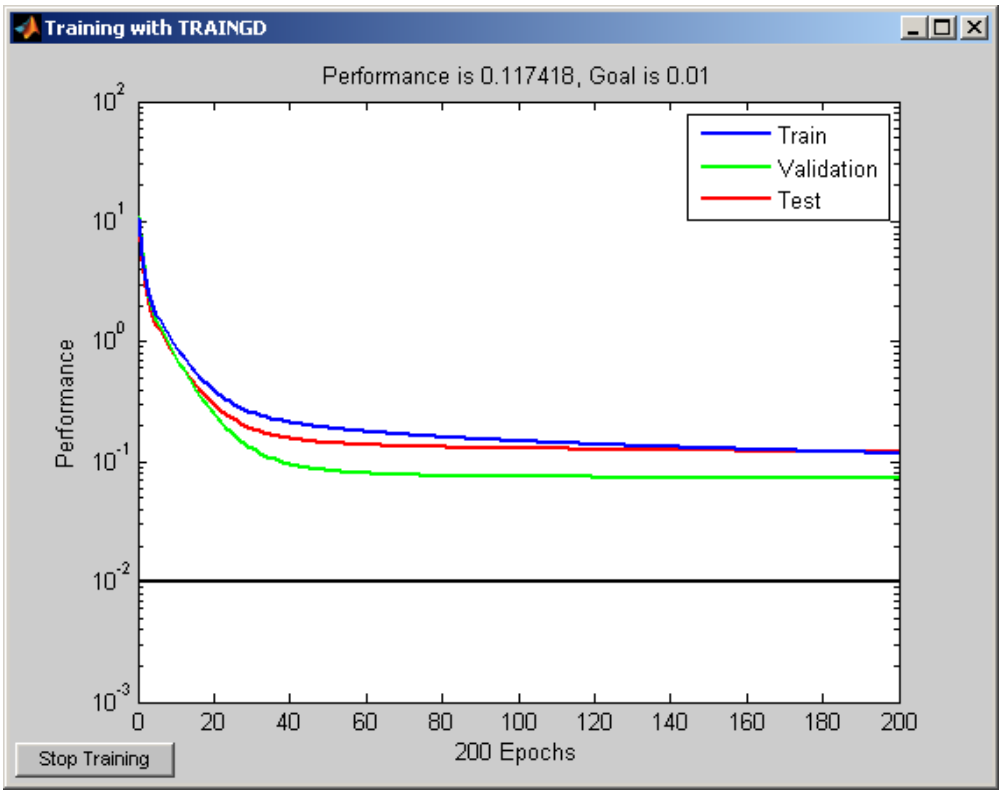
- *Kernel Function*: function used to determine the similarity between the points:
  - Linear
  - Polynomial
  - Radial Basis
  - Sigmoid

- *Verbosity*: amount of information displayed on Matlab's console.

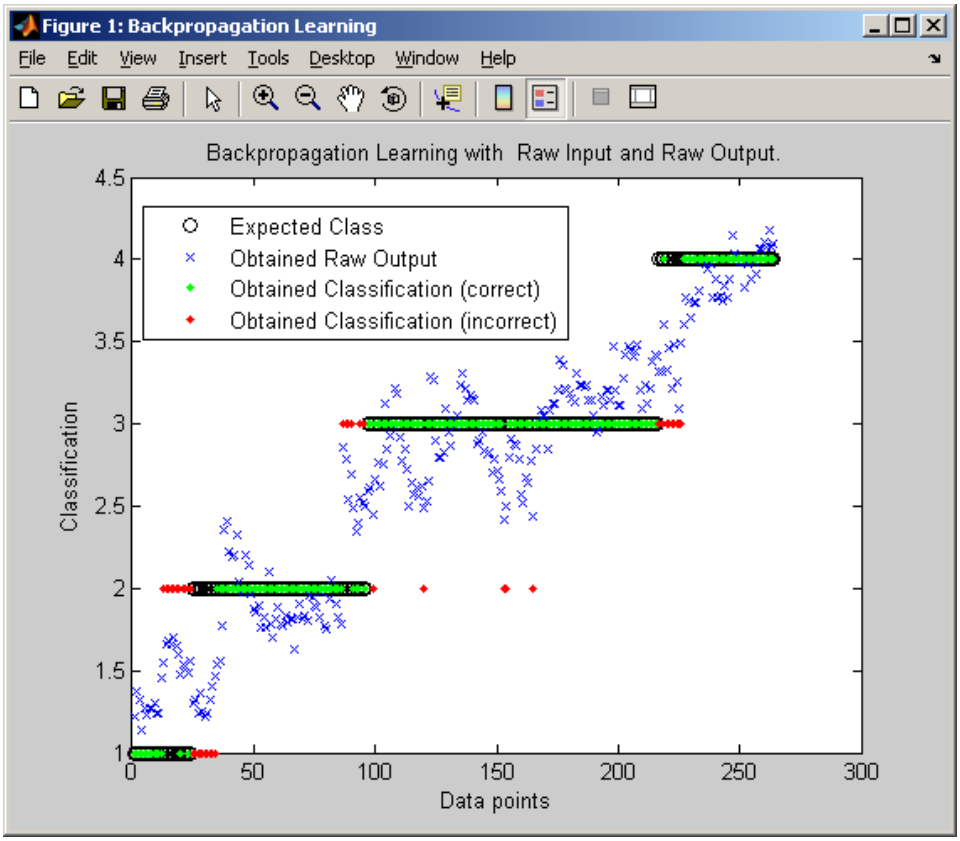
Most classification problems involve more than two classes, so the usage of SVMs to perform this task may not be straightforward. In this application several SMVs are arranged to performed multiclass classification. For each class in the training set, a different SVM is trained to separate this particular class from all the rest. When classifying a new data set, all the previously trained SVMs are used, and for each sample it is the SVM with the highest output that defines which class the sample belongs to. In case two or more SVMs return the exact same value, the sample is left unclassified.

Plotting of Training and Classification

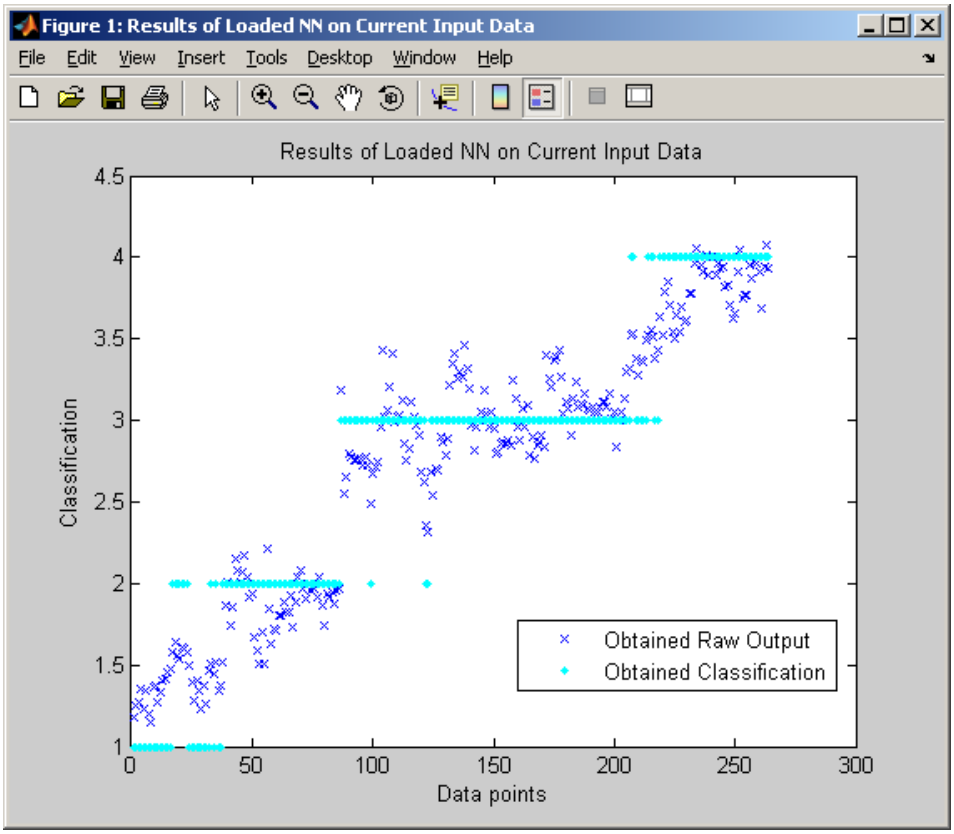
Depending on the learning algorithm and parameters selected, the application may plot the evolution of the error during training. In some cases the training data is divided in several sets (train, validation and test) and separate lines are drawn for each set:



When training is complete, a plot is made with the expected and obtained classification in the entire data set. The expected classification is represented by circles. If a raw target output was used, the raw output of the network is drawn (in blue), along with the classification deriving from it (the nearest integer number). Each classification is assigned a color: green for correct classifications and red for incorrect classifications:

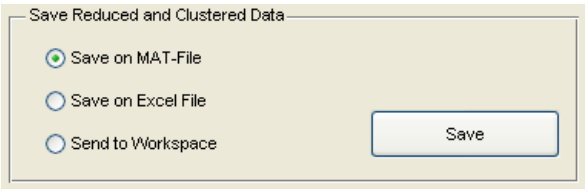


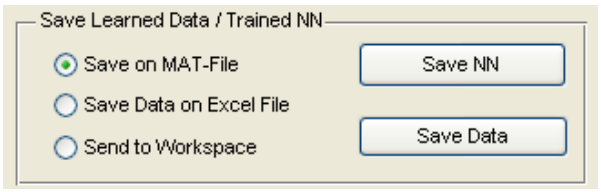
The previous coloring scheme is only used when plotting the results of training. When a previously trained network is used in a new data set the expected classification does not appear and the obtained classification is all drawn in cyan:



SAVING AND LOADING RESULTS

It is possible to save GUI generated information in *.mat* format, into an Excel file or in Matlab's workspace. The user only has to select the desired destination and click one of the *Save* buttons, on the *Save Reduced and Clustered Data* panel or on the *Save Learned Data / Trained NN* panel:





Besides saving data, it is also possible to save a trained neural network, by pressing the *Save NN* button on the *Save Learned Data / Trained NN* panel. This button is disabled when the "Save Data on Excel File" is selected, as this option refers only to data.

Save on / Load from MAT-File

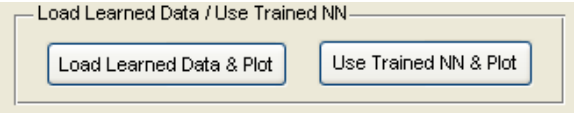
Similarly to what was described regarding the saving of reduced matrices (section Save *and Load Reduced Matrices*), it is possible to store dimension reduction and clustering and learning results into structures and save them on a *.mat* file. It is also possible to save a trained neural network.

By clicking the *Save* button (on the *Save Reduced and Clustered Data* panel) or the *Save Data* button (on the *Save Learned Data / Trained NN* panel) when the *Save on MAT-File* field is selected, the user can choose both the name and localization of the destination file, which will contain different data structures, depending on the type of data being saved, and whether the data has been read from a file that was not created by the application, or from an Excel file:

1. *rawData*: structure containing Excel read data, including each normalization method result;
2. *labels*: structure containing data labels, including variable names;
3. *reducedData*: structure containing reduced data, including several important parameters;
4. *clusteringData*: structure containing clustering results, including parameters information;
5. *learningData*: structure containing learning results, including parameters information.

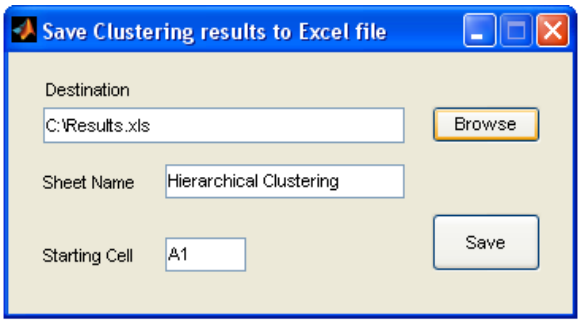
The saved data file can then be loaded on the *Save/Load Reduced Data* panel (*Open MAT-File* button), allowing reduced data analysis, on the *Load Clustered Data* panel (*Open MAT-File* button), allowing clusters visualization, or on the *Load Learned Data / Trained NN* panel (*Load Learned Data & Plot* button), allowing classification visualization. In all cases no loss of important information occurs. If the selected file does not contain all the necessary structures an error message will be displayed. When a valid data file is loaded, the *Plot* button of the *Load Clustered Data* panel will draw the clusters and labels representation:

To save a trained neural network the user should click the *Save NN* button on the *Save Learned Data / Trained NN* panel, and then choose the destination for the file containing the network. A saved network can later be used with different data sets, by loading new data and retrieving the saved network with the *Use Trained NN & Plot* button on the *Load Learned Data / Trained NN* panel:



Save on Excel File

By clicking one of the *Save* buttons when the *Save on Excel File* field is selected, the *Save results to Excel File* (clustering or learning) window is shown:



In this window it is possible to specify the destination file (using the *Browse* button or typing the file address in the *Destination* box). Then the worksheet must be set (which name is the applied clustering or learning method, by default) as well as the initial cell (top left cell) where the user wants to store the data. If the selected file or typed worksheet do not exist, they will be created when clicking the *Save* button.

Clustering data are stored according to the following configuration:

	A	B	C	D	E	F	G
1	Date / Time	Program	1st Dimension	2nd Dimension	3rd Dimension		Clusters
2	01-09-2005 0:00	Saharan39%Marlim54%EA7%	0,2913	-0,8247	0,8724		4
3	01-09-2005 1:00	Saharan39%Marlim54%EA7%	0,7137	-0,7183	1,2119		5
4	01-09-2005 2:00	Saharan39%Marlim54%EA7%	0,3503	-0,9914	0,8192		5
5	01-09-2005 3:00	Saharan39%Marlim54%EA7%	0,3427	-0,6286	0,3118		5
6	01-09-2005 4:00	Saharan39%Marlim54%EA7%	-0,7635	0,3428	-1,6859		2
7	01-09-2005 5:00	Saharan39%Marlim54%EA7%	0,2771	-1,2650	-1,5157		4
8	01-09-2005 6:00	Saharan39%Marlim54%EA7%	0,7055	-1,5091	-1,2432		4
9	01-09-2005 7:00	Saharan39%Marlim54%EA7%	0,4406	-1,1958	-1,5893		4
...	...	...	...	...	...		...

Column A contains the first label column information (date and time by default) created from *Read* on the *Read Raw Data* panel (or loaded from a *.mat* file created by this application). Column B contains, in this case, industrial information for each data entry; those labels are created from the second imported column on *Read*, on the *Read Raw Data* panel, in case this column is of non-numeric type. If more label columns are read from Excel file, they are also stored. Columns C–E contain reduced data dimensions (if dimension reduction was computed in 2D space, only columns C and D are filled). Column G contains the clustering index for each data entry.

Learning results are stored very similarly. Label Columns are the same, while the data columns may contain raw (the original variables) or reduced data (like columns C–E of the clustering example). The output columns are, however, slightly different, as they contain not only one column for the classification, but also one of more columns for the network output:

Output1	Output2	Output3	Output4	Class
0.028132	0.978165	0.025401	-0.0374	2
0.520279	-0.26206	0.29836	0.461406	1
0.119234	0.309719	0.0594	0.499702	4
0.149919	0.627634	-0.11125	0.362742	2
-0.01143	0.795738	0.724032	-0.46389	2
0.123539	0.640437	0.309593	-0.01108	2
-0.03245	0.368867	0.679544	0.021014	3
0.069996	0.159197	0.852897	-0.02831	3

Send to *Workspace*

By clicking one of the *Save* buttons when the *Send to Workspace* field is selected, the user can store data structures or trained neural networks in Matlab's workspace (see *Save on Mat-File* section).

Note: data to save includes cached and generated structures from the last time *Calculate & Plot* button was clicked, after selecting clustering method and parameters.