Exploratory Data Analysis GUIs for MATLAB (V1)

A User's Guide

Wendy L. Martinez, Ph.D.[1]
Angel R. Martinez, Ph.D.

Abstract:
The following provides a User's Guide to the Exploratory Data Analysis (EDA) GUI Toolbox for MATLAB. We include screenshots of the various GUI interfaces, along with brief explanations of what each GUI does. For more detailed information on the techniques that can be accessed using the GUIs, please see Exploratory Data Analysis with MATLAB *by the same authors.*

Acknowledgements
We would like to acknowledge the help of Tom Lane (The MathWorks, Inc.) and Heather (?) for their review of the GUI Toolbox. They offered many suggestions resulting in an improved product. Send to Seth Greenblatt and the guy at Anapolis. OTHERS?

1. Please direct inquiries, bug fixes, etc. to martinezw@verizon.net.

Introduction:

Operating System

This toolbox was developed and tested using a platform with MS Windows installed. There are versions of MATLAB for the MAC and Linux. While we did not test this toolbox for use in those environments, we expect the GUIs to work all right, but they will likely have a different appearance than the screenshots given here.

MATLAB Requirements

This EDA GUI Toolbox is meant to accompany the Exploratory Data Analysis Toolbox that was written by the same authors. The EDA GUIs do not incorporate all of the functionality available in the EDA Toolbox. The EDA Toolbox can be obtained from StatLib or the publisher's website for the book called *Exploratory Data Analysis with MATLAB.* However, the full EDA Toolbox is not required to use the GUIs.

However, to use *all* of the functionality of the EDA GUIs, the user *must* have the Statistics Toolbx from The MathWorks, Inc.

The toolbox was developed using MATLAB, version 6.5 and was tested using version 7.2 (2006a and 2006b).

Installation

Once the files have been downloaded, please follow these directions.

- Create a directory for the toolbox. It is preferable to use a subdirectyory under the main MATLAB directory.
- Unzip the files and save them to the toolbox directory you just created.
- Include the toolbox directory in the MATLAB search path by using the **`Set Path`** GUI from the **`File`** menu. Or, one can use the **`addpath`** function from the command line.

Getting Help

We do not include MATLAB **`Help`** files with this toolbox, except for the usual command line help that is available in MATLAB. In other words, typing

```
help edagui
```

at the command line will provide some information about the GUI.

So, we tried to include, on the GUI itself, some explanation of the functions and controls. We also provide Tool Tips that come up when one leaves the mouse pointer on a GUI control.

While we are not human computer interface experts, we tried to design the GUI layouts to be user friendly and intuitive. Most of the GUIs have controls that are grouped together by function, and step-by-step instructions are given where feasible. In all cases, the action, algorithm, or visualization does not take place until the user presses the appropriate button.

List of GUIs

The following is a summary of the GUIs that are included in the EDA GUI Toolbox. Please note that most of these GUIs can be used in a stand-alone manner. i.e., without calling **`edagui`**.

- **`edagui`**: This is the main entry point for the GUI system. While it is not necessary, one can use this to access the various GUIs available in the toolbox.
- **`loadgui`**: This is used to load the data set and other optional information (e.g., class labels, variable labels, and case labels. Most of the GUIs assume that the data is in a matrix form, where the rows correspond to observations, and the columns represent variables. The **`Multidimensional Scaling GUI`** allows one to load the interpoint distance matrix instead of the data matrix.
- **`transformgui`**: This GUI allows one to apply various transforms to the data before exploring them. This includes the ability to spherize the data, center them, and scale them.

- **`gedagui`**: This is the **`Graphical EDA GUI`**. It provides the standard tools for visualizing the data (e.g., scatterplots), as well as some non-standard tools (e.g., Andrews' curves and parallel coordinate plots). There is a button on this GUI that allows one to color the data using brushing and other means.
- **`univgui`**: This GUI has functionality to explore the distribution shapes of the individual variables (matrix columns). One can create boxplots, univariate histograms, and q-q plots.
- **`bivgui`**: One can use this to explore the distribution shapes of any two variables, by creating bivariate scatterplot smooths and histograms.
- **`tourgui`**: This GUI provides tools for the grand tour and permutation tour. These enable one to look at the data from many viewpoints.
- **`ppedagui`**: This tool implements the projection pursuit method for exploratory data analysis.
- **`mdsgui`**: This provides a way to reduce the dimensionality using classical or metric multidimensional scaling.
- **`dimredgui`**: This GUI has functions for principal component analysis and nonlinear dimensionality reduction.
- **`kmeansgui`**: This implements the *k*-means method for finding groups in data.
- **`agcgui`**: Using this GUI, one can cluster the data using agglomerative clustering methods. It includes the classical and model-based agglomerative clustering.
- **`mbcgui`**: This provides an interface to the model-based clustering methodology.

Please note that there is one additional GUI. We do not list it above because it cannot be used as a stand-alone tool. This is the **`brushgui`**, which can only be accessed through the **`gedagui`**. It allows one to brush observations and to color groups and view the same observations being highlighted in open linkable plots.

We provide a detailed description of each separate GUI in the following sections.

Notation

We assume that for most methods the input will consist of an *n* by *p* matrix of observations. Each row of this data matrix represents a *p*-dimensional data point. Multi-dimensional scaling can be used when the input matrix is an *n* by *n* interpoint distance matrix.[1]

A **`bold Courier`** font will be used to indicate MATLAB functions, commands, GUI, and control names.

A summary of the notation is provided here:

- **X** represents the *n* by *p* data matrix.
- *n* is the number of observations.
- *p* is the number of dimensions.
- *d* is the number of dimensions in a reduced-dimensionality space.

Actions and Outputs of GUI Controls

In general, actions are not performed by the computer unless the user clicks on a button. So, if the user changes a parameter or other option, then the associated button must be pushed to run the process again.

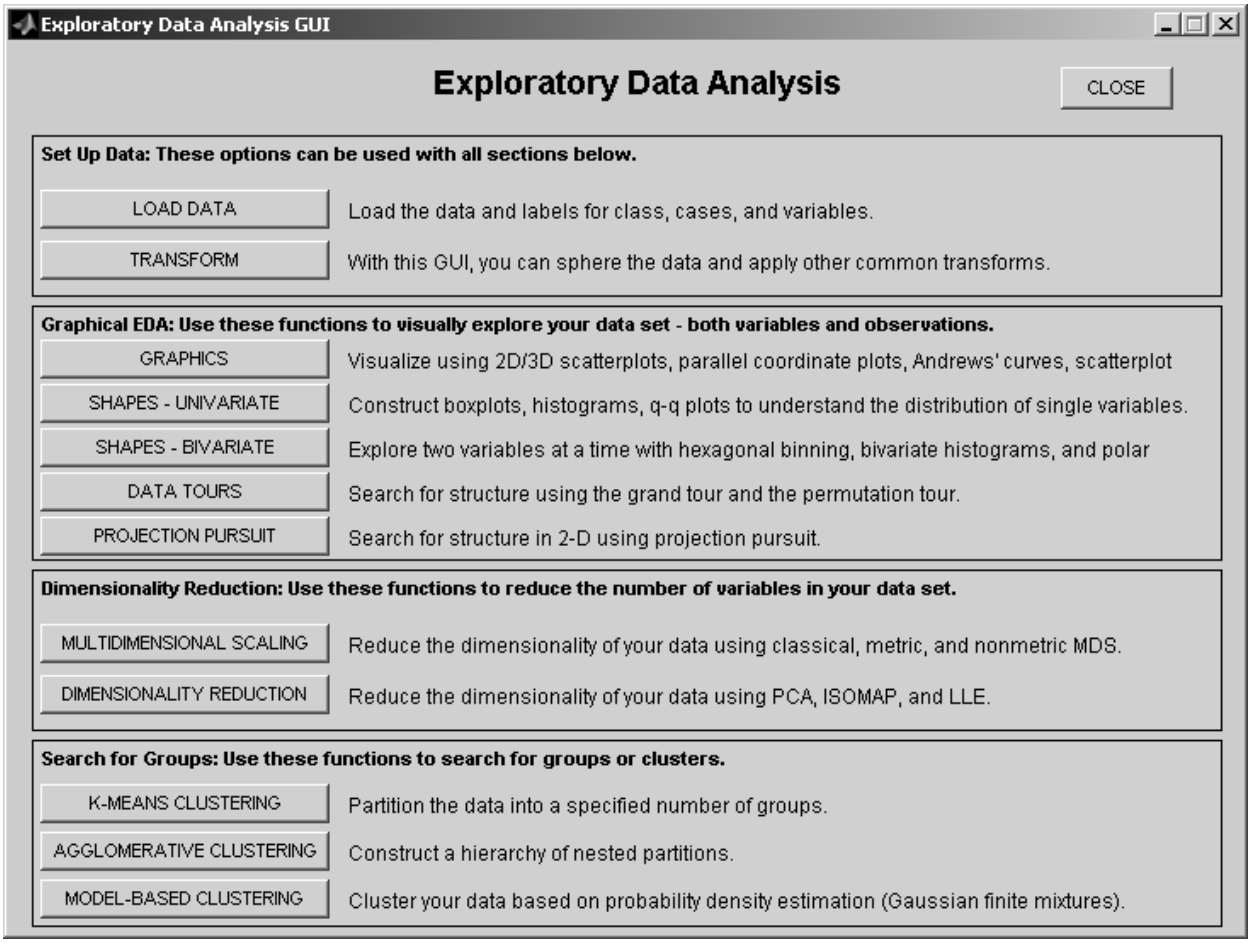
In most cases, outputs, data transformations, reductions, etc. are over-written when the process is repeated. The option of saving the output to the workspace for later processing is provided.

1. The *ij*-th element of the interpoint distance matrix represents the distance/dissimiarity between the *i*-th and *j*-th observations.

Exploratory Data Analysis GUI

To access from the command line use: **`edagui`**

Each of the bottons will bring up the associated GUI. A screenshot of the **`edagui`** is shown here.



This is the main entry point for the EDA GUI Toolbox. You do ***not*** have to use this to access the various GUIs. Each of the associated GUI tools can be opened separately, and appropriate auxilliary GUIs can be accessed from them. For example, one needs to be able to load data, so the **`Load Data GUI`** can be invoked from all GUI windows.

Besides providing access to the various GUI tools, the **`edagui`** window also gives the user a summary of the capabilities that are available in the toolkit, so in that sense, it serves as a type of guide.

We also tried to group these GUIs by functionality or purpose, as outlined below:

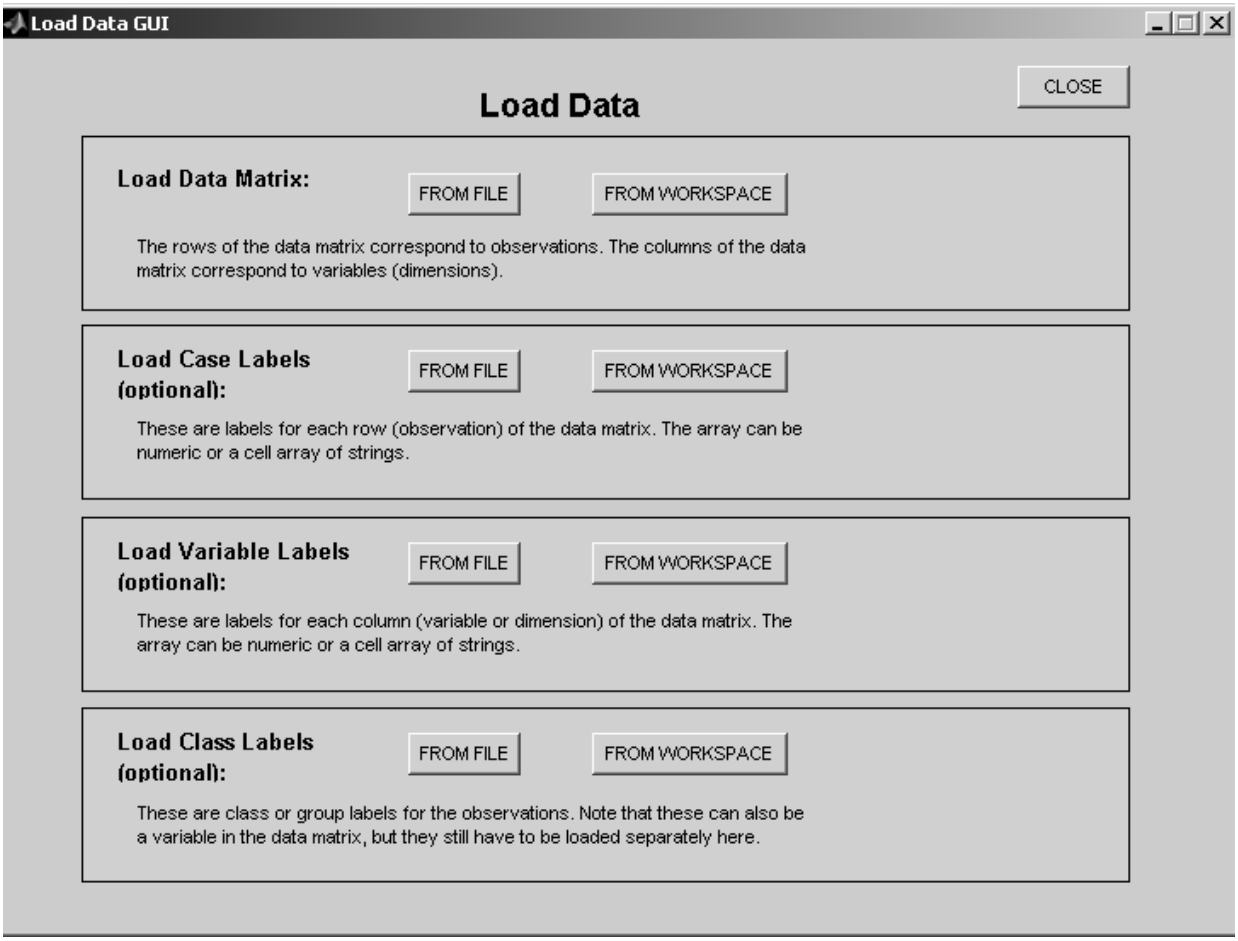
- **Setting up the data**: These are the GUIs for loading and preparing the data for analysis.
- **Graphic-based EDA**: These provide ways to visually explore the data.
- **Dimensionality reduction**: These are the tools for reducing the dimensionality using linear and nonlinear methods.
- **Searching for groups**: Clustering or unsupervised learning is often a main goal of EDA.

Load Data GUI

To access from the command line use: **`loadgui`**

This GUI can be accessed from most of the other GUIs (except for the **`Transform Data GUI`** and the **`Brushing and Labeling Data GUI`**) by pushing the appropriate button.

This GUI provides a means to load up the data for analysis. Once the data are loaded, they are then available to other GUIs. The screenshot of the GUI is shown here.



We can load information from either the MATLAB workspace or from an ASCII file. If the information is in a MATLAB **`.mat`** file, then you must first load it into the workspace and then use **`loadgui`** to load it for use by the GUIs.

Load Data Matrix:
The system assumes that the data are arranged in matrix format where the rows represent observations (cases) and the columns correspond to variables. So, this is an *n* by *p* matrix. You must load a data set before using the other GUIs. If you already have a data set loaded, then you will get a dialog box asking if you want to replace the data set that is already there.

Note that one can load the *n* by *n* interpoint distance matrix for the multidimensional scaling GUI.

Load Case Labels:
Loading case labels is optional. These would be labels that correspond to each observation or row in the data matrix. This is sometimes the case when we have rather small data sets.

For example, the **`cereal`** data has information/measurements on 77 types of cereal. Each type of cereal is an observation. So, we could use this to load up the name of the cereal.

If the user does not load up case labels, then the system uses 1 through *n* as the default.

Note that the information can be in a numeric array or a cell array of strings.

Load Variable Labels:
Loading variable labels is optional. Variable labels are used more often than observation names. For example, in the case of the **`cereal`** data, we have variables such as protein (grams), fat (grams), fiber (grams), etc.

If the user does not load up variable labels, then the system uses 1 through *p* as the default.

Note that the information can be in a numeric array or a cell array of strings.

Load Class Labels:
Loading class labels is also optional. In some applications, we already know class or group labels for our observations. For example, we know the manufacturer of the cereals in the **`cereal`** data. Sometimes the data matrix has one of the columns that contains the class label. The user would need to save that in a separate array and load it using the buttons in this group.

There are no default values for class labels.

IMPORTANT NOTE: We decided to store all data associated with the GUIs in the root's User Data. This means that it is always available outside the GUIs and is available even when the GUIs are closed. One can access this information by using the following at the command line:

```
ud = get(0,'userdata');
```

This has several advantages when debugging, and the advanced user can take advantage of this to extract data from the GUIs.

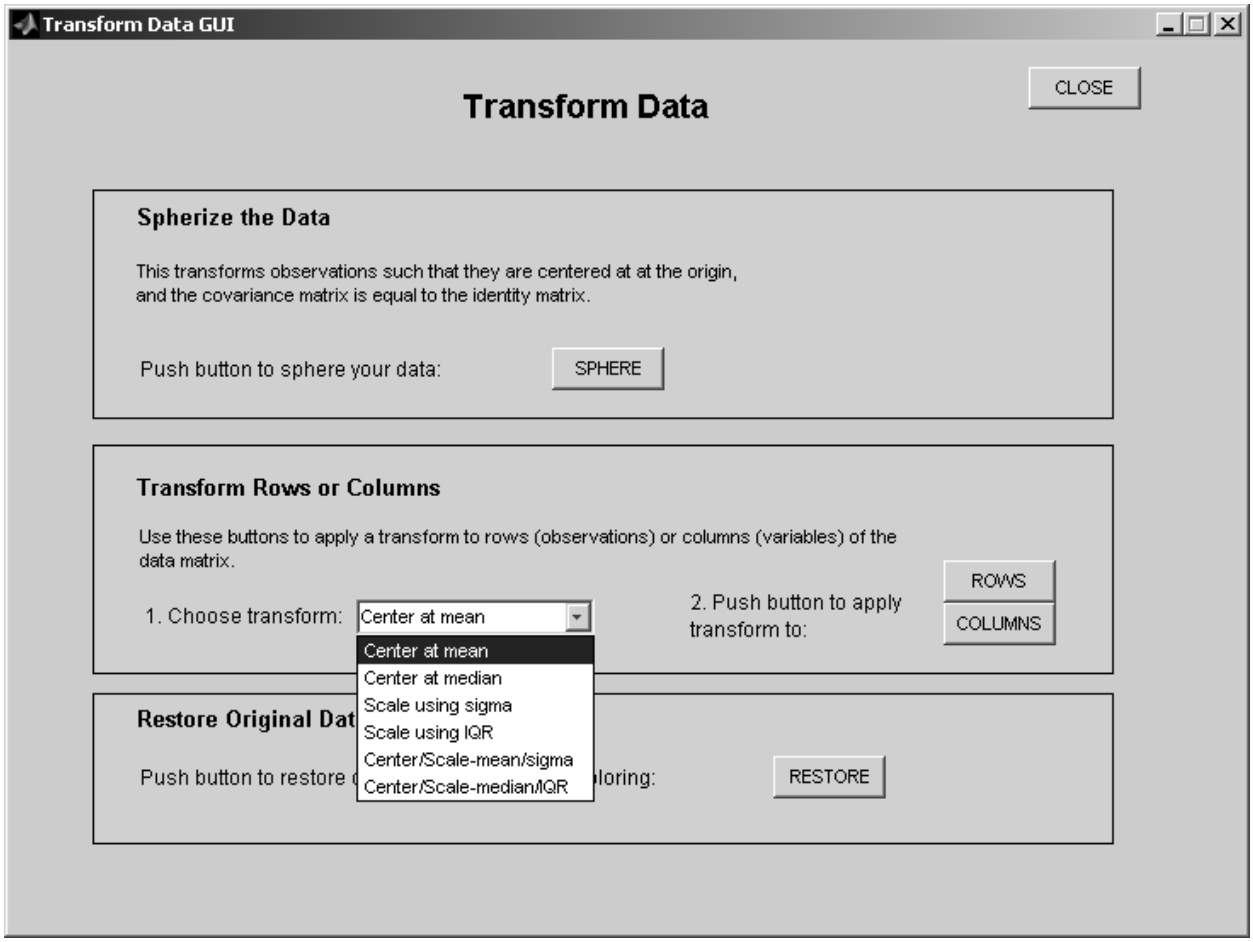
Transform Data GUI

To access from the command line use: **`transformgui`**

This GUI can be accessed from most of the other GUIs using the appropriate button, locagted in the upper left corner.

We often need to transform the data before we explore it. Some typical transforms include translating the observations, scaling, or spherizing. This GUI allows one to restore the original data set, if desired.

A screenshot of the GUI is shown here:



Spherize the Data
This transforms the data such that the resulting data are centered at the origin, and the covariance matrix is equal to the identity matrix. Simply push the button to spherize the data.

Transform Rows or Columns
This allows one to apply a translation and/or a scaling to the data. This transformation can be applied to either the rows or the columns of the data matrix **X**.

This toolbox expects the data matrix to be of the form where rows correspond to observations and columns represent variables. So, typically these transforms would be applied to the observations (rows). However, we assumed that

some users would have data sets where transformations might need to be applied to the columns (e.g., are gene expression data from microarrays). So, we provide that capability with this GUI.

Restore Original Data

Note that applying the transforms makes the transformed data the current data set, and all subsequent analyses will be performed on the transformed data. You can restore the original data set using the button provided in this area of the GUI.

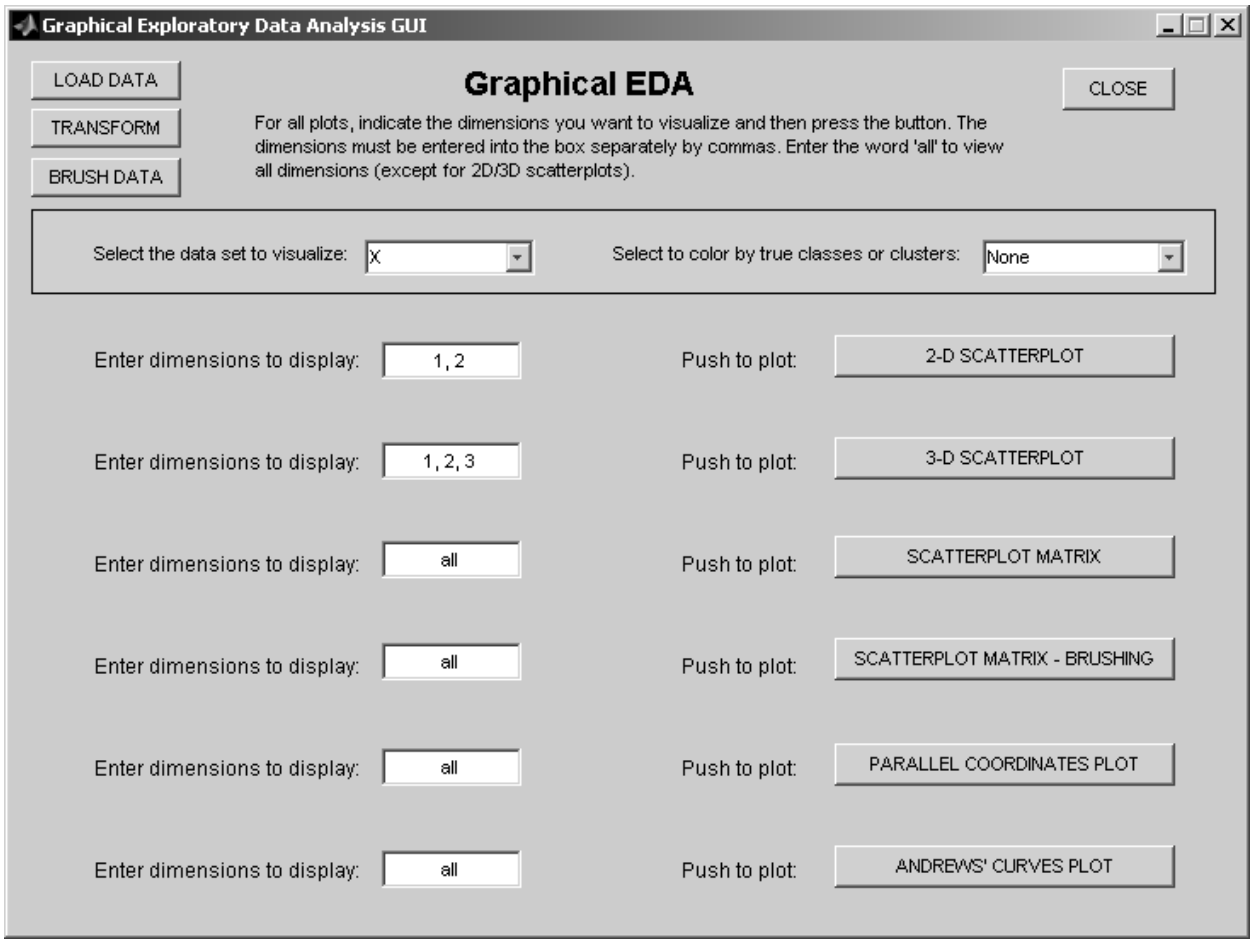
Graphical Exploratory Data Analysis GUI

To access from the command line use: **`gedagui`**

This GUI can be accessed from several other GUIs:

- Multi-Dimensional Scaling
- Projection Pursuit EDA
- K-Means Clustering
- Agglomerative Clustering
- Model-Based Clustering

This provides access to several displays useful for exploratory data analysis. A screenshot of the GUI is given here.



The buttons in the upper left corner are fairly standard and are available from most GUIs. The exception is the **`Brush Data`** button. This brings up an auxilliary GUI that allows you to brush data and have the same points higlighted in all applicable open plots. The **`Brushing and Labeling GUI`** can only be opened through this GUI.

Where appropriate, you can choose the dimensions (columns) that you want to work with. You must enter the numbers separating using spaces or commas. If you want to use all of the dimensions, then just use the word **`all`**. Default values are provided.

SPECIAL CASE: It should be noted that there are special cases if you are visualizing the data after you have reduced the dimensionality using ISOMAP or Principal Component Analysis (PCA) from the **`dimredgui`**. If you want to

visualize these data, then you must make some more choices regarding dimensionality in the **gedagui**. We did this to allow the flexibility of viewing all available dimensions. We cannot provide this capability for the LLE methods.

There is a popup menu that allows you to choose what data set you want to display. For example, if you used something to reduce the dimensionality or did some clustering, then this will show up in the menu. **X** refers to the original data set.

If applicable, you can choose to color the displays according to the following.

- **Color by groups**: If you load class labels, then this choice will color the observations using this label.
- **Color by cluster IDs**: This can be used to explore the results of your clustering. Choose this to color your data by cluster ID.

Note that you can have more than one plot or graphic open at the same time.

If you have plots open and you close the GUI, then a popup box will appear informing the user that all open grahics will be closed. You can choose to proceed (and all plots will be closed) or you can cancel. This is important if you need to export the plots to some other document.

2-D Scatterplot

This displays the usual 2-D scatterplot. Enter the dimensions you would like to visualize using numbers that are separated by commas or spaces. Push the button to display the plot.

3-D Scatterplot

You can use this button to create a 3-D scatterplot. Enter the dimensions you would like to visualize, separating the numbers by commas or spaces. Then push the button to display the plot.

Scatterplot Matrix

A scatterplot matrix shows all pairwise scatterplots for the selected dimensions. The MATLAB **plotmatrix** function is called when you push the button. This function displays histograms of the columns (variables) of the data matrix **X** along the diagonal plot windows.

Enter the dimensions you would like to plot, separated by commas or spaces. The default is to plot all of them.

Scatterplot Matrix - Brushing

This button creates a scatterplot matrix using a function that comes with the EDA Toolbox. One can construct a brush (i.e., a box) in one of the scatterplots by clicking and dragging the mouse.

To grab the brush, just click on the box, hold down the left mouse button, and drag the box over the points. Observations inside the box will be highlighted, along with corresponding points in other plots.

There are several modes available to the user. To access other options, right-click in any of the diagonal plot boxes to get a shortcut menu. You will see the following menu:

- **Highlight** (default): This means brushed observations will be highlighted with color.
- **Delete**: This is not available yet. However, future versions will allow the user to delete observations from the plot.
- **Transient**: This mode means that observations that are inside the brush are highlighted, while observations outside the brush are not.
- **Lasting**: This mode highlights observations inside the brush. Once painted, they remain so.
- **Undo**: This mode erases the highlighting for observations that are brushed.
- **Delete Brush**: This deletes the brush, but keeps the observations highlighted. This is useful when one wants to use the graphic in a document or presentation.
- **Reset Figure**: This deletes the brush and resets the figure to the original form.

Parallel Coordinates Plot
Parallel coordinate plots use axes that are parallel rather than orthogonal, enabling us to visualize more than three dimensions. As with the other plots, you can specify what dimensions you would like to plot.

In parallel coordinate plots, each observation is represented by a broken line segment connecting all of the axes. The value of the observation is indicated by the position of the line along the axis. Parallel coordinate plots allow one to visualize pairwise values, but it is difficult to understand the relationship between variables that are not on consecutive axes (e.g., the relationship between x_1 and x_5). So, we provide a tour or animation that permutes the axes and replots the observations, allowing one to visualize all possible pairwise relationships of the variables. This can be accessed via the **`Data Tours GUI`** (**`tourgui`**).

Using the **`Color by Groups`** or **`Color by Cluster IDs`** will color the lines according to the class labels or clusters.

Andrews' Curves Plot
Andrews' curves are similar to parallel coordinate plots in that they show observations as lines. The basic idea behind Andrews' curves is to transform the observation into a line by projecting the observations onto a set of orthogonal basis functions, usually sines and cosines. One nice thing about Andrews' curves is that they preserve distances, so means and variances are also preserved.

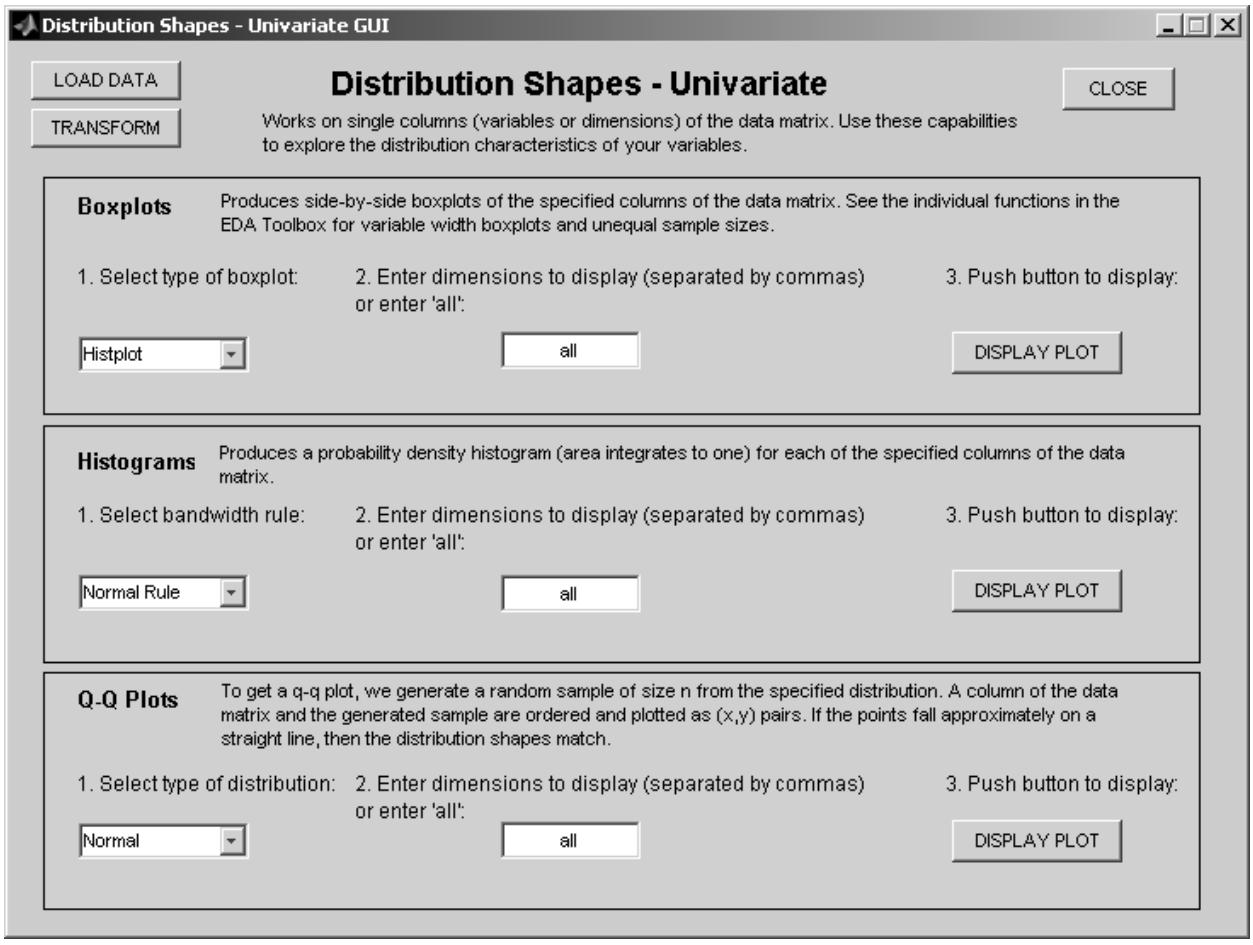
One of the problems with Andrews' curves is that the resulting curve is dependent on the order of the variables. Variables that appear sooner (e.g., x_1) in the sequence, have more influcence on the appearance of the curve. In other words, if we change the order, we might change the curve. We provide a type of tour that re-orders the variables and replots the Andrews' curves. This is available on the **`Data Tours GUI`** (**`tourgui`**).

References
Andrews, D. (1972) 'Plots of high-dimensional data,' *Biometrics*, 28: 125 - 136.
Martinez, W., and A. Martinez (2004) *Exploratory Data Analysis with MATLAB*, CRC Press.
Wegman, E. J. (1990) 'Hyperdimensional data analysis using parallel coordinates,' Journal of the Americal Statistical Association, 85: 664 - 675.

Distribution Shapes - Univariate GUI

To access from the command line use: **`univgui`**

This provides access to various functions for understanding the distributions of the variables. A screenshot is given here.



This GUI allows one to visually explore the distributions of each variable alone. We provide access to many of the common techniques and methods for looking at univariate distributions. We provide more detail on these capabilities below.

Boxplots
This section of the GUI includes four types of boxplots. Note that the basic boxplot and the notched boxplot require the Statistics Toolbox from The MathWorks, Inc.

If more than one dimension is chosen, then the boxplots are shown side-by-side; each column (or variable) is indexed along the horizontal axis. These will always be labeled by column number, not variable names (even if you loaded them).

The four types of boxplots are described next.

- **`Regular`**: This is the basic boxplot, where the ends of the box correspond to the first and third quartiles, the line inside the box is the median, and the ends of the whiskers (lines extending from the boxes) show the extent of the data. Possible outliers are indicated by symbols.
- **`Notched`**: This includes notches in the boxes. The notches correspond to a robust estimate of the uncertainty in the median, allowing one to compare distributions. See the MATLAB **`Help`** file on the **`boxplot`** function for more information.
- **`Box-percentile`**: This constructs a type of boxplot where the width of the box indicates encodes the same information one has in a percentile plot.
- **`Histplot`**: This displays a type of boxplot where the width of the box is proportional to the estimated density.

Histograms
This part of the GUI creates histograms for each of the variables (columns of **`X`**). These are probability density histograms, so the areas represented by the bars is equal to one.

When you select more than one dimension, you will get a plot matrix with a histogram corresponding to each of the specified dimensions. One can create one of these plots for each of the ruels and compare them. Note that one can distinguish the plots by looking at the figure title bar.

We provide three choices for selecting the bandwidth: Normal Rule, Freedman-Diaconis Rule, and Sturge's Rule. These are given by

- Normal Reference Rule: The histogram bandwidth is given by $3.5 \times \sigma \times n^{-1/3}$.

- Freedman-Diaconis Rule: The histogram bandwidth is given by $2 \times IQR \times n^{-1/3}$, where IQR is the interquartile range.

- Sturge's Rule: The number of bins is given by $1 + \log_2 n$.

Q-Q Plots
It is sometimes instructive to look at the distribution of the data using quantile-quantile plots. The idea is to do a scatterplot with the quantiles of the data set versus the quantiles of the reference distribution. If the two samples come from the same distribution, then the plot will be approximately linear.

Note that we employ a somewhat non-standard method by plotting against random samples generated from the refernce distribution (e.g., normal, exponential, etc.). It is customary to plot against the theoretical quantiles of the reference distribution. However, it is often the case that one wants to determine whether the two samples are from the same distribution, and quantile-quantile plots can be used. This is the same thing we are doing here.

The user can select the reference distribution from the popupmenu. We provide the following reference distributions: normal, exponential, gamma, lognormal, chi-square,uUniform, and Poisson.

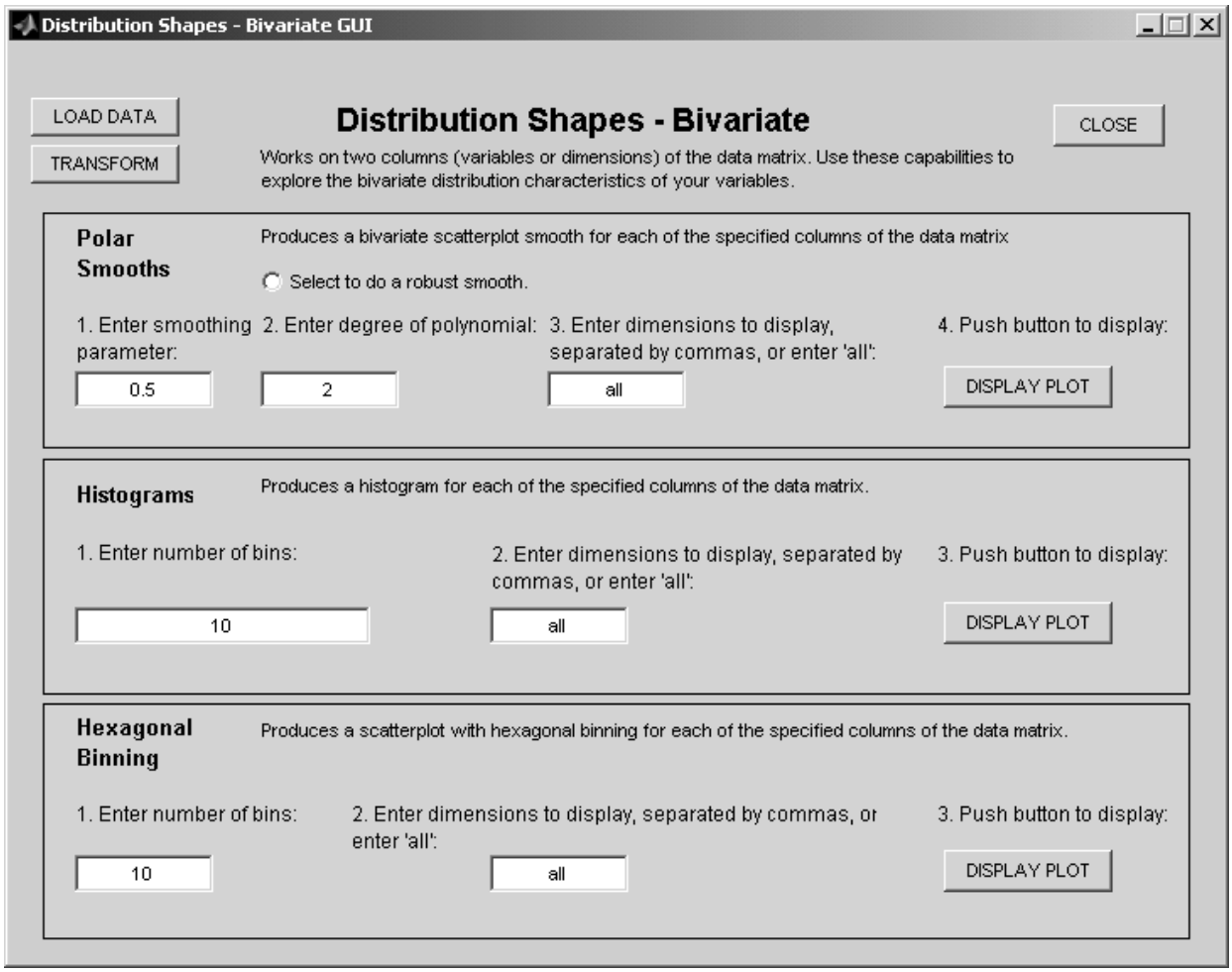
The user can have serveral of these plots open at the same time, so one can make comparisons for different reference distributions. The figure title bars will contain the name of the distribution.

References
Cleveland, W. S. (1993) *Visualizing Data*, Hobart Press.
Esty, W. W. and J. D, Banfield (2003) 'The box-percentile plot,' *Journal f Statistical Software*, 8, www.jstatsoft.org
Scott, David (1992) *Multivariate Density Estimation: Theory, Practice, and Visualization*, Wiley.

Distribution Shapes - Bivariate GUI

To access from the command line use: **`bivgui`**

This provides access to the main capabilities that are used to understand the distributions of pairs of variables (two columns of the data matrix **X**). A screenshot is given here.



This GUI allows one to visually explore the distributions of two variables. We provide access to many of the common techniques and methods for looking at bivariate distributions. More detail on these capabilities is given below.

Polar Smooths

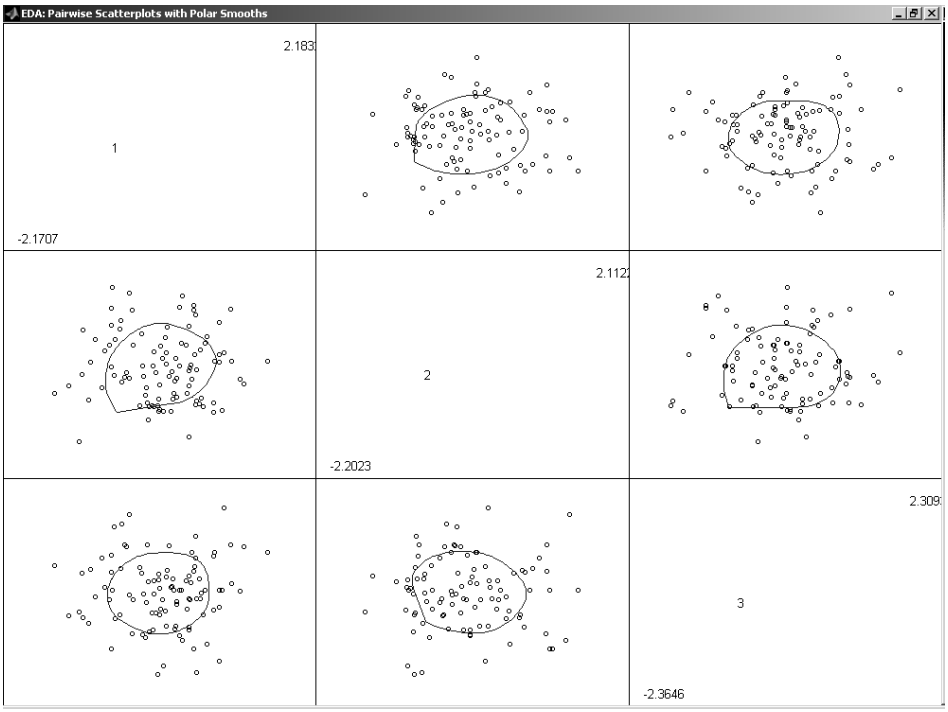
Some users might be familiar with the scatterplot smoothing method called *loess* that shows a nonparametric relationship between two variables when one is interested in $y = f(x)$. In other words, we want to know how y depends on x. We do not implement this type of smoothing in the EDA GUI toolbox.

In our case, we do not have a predictor or response variable, but rather we are interested in understanding the distributional relationship between the two variables. The idea behind the loess scatterplot smooth has been adapted to the the bivariate distribution case. It is called *polar smoothing*.

We provide the ability to create bivariate polar smooths to help one understand these relationships. A plot matrix with pairwise bivariate polar smooths is provided for the designated dimensions (coloumns of the data matrix **X**). The user first enters the required information, as follows:

- Smoothing parameter: This is a value between 0 and 1.
- Degree of the polynomial: Enter 1 for linear and 2 for quadratic.
- Number of dimensions: Enter the dimensions to be used in the plots.
- Robust smooth: Select this if you want to construct a robust smooth that is not affected by outliers.

Push the **`Display Plot`** button to obtain a pairwise scatterplot with the polar smooths superimposed over the scatterplot. The minimum and maximum values for each variable are shown in the corners of the diagonal boxes. For example, x_1 has a minimum value of -2.1707 and a maximum value of 2.183.

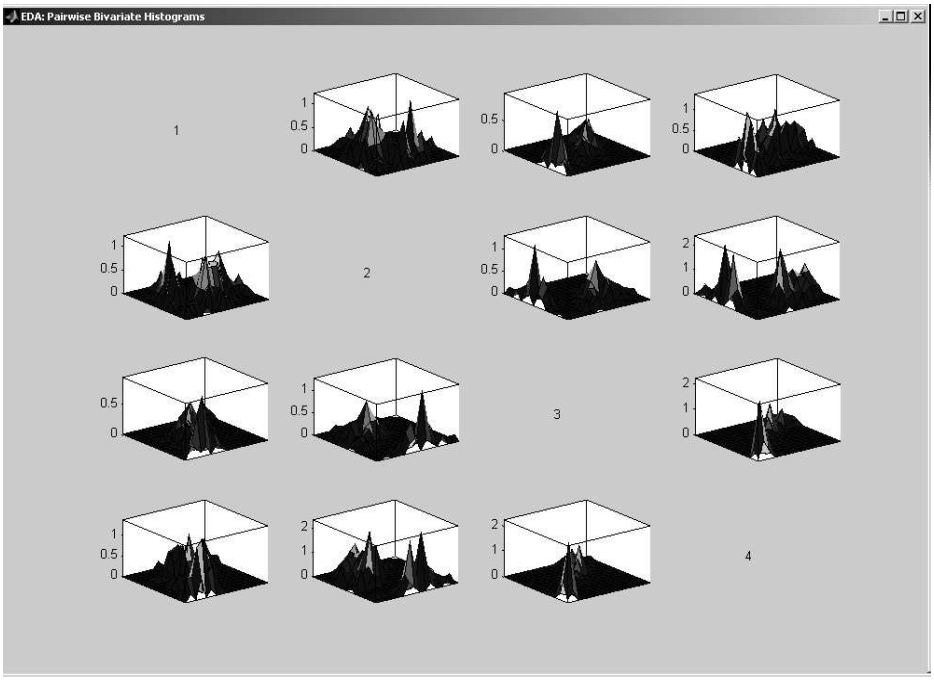


Histograms

As with polar smooths, one can create bivariate histograms to explore the bivariate distribution shapes. This section will provide bivariate histograms for the dimensions specified in the text box. The required information is:

- Number of bins: Enter the number of bins for each dimension, separated by commas or enter just one value that will be used for all dimensions.
- Dimensions to plot: Enter the dimensions to be used in the plot.

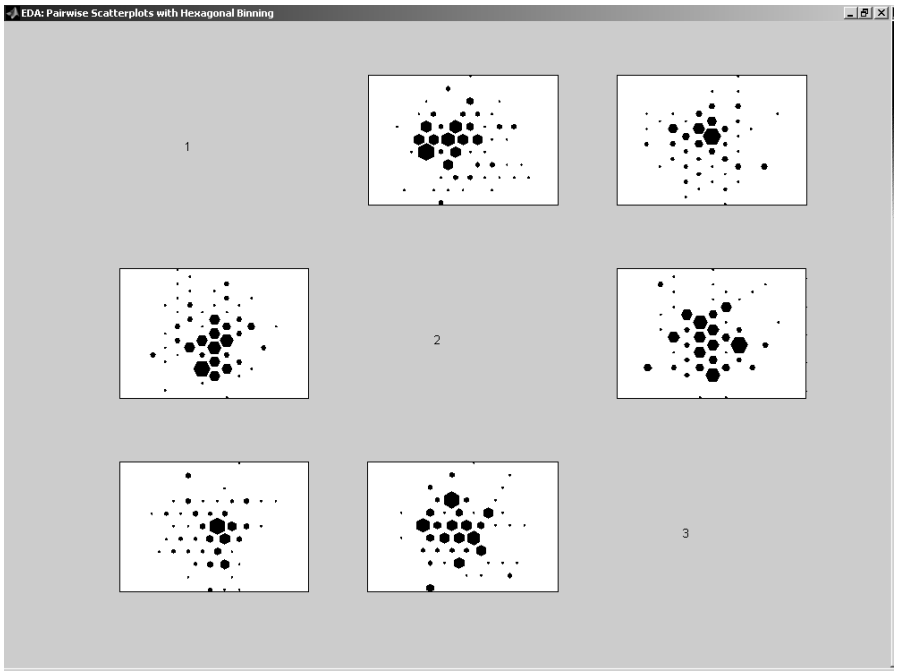
Push the **`Display Plot`** button to get the desired display. An example is given below.



Hexagonal Binning

Hexgonal binning provides similar information regarding the bivariate distribution of the data. In some sense, one can think of this as a type of scatterplot smooth known as scatterplot binning. The size of the hexagonal bin is an indication of the number of observations in the bin. We enter the number of bins and the dimensions one would like to see in the plot.

Push the **`Display Plot`** button to get the graphic. An example is given below.



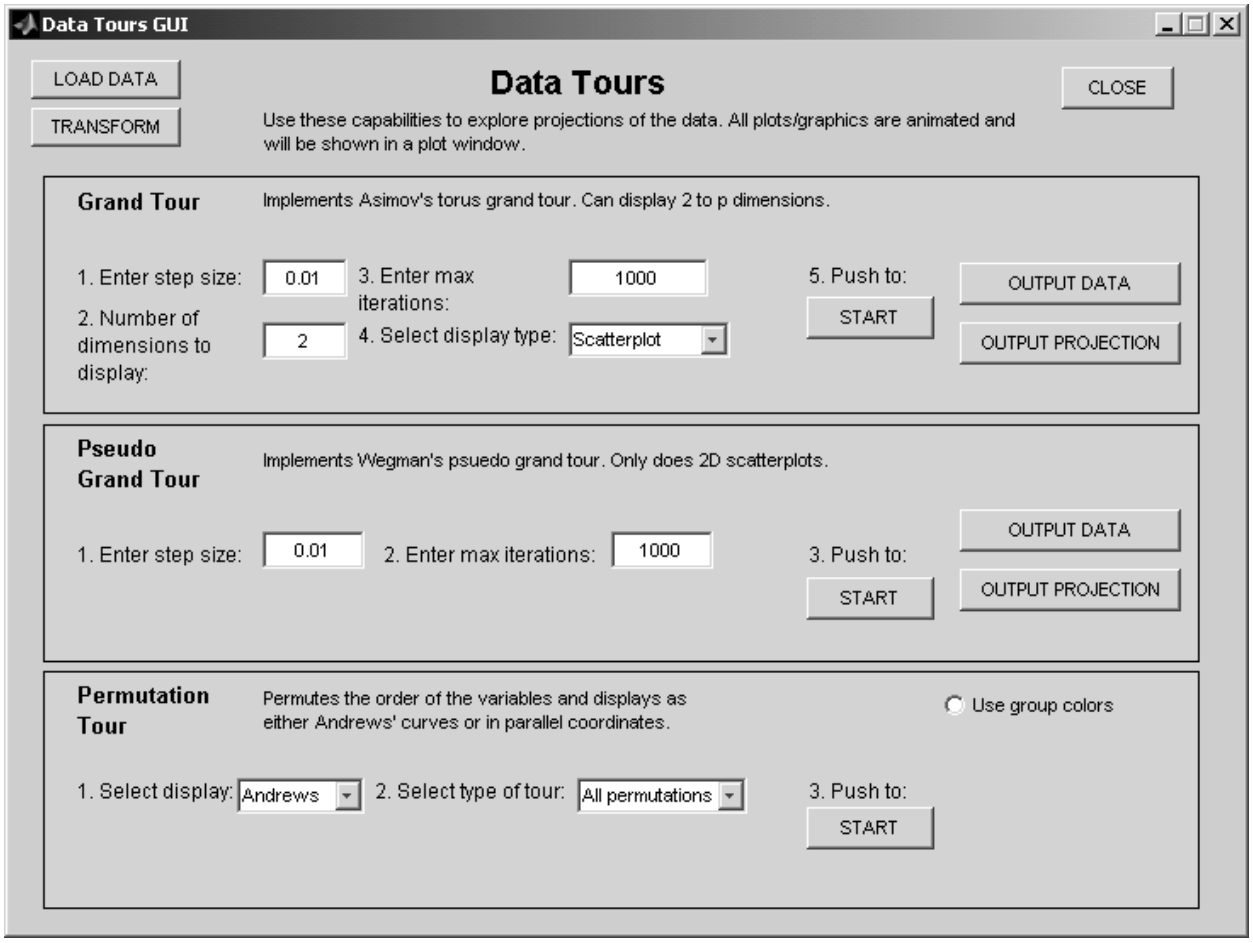
References

Carr, D. B., Littlefield, R. J., Nichologson, W. L., and Littlefield, J. S. (1987) 'Scatterplot matrix techniques for large n,' *Journal of the American Statistical Association*, 82, 424 - 436.
Cleveland, W. S. (1993) *Visualizing Data*, Hobart Press.
Scott, David (1992) *Multivariate Density Estimation: Theory, Practice, and Visualization*, Wiley.

Data Tours GUI

To access from the command line use: **`tourgui`**

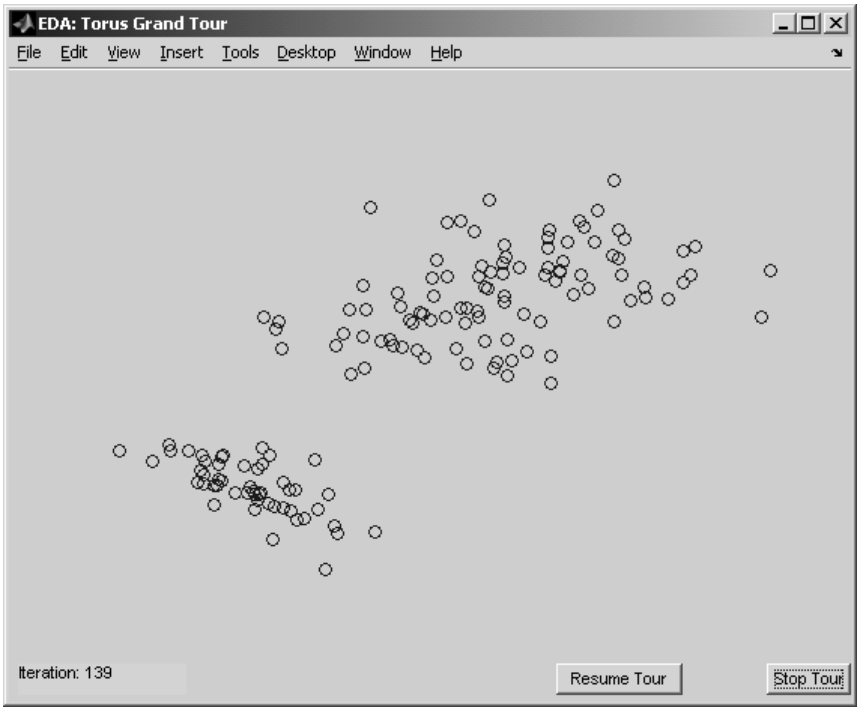
This provides access to several data touring capabilities. Data tours are a way to see the data from different viewpoints in the search for structure and important patterns. In particular, this allows one to do a torus or a pseudo grand tour of the data, as well as a perumuation tour. Tour are animations with some interactions by the user. A screenshot of the GUI is given here.



Grand Tour
Most of the required inputs for the grand tour are self-explanatory. To get a more 'continuous' tour, use a smaller step size. The tour display can be a scatterplot (2-D), Andrews' curves, or parallel coordinates.

The **`Output Data`** button will save the projected data (for the current tour step) to the workspace, and the Output Projection button will save the projection matrix to the workspace.

A snapshot of one frame of the tour is shown here:



The user can pause in the tour by hitting the **`Stop Tour`** button. To continue the current tour, press the **`Resume Tour`** button.

NOTE: To start a brand new tour, perhaps because you changed tour values, you must close the current tour window first.

Pseudo Grand Tour

A pseudo grand tour was proposed by Wegman and Shen (1993) as an alternative that addresses some of the issues with a grand tour. They showed that it is not a real grand tour, hence the name *pseudo* grand tour. As with the grand tour, a smaller step size provides a more 'continuous' tour.

The pseudo grand tour will display 2-D scatterplots only.

Permutation Tour

One of the problems with Andrews' curves and parallel coordinates is the dependence on the order of the variables. In other words, if one changes the order of the variables, then the display output might change. To help address that issue, we implemented a permutation tour for these types of plots.

The user can select either Andrews' curves or parallel coordinates from the pull-down menu.

The other menu allows one to choose between two tours. One tour goes through all permuations, replotting after each permutation. This can result in a very long tour, depending on the dimensionality. The other tour follows the minimal permutation tour of Wegman (1990).

References

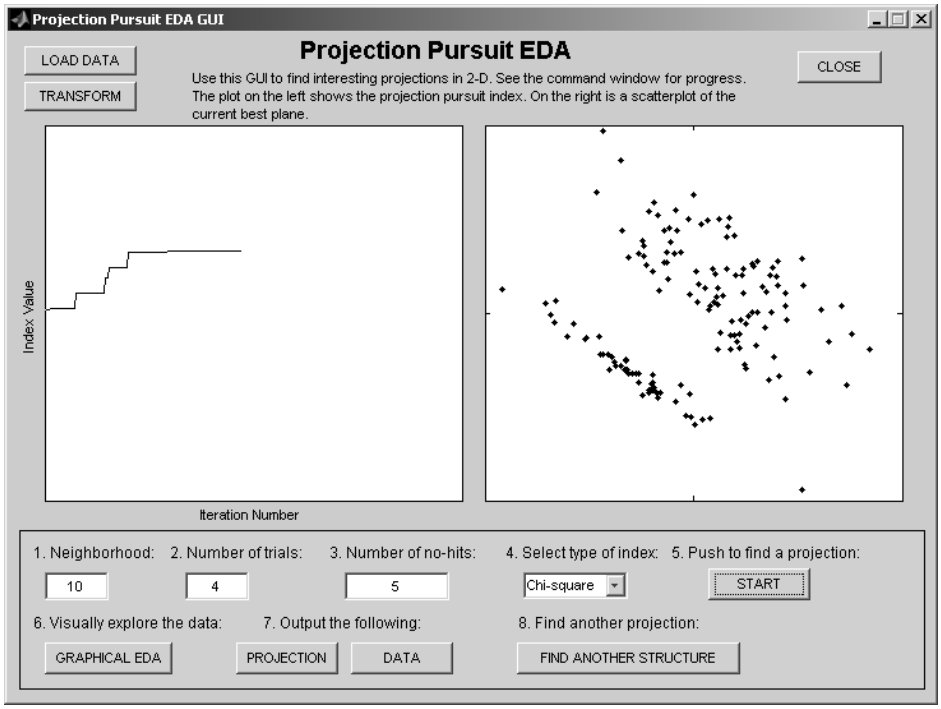
Andrews, D. (1972) 'Plots of high-dimensional data,' *Biometrics*, 28: 125 - 136.
Asimov, D. (1985), 'The grand tour: A tool for viewing multidmensional data,' *SIAM Journal on Scientific and Statistical Computing*, 6: 128 - 143.
Martinez, W., and A. Martinez (2004) *Exploratory Data Analysis with MATLAB*, CRC Press.

Wegman, E. J. (1990) ‘Hyperdimensional data analysis using parallel coordinates,’ *Journal of the Americal Statistical Association*, 85: 664 - 675.
Wegman, E. J. (1992) ‘The grand tour in k-dimensions,’ in Page, C., and LePage, R. (eds), Computing Science and Statistics, Proceedings of the 22nd Symposium on the Interface, Springer-Verlag, New York, 127 - 136.
Wegman, E. J., and J. Shen. (1993) ‘Three-dimensional Anderews plots and the grand tour,’ in Tarter, M. E., and Lock, M. D. (eds), Computing Science and Statistics: Proceedings of the 25th Symposium on the Interface, p. 284-288.

Projection Pursuit EDA GUI

To access from the command line use: **`ppedagui`**

This allows the user to run Projection Pursuit Exploratory Data Analysis (PPEDA), where one looks for interesting structure in 2-D projections. Here, structure is defined as some departure from normality. We implemented the method by Posse with two projection pursuit indices available to the user. The indices are the chi-square index (Posse) and the moment index. A screenshot of the GUI after PPEDA is performed is shown here. Note that the procedure found sime interesting structure.



Required Data Entry
The following values are required for projection pursuit EDA:

- Neighborhood size: The algorithm searches for new projections within this neighborhood size.
- Number of trials: The PPEDA algorithm is a greedy one, and it depends on the initial starting value. So, it is a good idea in cases like this to perform several searches and pick the best one (projection yielding the highest value for the projection index).
- Number of No Hits: The algorithm searches for new projections within the specified neighborhood. The projection pursuit index is calculated for each projection. If the search does not find a better projection within the number specified in this field, then the neighborhood size is decreased.
- Projection Pursuit Index: These provide a way of measuring the degree of departure from normality. We provide two indices: chi-square index and the moment index. The moment index tends to find outliers.

Running Projection Pursuit
Press the **`Start`** button to find an initial structure. We provide some displays on the GUI to show the progress while the algorithm is looking for structure. The plot on the left shows how the value of the index changes during the search for structure, and the display on the right shows a scatterplot of the data in the current projection.

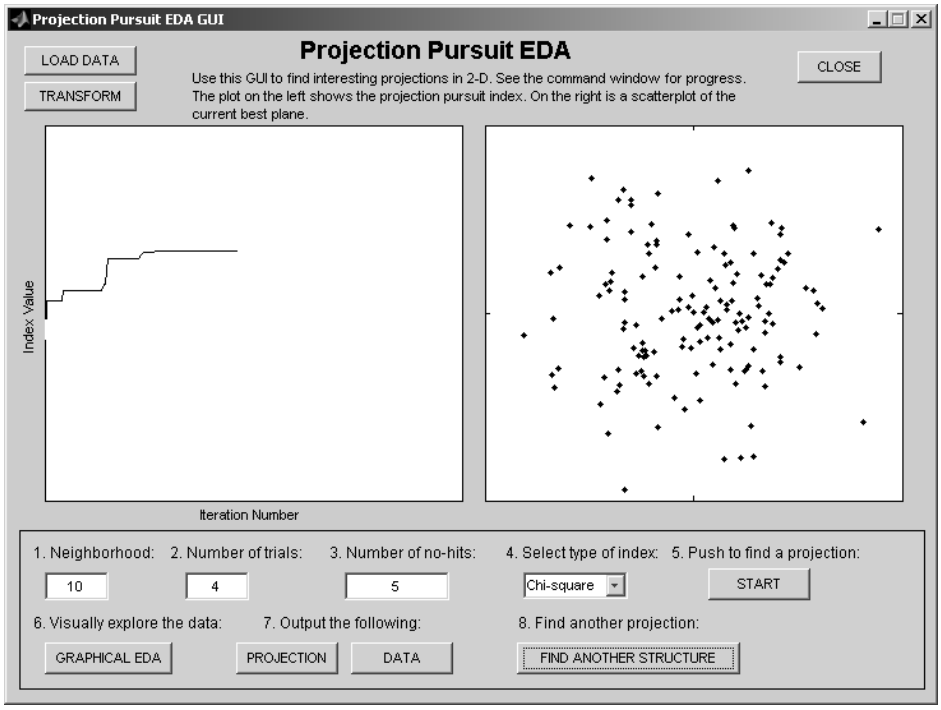
The progress of the method can also be viewed in the command window, where the current trial is indicated, along with the value of the index. An example of how this might look is

```
Trial =1  Index =1.3147
Trial =2  Index =1.3508
Trial =3  Index =1.3854
Trial =4  Index =1.5062
Best projection over all trials has an index of 1.5062
```

The user can visually explore the data (in the best projection) using the GEDA GUI by clicking the Graphical EDA button. The user can also output the current best projection or the projected data to the workspace using the appropriate buttons.

Friedman (REF?) says that one should look for several structures, and his method for doing so is implemented here. Once the user has an initial interesting structure, another one can be found by pushing the Find Another Structure button. It is recommended that the initial projection and/or the projected data be saved to the workspace before proceeding. Any new structure that is found will replace the current one in memory.

Here is a second structure that was found. Note that it is not really an interesting structure, as it looks close to normal. The value of the projection pursuit index was 0.53.



References

Friedman, J. H. (1987) 'Exploratory projection pursuit,' *Journal of the American Statistics Association*, 82: 249 - 266.

Posse, C. (1995) 'Projection pursuit exploratory data analysis,' *Computational Statistics and Data Analysis*, 20: 669 - 687.

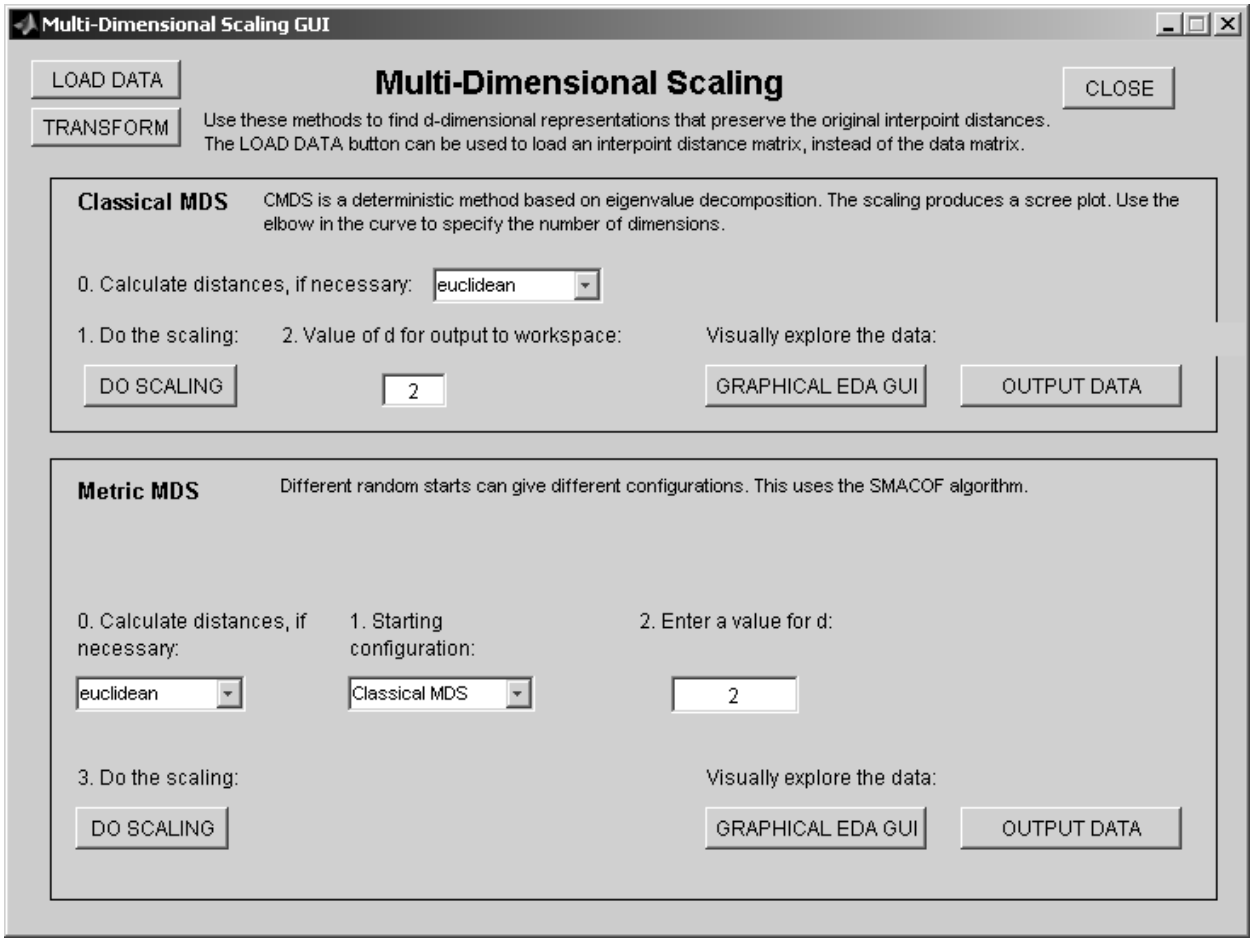
Multi-Dimensional Scaling GUI

To access from the command line use: **`mdsgui`**

This allows the user to reduce the dimensionality of the data using multi-dimensional scaling (MDS) methods. The goal of these methods is to find a configuration for the observations in a *d*-dimensional space, where *d* is less than *p*, such that the distance (or similarity) between points is preserved. In other words, observations that are close together (or similar) in the *p*-dimensional space are also close together in the *d*-dimensional space.

The data required for the MDS methods is actually the interpoint distance (or dissimilarity) matrix, where the *ij*-th element of the matrix indicates the distance between the *i*-th and *j*-th observations. Of course, one can start with the data matrix and find the interpoint distance matrix. Note that one could load an interpoint distance matrix instead of the data matrix, in which case, the distances do not have to be calculated.

The output from MDS can be further explored using graphical EDA or any of the clustering methods included with the GUI Toolbox. A screenshot of the GUI is shown below.

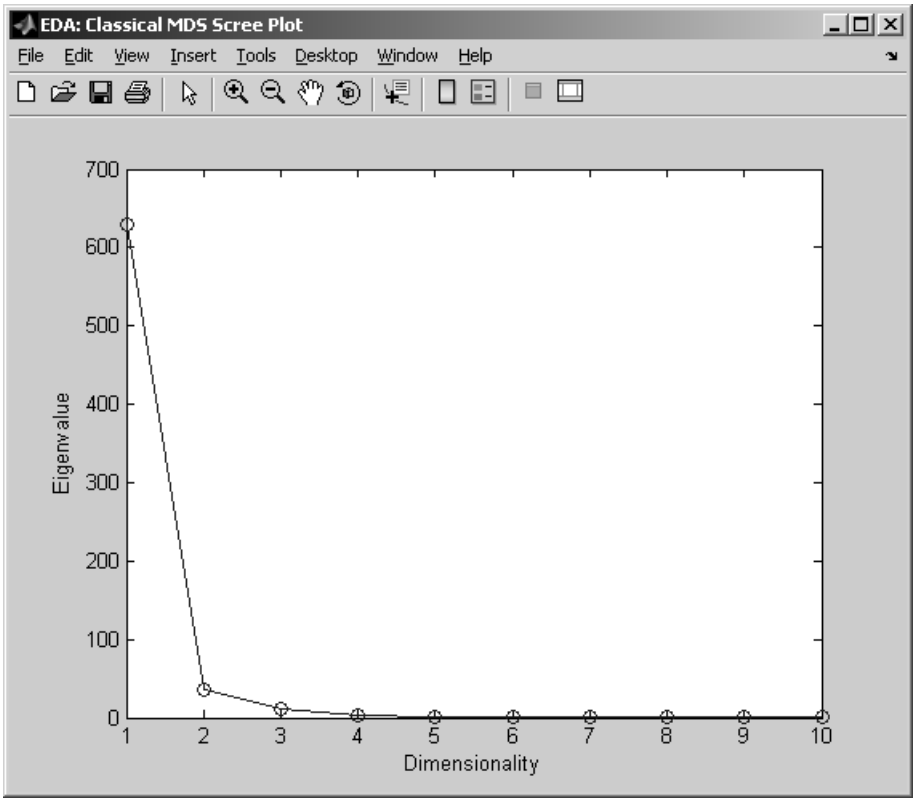


Classical Multi-Dimensional Scaling

The classical multi-dimensional scaling (MDS) method is deterministic; i.e., it is not a greedy algorithm that is dependent on the starting point. In some cases, CMDS is equivalent to principal component analysis (PCA). These conditions imply that one can perform CMDS without first specifying the number of dimensions *d*.

The steps involved in CMDS are described below:

- Step 0: This step is needed if the information loaded using the **`Load Data`** button is the *n* by *p* data matrix. If the information loaded is an interpoint distance matrix, then this step can be skipped.
- Step 1: Push this button to do the scaling. You will get a scree plot and a message box telling you that the scaling was done. The scree plot is shown below, where the eigenvalue is shown as a function of the dimensionality. We look for the elbow in the curve to get a value for *d*. This is similar to what one does in PCA. We have a good elbow at 2, so this is a reasonable number of dimensions to use for this data set.



- Step 2: Enter a value for *d* in this box. This value affects only the output to the data space (**`Output Data`** button). It does not affect the output to the **`Graphical EDA GUI`**.

Displaying Output: The **`Graphical EDA GUI`** button brings up the **`gedagui`**. With classical MDS, the user can view *all* dimensions for $d \le p$. The number of dimensions in the edit box (Step 2) does not affect this capability.

Saving Output to Workspace: When the user clicks on the **`Output Data`** button, a dialog box comes up, where one can specify a variable name. An *n* by *d* matrix is saved to the workspace using the name given in the dialog box. Note that the value used for *d* is taken from the edit box.

Metric Multi-Dimensional Scaling

Several algorithms exist for metric MDS. These require the user to specify the number of dimensions *d* in advance. These are typically greedy algorithms that are dependent on the starting configuration.

Also, note that the configurations are not nested as *d* is changed. In other words, say we run metric MDS for $d = 2$ and $d = 3$. The configuration for two of the dimensions is not necessarily the same across these two runs.

We chose to implement the SMACOF algorithm for metric MDS. The steps for metric MDS are described here:

- Step 0: This step is needed if the information loaded using the **Load Data** button is the *n* by *p* data matrix. If the information loaded is actually an interpoint distance matrix, then this step can be skipped.
- Step 1: Select the starting configuration using the popup menu. One can choose to start from the classical MDS configuration or a random configuration.
- Step 2: Enter a value for the number of dimensions you want for the configuration. This number will be used if the **Graphical EDA GUI** button or the **Output Data** button is pushed.
- Step 3: Push the **Do Scaling** button to run the algorithm. An information window pops up to let the user know that the process has finished. A scree plot is not produced, since it is inappropriate in this case.

As before, one can now explore the data in the new configuration using graphical EDA and also save the configuration to the workspace for further processing.

CAUTION: It is important to note that the user can explore or process only one configuration at a time. The configuration is over-written when the user does another **Do Scaling**.

References

Cox, T. F. and M. A. A. Cox (2000) *Multidimensional Scaling*, 2nd Edition, Chapman and Hall/CRC.
Jackson, J. Edward (1991) *A User's Guide to Principal Components*, Wiley.
de Leeuw, J. (1977) 'Applications of convex analysis to multidimensional scaling,' in J. R. Barra, F. Brodeau, G. Romier & B. van Cutsem (eds.) *Recent Developments in Statistics*, North-Holland, p. 133 - 145.

Dimensionality Reduction GUI

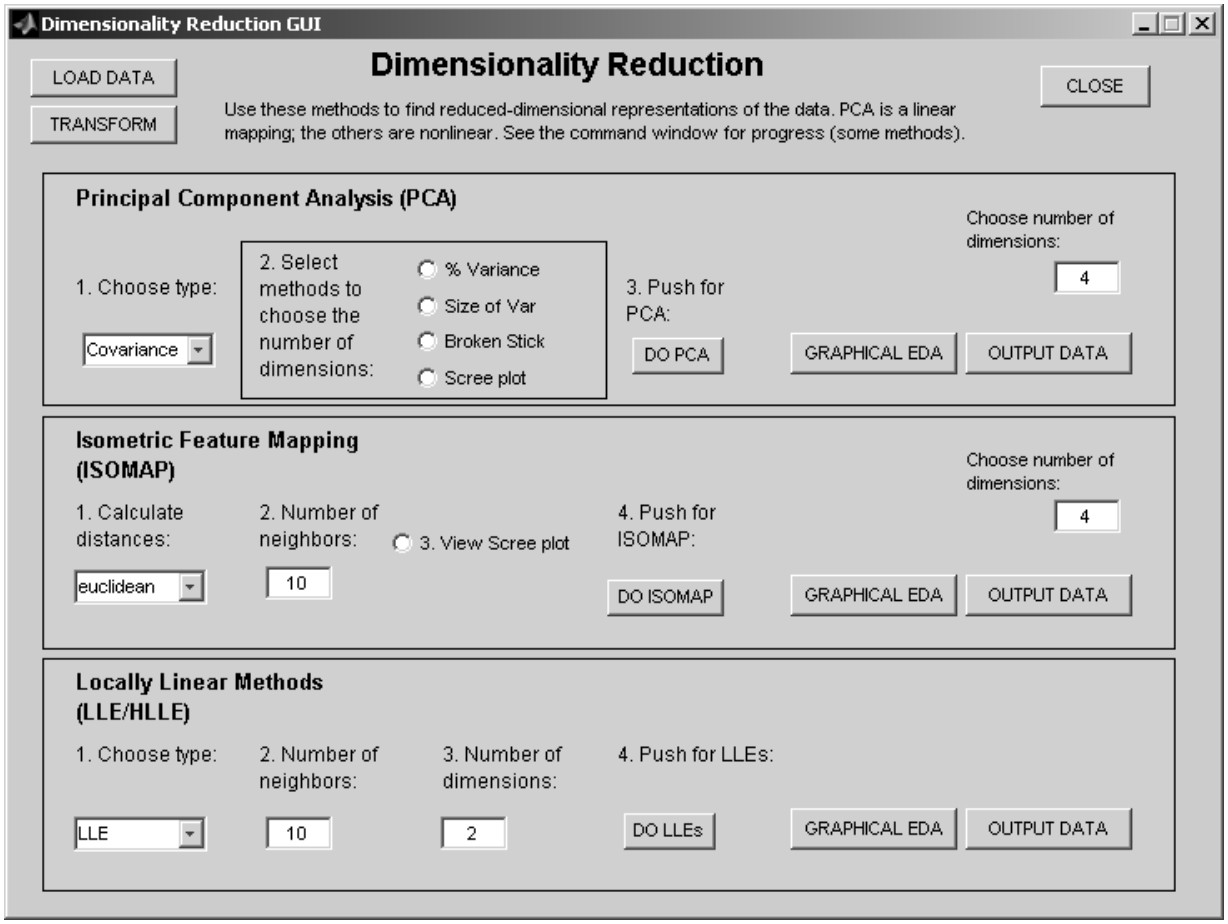
To access from the command line use: **`dimredgui`**

This allows the user to reduce the dimensionality of the data using several methods. These include

- Principal Component Analysis (PCA)
- Isometric Feature Mapping (ISOMAP)
- Locally Linear Methods (LLE and HLLE)

The **`Dimensionality Reduction GUI`** can also be accessed from the clustering GUIs, so one can perform dimensionality reduction prior to any clustering.

The output from MDS can be further explored using graphical EDA or any of the clustering methods included with the GUI Toolbox. A screenshot of the GUI is shown below.



Principal Component Analysis (PCA)

Principal Component Analysis (PCA) is a deterministic method for reducing the dimensionality of a data set, and a lot of results have been developed regarding the theory and output from PCA.

The input for the method is either the covariance matrix or the correlation matrix that is estimated from the data. The output is a new set of *p*-dimensional variables called principal components. The principal components are orthogonal, and they are linear combinations of the original variables.

One can keep all *p* principal components or reduce the dimensionality by keeping $d \leq p$ variables. An important question that must be answered is: How many dimensions should one keep? We provide several tools to help answer this question. We describe these below where we discuss the steps of PCA.

- Step 1: Choose the input for PCA. This can either be the correlation matrix or the covariance matrix. Note that results are different, so one should explore the data using both matrices. For information on the issues with these methods, see some of the references listed below.
- Step 2: Choose the types of tests you would like to run that offer suggestions on the number of dimensions to keep (*d*). You can select all that apply. Please see (REFS) for information on these tests. Tool Tip information for each test is provided when the user leaves the cursor on the text.
- Step 3: Push the **`Do PCA`** button to get the principal components.

Displaying Output: The **`Graphical EDA GUI`** button brings up the **`gedagui`**. With PCA, the user can view *all* dimensions for $d \leq p$. The number of dimensions in the edit box does not affect this capability.

Saving Output to Workspace: When the user clicks on the **`Output Data`** button, a dialog box comes up, where one can specify a variable name. An *n* by *d* matrix is saved to the workspace using the name given in the dialog box. Note that the value used for *d* is taken from the edit box.

CAUTION: It is important to note that the user can explore or process only one configuration at a time. The configuration is over-written when the user does another **`Do PCA`**.

Isometric Feature Mapping (ISOMAP)

Principal component analysis is a linear dimensionality reduction technique. However, in many cases, a better (or at least different!) lower-dimensional space can be obtained using non-linear methods. We provide access to one such method (and MATLAB function) called Isometric Feature Mapping (ISOMAP) that was developed by Tenenbaum, Langford, and de Silva.

ISOMAP is a form of classical MDS, where the interpoint distance matrix contains the estimated geodesic distance between observations. The idea is that such a distance would be more appropriate in determining the proper lower-dimensional nonlinear manifold where the data reside.

Because of the connection with classical MDS (and PCA), the inputs and outputs from this part of the GUI are similar to those methods. One important difference, though, is that the only method for estimating the value for *d* (the number of dimensions to keep) in ISOMAP is the scree plot.

We outline the steps for ISOMAP below:

- Step 1: As with MDS, the input to the ISOMAP methodology is an interpoint distance matrix. The distance used here is the usual one, and the user has a choice of several available in the drop-down menu. The geodesic distance is estimated in the ISOMAP function.
- Step 2: Choose the number of nearest neighbors to use in the estimation of the geodesic distance. Note that the results can be different depending on the value used here. One important point is that one might get disconnected configurations of points in the reduced-dimensionality space, resulting in fewer than *n* points in the *d*-dimensional configuration. A warning box will be displayed when this happens. The user should then make the neighborhood larger or try another distance in Step 1.
- Step 3: Select this radio button to produce the scrre plot, similar to what one gets in PCA.
- Step 4: Push this button do run ISOMAP. The user can follow the progress and view other information in the MATLAB workspace.

Displaying Output: The **Graphical EDA GUI** button brings up the **gedagui**. With ISOMAP, the user can view *all* dimensions for $d \leq p$. The number of dimensions in the edit box does not affect this capability.

Saving Output to Workspace: When the user clicks on the **Output Data** button, a dialog box comes up, where one can specify a variable name. An *n* by *d* matrix is saved to the workspace using the name given in the dialog box. Note that the value used for *d* is taken from the edit box.

CAUTION: It is important to note that the user can explore or process only one configuration at a time. The configuration is over-written when the user does another **Do ISOMAP**.

Note that we use the MATLAB function downloaded from the ISOMAP website: http://isomap.stanford.edu/

Locally Linear Methods: LLE and HLLE

Two other nonlinear methods have been developed and implemented in MATLAB. The first is Local Linear Embedding (LLE); the second is an extension to this called Hessian LLE. Both of these methods are different from PCA and classical MDS in that one must first specify the number of dimensions *d*. In this sense, it is like nonmetric MDS. Also, we do not have access to scree plots or other methods to estimate a value for *d*. The input to these methods is the data matrix, not the interpoint distance matrix.

Hessian LLE is useful when the underlying low-dimensional manifold might not be convex (e.g., it has a hole in it). However, our experience with LLE and HLLE indicate that they are not very robust, as strange configurations can be obtained.

We outline the steps for LLE below:

- Step 1: Select the type of LLE desired: LLE or Hessiann LLE.
- Step 2: Specify the number of nearest neighbors to use for the neighborhood.
- Step 3: Enter the number of dimensions *d*.
- Step 4: Push the button to perform LLE.

As with ISOMAP, the progress of the method and other useful information is shown in the MATLAB workspace.

Displaying Output: The **Graphical EDA GUI** button brings up the **gedagui**. With the LLE methods, the user can view the *d* dimensions for only. The number of dimensions is specified in the edit box.

Saving Output to Workspace: When the user clicks on the **Output Data** button, a dialog box comes up, where one can specify a variable name. An *n* by *d* matrix is saved to the workspace using the name given in the dialog box. Note that the value used for *d* is taken from the edit box.

CAUTION: It is important to note that the user can explore or process only one configuration at a time. The configuration is over-written when the user does another **Do LLEs**.

Note that we use the MATLAB function for LLE written by the author (http://www.cs.toronto.edu/~roweis/nldr.html).

References

Donaho, D. L. and C. Grimes (2003) 'Hessian eigenmaps: Locally linear embedding techniques for high-dimensional data,' *Proceedings of the National Academy of Science*, 100(10): 5591 - 5596.

Jackson, J. Edward (1991) *A User's Guide to Principal Components*, Wiley.

Jolliffe, I. T. (2002) *Principal Component Analysis*, 2nd Edition, Springer

Martinez, W. and A. Martinez (2004) *Exploratory Data Analysis with MATLAB*, CRC Press

Tenenbaum, J. B., V. de Silva, and J. C. Langford, (2000) 'A glogal geometric framework for nonlinear dimensionality reduction,' Science, 290: 2319 - 2323

K-Means Clustering GUI

To access from the command line use: **`kmeansgui`**

Unlike the other GUIs, this one provides the capability for one method: *k*-means clustering. This allows the user to find groups in the data and then explore the results visually using the **`Graphical EDA GUI`** and other tools.

In *k*-means clustering, one first has to specify the number of clusters (*k*) as well as an initial location for the *k* cluster centers. These can be found in several ways, e.g., use *k* observations randomly selected from the sample. After we have the initial cluster centers, then we group each observation with the closest cluster. The centers are then calculated. This process is repeated until no changes are made.

We use the *k*-means clustering functionality provided by The MathWorks, Inc. in the Statistics Toolbox. Note that the actual MATLAB **`kmeans`** function provides a lot more capbility and options than what this GUI accesses. We encourage the user to look at the MATLAB documentation for the Statistics Toolbox to see what else is available.

A screenshot of the GUI is shown below.

Note that we provide the usual buttons (**`Load Data`**, **`Transform Data`**, etc.), as well as a button that brings up the **`Dimensionality Reduction GUI`**.

Section 1: Create the Clusters

This section creates the clusters, and it involves several steps that are outlined below.

- Step 1: Select the data set you want to cluster using the pull-down menu.
- Step 2: Choose the type of initialization. The *k*-means method of clustering starts with an initial configuration of cluster centers. These can be obtained in several ways. One can take a random sample of *k* observations in the data set or find *k* points randomly selected (uniformly) from the range of the data.
- Step 3: The user can select a distance that will be used to group observations with the closest cluster center. This GUI provides access to two of the options provided by the MATLAB **`kmeans`** function.
- Step 4: Enter the number of clusters (*k*) in the edit box.
- Step 5: The algorithm for doing *k*-means clustering is a greedy algorithm, and it relies on an initial starting configuration of cluster centers. Thus, it is best to run the method several times. MATLAB provides this capability. The grouping that provides the best results (smallest within-cluster sums of point-to-cluster-centroid distances) is returned. Enter the number of times you want to run *k*-means in the edit box.
- Step 6: Push the button to run *k*-means. You can see the progress and other results in the MATLAB command window.

Sesction 2: Visualize the Clusters

Clustering algorithms usually produce the desired number of clusters. However, the resulting clusters might not make sense (i.e., similar observations are not grouped together or the groupings are spurious ones.). So, one should always look at the clusters and evaluate the usefulness of the results. For several reasons, this is often very difficult to do. In keeping with the EDA philosophy, we provide some ways to graphically assess the clusters.

In this part of the GUI, one can select from three visualization capabilities: ReClus, silhouette plots, and the **`Graphical EDA GUI`**.

- *ReClus* is a method that lays out the observations according to their groups. The form of ReClus that is accessed by this GUI has minimal capability and is really useful in conjunction with the **`Brushing and Linking GUI`** (accessed from the **`Graphical EDA GUI`**). The function for ReClus provided with the EDA Toolbox (written by these authors) has more capabilities.
- A *silhouette plot* displays the silhouette value for each point, by cluster. The silhouette value is a measure of how similar a point is to points in its own cluster, as compared to points in other clusters. Silhouette values range from -1 to +1, where large values are better.
- The **`Graphical EDA GUI`** option will bring up the **`gedagui`**. One can then explore the results using brushing and linking, coloring by cluster IDs, and other methods.

Section 3: Output the Cluster IDs

This section provides only one option: to output the cluster IDs to the workspace. Push this button, and a dialog box will come up.

Enter the desired name in the box, and an *n*-dimensional vector of IDs will be saved to the workspace. Each value in the array indicates the cluster number for the observation.

References

Everitt, B. S., S. Landau, L. Morven (2001) *Cluster Analysis*, Hodder Arnold Production.
Kaufman L., and P. J. Rousseeuw, (1990) *Finding Groups in Data: An Introduction to Cluster Analysis*, Wiley.
Martinez, W. and A. Martinez (2004) *Exploratory Data Analysis with MATLAB*, CRC Press.

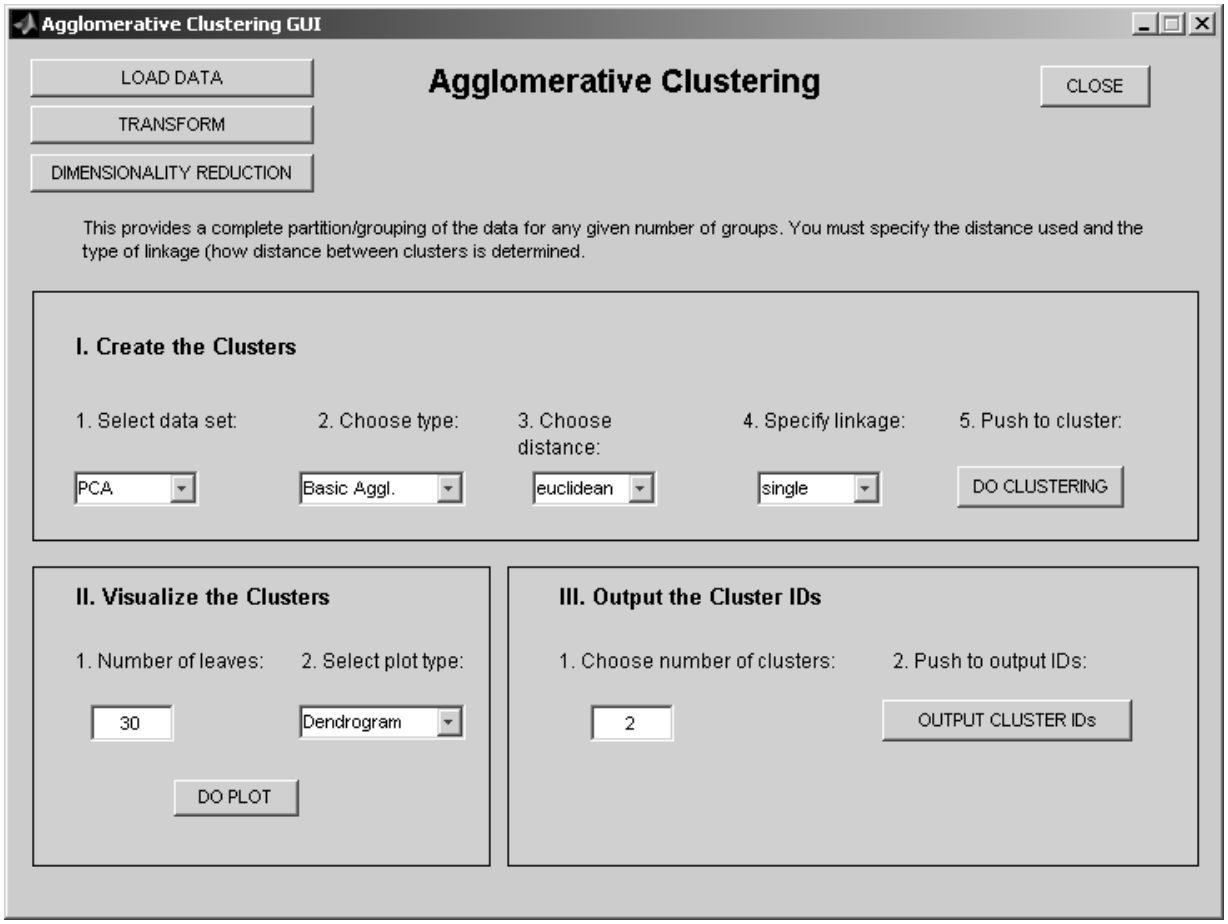
Agglomerative Clustering GUI

To access from the command line use: **`agcgui`**

Unlike most of the other GUIs, this one provides the capability for one method: *agglomerative clustering*. This allows the user to find groups in the data and then explore the results visually using the **`Graphical EDA GUI`** and other tools.

In agglomerative clustering, each observation starts out in its own single cluster, i.e., we have *n* clusters. During the clustering process, the two closest groups are merged, continuing until all observations are in one group. The user must specify a distance that will be used, as well as a way to measure the distance between clusters (called *linkage*). Clustering results are highly dependent on these two choices, especially on the type of linkage that is used. So, we recommend that users try different options for the distance and linkage to explore different groupings.

We show a screenshot of the GUI below.



We use the agglomerative clustering functionality in the Statistics Toolbox (The MathWorks, Inc.). Note that the actual MATLAB functionality provides a lot more capbility and options than what our GUI accesses. See the MATLAB **`Help`** file and other documentation for further information on these options.

Note the additional button we provide that allows one to access the **`Dimensionality Reduction GUI`** (Principal Component Analysis, ISOMAP, and LLE). The user can click on this, reduce the dimensionality, and return to the clustering GUI to cluster the data in fewer dimensions.

Section I: Create the Clusters

The first section accesses controls that provide information regarding the desired clustering options, as well as the means to create the clusters. We briefly discuss the various steps below.

- Step 1: Select the data set to be clustered from the menu. Reduced dimensionality data sets will appear, if available.
- Step 2: Choose the type of agglomerative clustering. There are two choices here: *Basic Agglomerative Clustering* and *Model-Based Agglomerative Clustering*. The basic version of agglomerative clustering is the one typically described in the books on clustering and pattern recognition. The model-based agglomerative clustering merges clusters at each stage that maximize a likelihood function, so information about the distance and linkage is not used.
- Step 3: Choose the distance. The user can choose from euclidean, standardized euclidean, cityblock, Mahalanobis, and Minkowski (exponent is 2).
- Step 4: Choose the type of linkage. Available choices are single, complete, average, and Ward's. See the MATLAB **`Help`** documentation or other references on clustering.
- Step 5: Do the clustering. Push the button to cluster the data. A dialog box will appear when the data have been clustered. Information on the process is not shown in the command window when the basic agglomerative clustering is done. However, if the user chooses model-based agglomerative clustering, then the status can be tracked in the command window.

Section II: Visualize the Clusters

This section provides 5 ways to plot the resulting clusters. These are: *dendrograms*, *treemaps*, *rectangle plots*, *ReClus*, and the **`Graphical EDA GUI`**. In some of the cases, the user must first specify the number of leaves (or clusters) to plot. To construct the desired plot, just push the **`Do Plot`** button. We now briefly describe these methods.

Dendrograms are a tree-like structure that consists of many box-like lines that show the hierarchical structure of the process. Each internal node of the dendrogram links two clusters. The dendrograms can be shown horizontally with the root node on the left or vertically with the root node at the top. The dendrogram function provided in the Statistics Toolbox allows the user to construct either type of dendrogram, along with other options. The GUI implements the vertical dendrogram only.

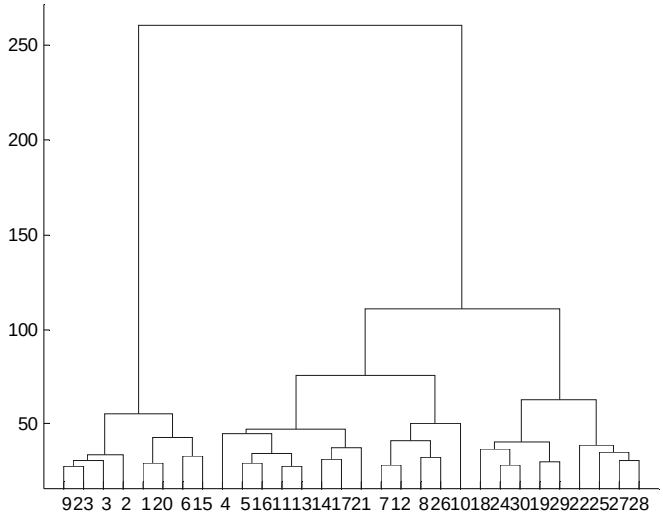
The leaves of the tree represent single data points. As one might imagine, if n is large, then there will be a lot of overplotting. The MATLAB dendrogram function displays a maximum of 30 leaf nodes as a default. Thus, a leaf node might contain more than one observation, and the leaf labels do not correspond to actual data points. The user can specify the number of leaf nodes to display in the GUI text box.

The vertical axis (or horizontal axis for horizontal dendrograms) represents the distance at which clusters are merged. In the case of model-based agglomerative clustering, the vertical axis represents the objective function (which is an indication of the likelihood).

One other important consideration regarding the dendrogram is its utility in estimating the number of groups in the data set. To do this, one looks for large vertical (or horizontal in the case of horizontal dendrograms) joins. These are an indication that two disparate groups were merged.

We show an example of a dendrogram below. Note that we see evidence for two clusters; we can obtain these clusters by cutting the dendrogram at a distance of say 150 or 200. If we cut the dendrogram at 75, then we will get 3 clusters.

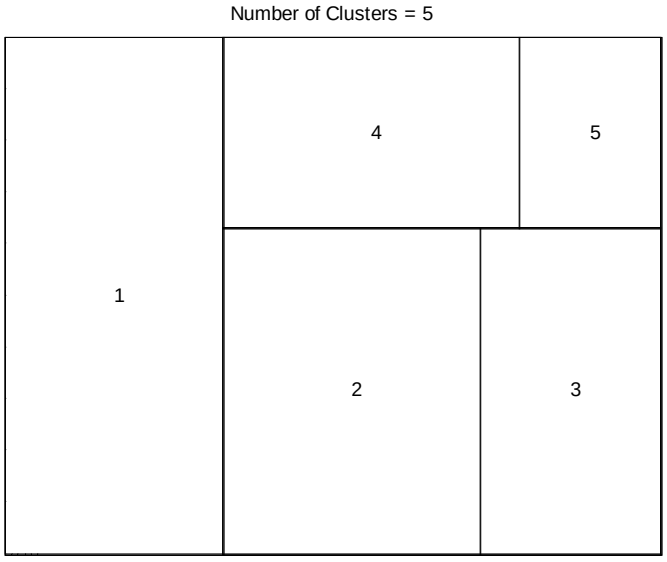
Note that hierarchical clustering provides a complete partitioning of the data set for any given number of groups (up to n).



The *treemap* plot was developed as a way of constructing a space-filling dendrogram. We can see in the above dendrogram example that most of the plotting information is given in a small area of the plot. There is a lot of wasted whitespace that could be better utilized to convey information.

The main rectangle in a treemap plot represents the entire data set; one can think of it as the root. The area of the root rectangle is subdivided into rectangles, each one representing a cluster or group. The area of each cluster rectangle is an indication of the size of the cluster, something that is not apparent in dendrograms.

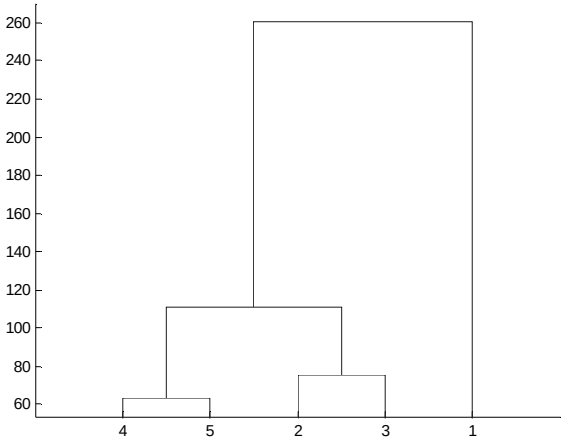
One has to specify the number of clusters to display, using the edit box. Push the **`Do Plot`** button to construct the plot. An example of a treemap plot for 5 groups is shown here.



Note that the hierarchical nature can be seen in this plot when we do not have a lot of clusters to display. For example, groups 4 and 5 have the same parent, as do groups 2 and 3. However, sometimes it is hard to see what is connected, and we lose the information regarding the distance (or other value) at which groups are merged.

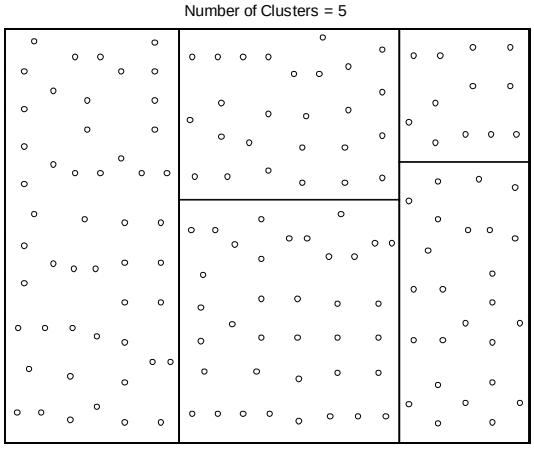
So, it is a good idea to construct a corresponding dendrogram to get more information. Here we have the associated dendrogram with 5 groups. The labels match in the two plots; i,e, group 5 references the same group in the dendrogram and the treemap.

The *rectangle* plot is an adaptation of the treemap idea. We still have the main rectangle representing the root node. The parent is further subdivided into the number of groups specified in the edit box. Individual data points are also shown on the plot. The placement of the points are done in such a way that each one would be in the center of a rectangle if one chooses *n* groups to plot.



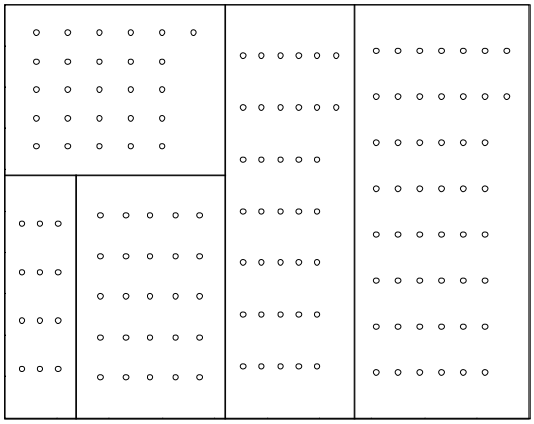
The rectangle plot is useful in brushing and linking. This can be done via the **`Graphics EDA GUI`** and the **`Brushing and Linking`** button.

A rectangle plot for the same 5 groups is shown here.



ReClus is an adaptation of the treemap and rectangle plot that is suitable for non-hierarchical clustering methods, such as *k*-means clustering and model-based clustering. Because of this, ReClus does not convey any hierarchical information.

One still has to specify the number of groups to display using the edit box. The parent rectangle indicates the entire data set, and sub-rectangles indicate clusters. The individual data points are shown as symbols and are evenly laid out through the rectangle. The ReClus plot is given below.



The ReClus functionality accessed via the GUI does not take advantage of the full power of the ReClus idea. Please see the function included in the EDA toolbox and Martinez and Martinez (2004) for more information on what is available.

Like the rectangle plot, the ReClus plot is helpful in exploring the clusters in conjunction with other plots via the **`Brushing and Linking GUI`**.

Section III: Output the Cluster IDs

This section provides only one option: to output the cluster IDs to the workspace. However, one first has to specify the number of clusters. Enter the value in the text box, push the button, and a dialog box will come up.

Enter the desired name in the box, and an *n*-dimensional vector of IDs will be saved to the workspace. Each value in the array indicates the cluster number for the observation.

References

Everitt, B. S., S. Landau, L. Morven (2001) *Cluster Analysis*, Hodder Arnold Production.

Fraley, C. and A. E. Raftery (1998) 'How many clusters? Which clustering method? Answers via model-based cluster analysis,' *The Computer Journal*, 41: 578 - 588.

Martinez, A. R. (2002) *A Statistical Framework for the Representation of Semantics*, Ph.D. Dissertation, George Mason University.

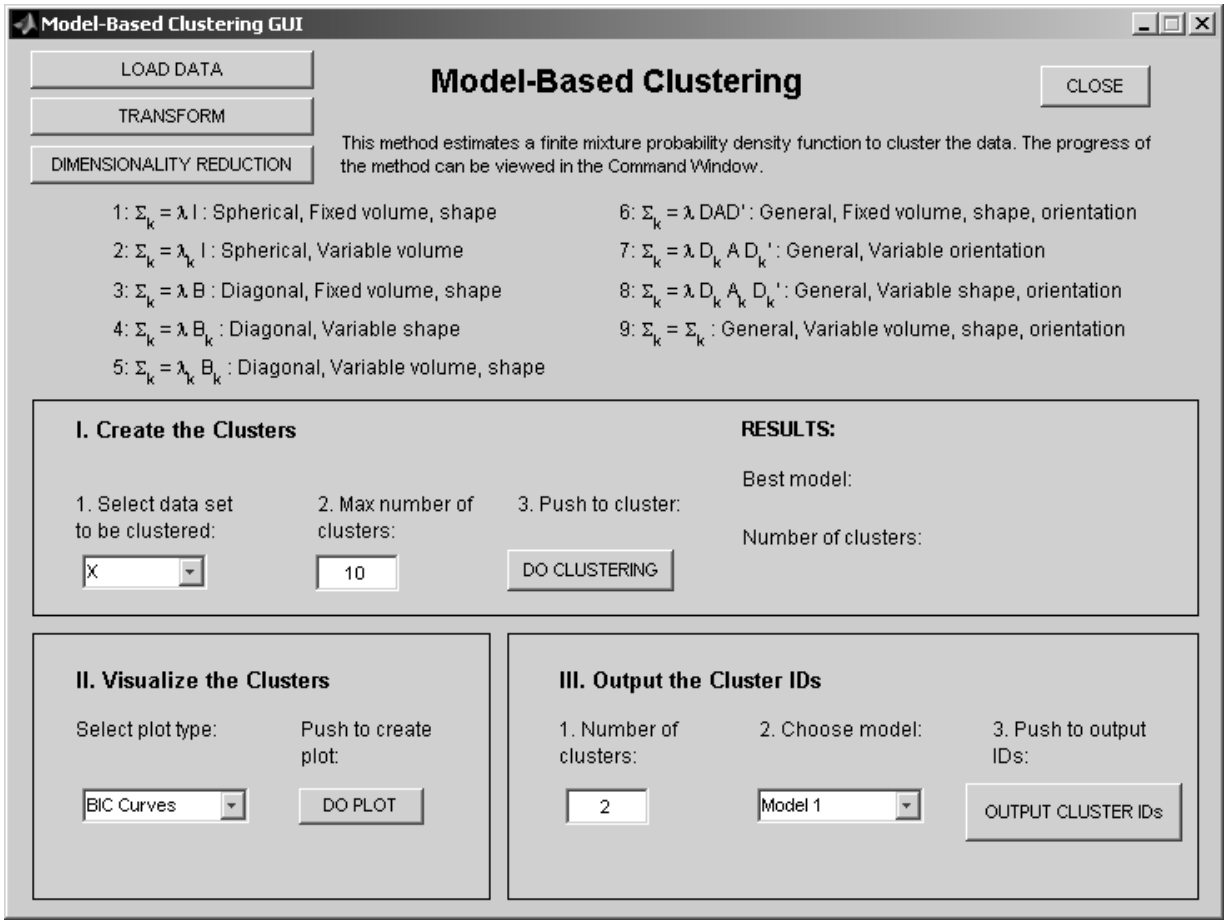
Martinez, A. R. and E. Wegman (2002) 'Encoding of text to preserve meaning,' *Proceedings Army Conference on Applied Statistics*

Schneiderman, Ben (1992) 'Tree visualization with tree-maps: 2-D space-filling approach,' *ACM Trans. on Graphics*, 11:92 - 99 (http://www.cs.umd.edu/hcil/treemap-history/).

Model-Based Clustering (MBC) GUI

To access from the command line use: **`mbcgui`**

Unlike most of the other GUIs, this one provides the capability for one method: *model-based clustering*. This allows the user to find groups in the data and then explore the results visually using the **`Graphical EDA GUI`** and other graphical tools. We present a screenshot of the GUI below



Model-based clustering is a methodology that is based on a probability density estimation methodology called finite mixtures. It employs three basic steps, and it also provides a mechanism for estimating the number of groups represented by the data.

The three main parts of model-based clustering are

- A model-based agglomerative clustering to get an initial partition of the data
- The Expectation-Maximization (EM) algorithm for estimating the finite mixture probability density function
- The Bayesian Information Criterion (BIC) for choosing the best model

More detail on these individual components of model-based clustering is beyond the scope of this user's guide. For more information, we refer the user to the reference given below and the MBC website: http:// www.stat.washington.edu/mclust/.

However, the user should note that the model-based agglomerative part of MBC is available as a stand-alone clustering tool via the Agglomerative Clustering GUI. The user cannot get to the agglomerative clustering information via the **`MBC GUI`**.

As with the other clustering GUIs, we provide a button to first reduce the dimensionality. If the user does this (either via this GUI or previously through another GUI), then the reduced dimensionality data can be accessed via the MBC GUI.

This GUI has sections that are similar to the other clustering GUIs: creating the clusters, visualizing them, and sending cluster IDs to the workspace. We describe each of these here.

Section I: Create the Clusters

The first section contains the necessary inputs. Note that the user does not have to specify the number of clusters, as in *k*-means.

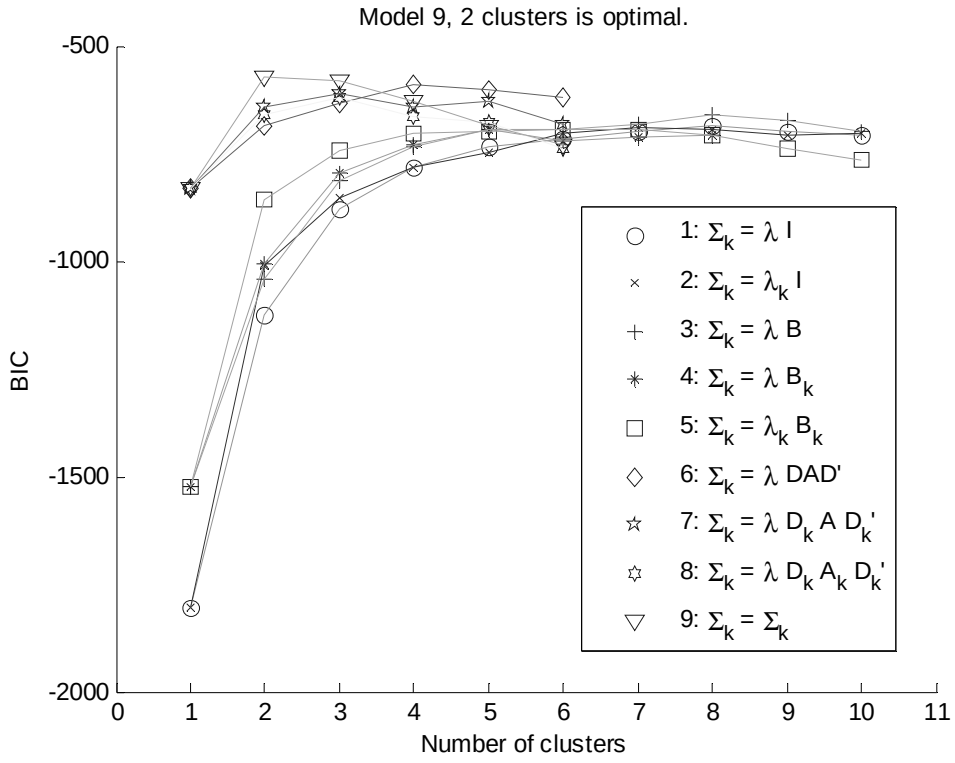
- Step 1: Select the data set to be clustered. This menu will have the original data set (possibly transformed) as well as data that have been reduced in dimensionality.
- Step 2: Specify the maximum number of clusters to estimate. In some sense, model-based clustering is an exhaustive search over 9 different models (specified at the top of the GUI) and the number of clusters one is looking for (1 through the maximum number of clusters). If the maximum number is large, then the time required to perform model-based clustering can be significant.
- Step 3: Push the button to perform the clustering. One can see the progress being made by looking in the command window.

Once the algorithm has finished, the best model and the estimated number of clusters is shown in the GUI window. An information box will pop up letting the user know that the clustering has been done.

Section II: Visualize the Clusters

The next section accesses various visualization tools that allow the user to visualize the data. We provide the following displays: the *BIC curves*, *ReClus*, and the Graphical EDA GUI.

The *BIC curves* plot shows the values of the BIC for the 9 different models, one curve per model. The horizontal axis represents the number of clusters, and the vertical axis shows the value of the BIC. We are looking for maximum values of the BIC to give us an estimate of the best model (the curve with the maximum) and the number of clusters. In reality, we are looking for an elbow in the curve rather than the absolute maximum. The plot of the BIC curves for these data are shown here.

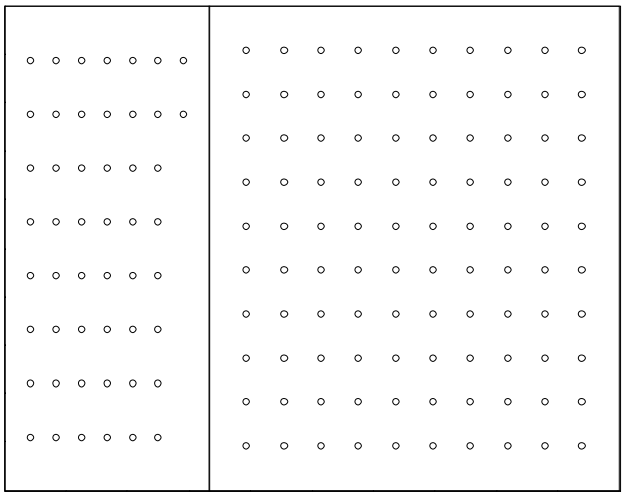


Note the elbow in the curve for model 9 and 2 clusters. This seems to be a stronger indication of 2 groups. However, the gradual increase in BIC values for some of the other curves (at 7 through 10 clusters) is an indication of overfitting.

It should be noted that the BIC gives us a way to choose a model and number of groups, but one of the other groupings might contain useful information. As with all cluster results, one needs to go back and look at other cluster configurations to look for interesting structure and patterns. See
Section III below for more information on how one can access all the groupings obtained through MBC.

ReClus is an adaptation of the treemap and rectangle plot (see the description of the **`Agglomerative Clustering GUI`** for more information) that is suitable for non-hierarchical clustering methods, such as *k*-means clustering and model-based clustering.

The ReClus plot in the MBC GUI displays the results of the best clustering (model and number of groups). The parent rectangle indicates the entire data set, and sub-rectangles indicate clusters. The individual data points are shown as symbols and are evenly laid out through the rectangle. The ReClus plot for two clusters is given below.



The ReClus functionality accessed via the GUI does not take advantage of the full power of the ReClus idea. Please see the function included in the EDA toolbox and Martinez and Martinez (2004) for more information on what is available.

The version of the ReClus plot accessed through the EDA GUI Toolbox is helpful in exploring the clusters in conjunction with other plots via the **`Brushing and Linking GUI`**.

The final choice for plotting is to bring up the **`Graphical EDA GUI`**. This allows one to look at various plots where the clusters are displayed with different colors, in addition to accessing the **`Brushing and Linking GUI`**. Note that the model-based cluster IDs used by the **`Graphical EDA GUI`** will be the ones for the best model and number of clusters.

Section III: Output the Cluster IDs

We use the MBC function written for the book *Exploratory Data Analysis with MATLAB*. See that text for more information on the functions related to model-based clustering that can be accessed via the command window.

There are more options included with MBC function (invoked via the command window), most notably, the ability to obtain all cluster configurations. We provide some of that capability in the final section of this GUI. One can output cluster IDs for any of the cluster configurations by specifying the model and the number of clusters. We recommend doing this for all groupings of interest before closing the GUI, since all of that information (except for the best model and number of clusters) will be lost once the GUI is closed.

More Information

We wrote a previous version of MBC for MATLAB, and it is available at StatLib as the Model-Based Clustering Toolbox. Please note, though, that as of October 2006, it does not have all 9 models as we have here with the EDA Toolbox and GUIs.

References

Everitt, B. S., S. Landau, L. Morven (2001) *Cluster Analysis*, Hodder Arnold Production.

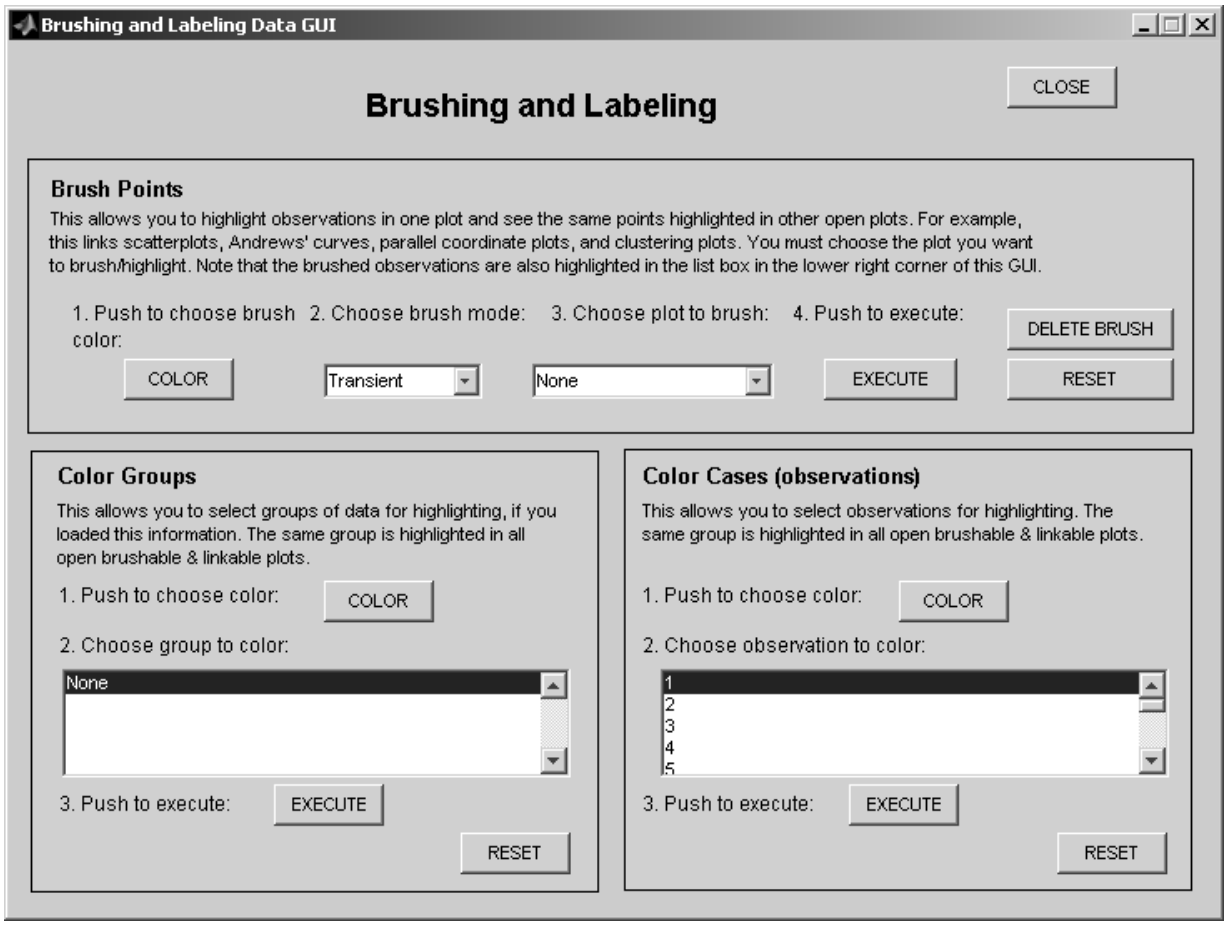
Fraley, C. and A. E. Raftery (1998) 'How many clusters? Which clustering method? Answers via model-based cluster analysis,' *The Computer Journal*, 41: 578 - 588.

Brushing and Linking GUI

This GUI cannot be accessed from the command line. One has to open this GUI via the **`Brush Data`** button in the ***upper left corner*** of the **`Graphical EDA GUI`**.

The **`Brushing and Linking GUI`** allows the user to highlight points in one plot and see the same points highlighted in linkeable plots. This is similar to the scatterplot brushing option that is accessed in the main **`Graphical EDA GUI`**. In that plot, points can be highlighted in one subplot with corresponding points being highlighted in other subplots in the same window. In contrast, the brushing and linking in this GUI affects all appropriate open plots.

It has several other capabilities. We outline these shortly, but first we show a screenshot of the GUI.



The GUI has three main sections: *Brush Points*, *Color Groups*, and *Color Cases*. We briefly describe each of these capabilities.

Brush Points

A useful interactive tool for exploratory data analysis is *brushing and linking*. 2-D scatterplots are very informative and easy to understand, but it is difficult to visualize the data in a similar manner when we have more than two

dimensions. The scatterplot matrix is one display that was developed to link 2-D scatterplots together by showing them in the same figure. The human needs to then mentally link points across plots and axes.

Cleveland (1993) discusses the idea of brushing or painting points in one plot, which are then highlighted in other linked plots. This allows one to discover patterns and anomalies in the data.

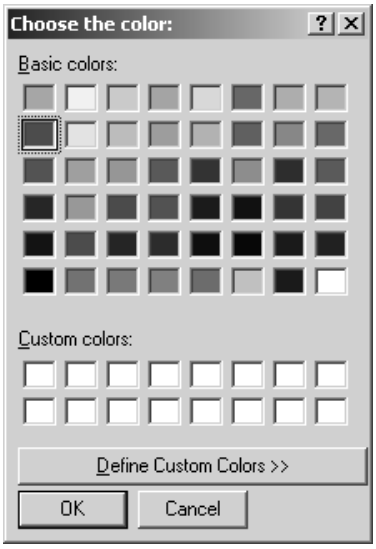
Brushable plots in the EDA Toolbox are:

- 2-D scatterplots
- ReClus plots
- Rectangle plots

Linkable plots in the EDA Toolbox are:

- 2-D scatterplots
- ReClus plots
- Rectangle plots
- Andrews' curve plots
- Parallel coordinate plots

The steps for brushing and linking are outlined here.



- Step 1: Choose brush color. A dialog box (shown above) will come up when this buton is pushed. It allows one to pick the color that will be used to paint the observations that are highlighted with the brush. This dialog box is pictured above. Just select the color and click OK. The default color is red.
- Step 2: Choose the mode. There are three modes for brushing: transient, lasting, and erasing. With *transient* brushing, points are highlighted only while inside the brush. The color is removed once the points go outside of the brush. When the brush is in *lasting* mode, points remain painted, even when they are outside of the brush. The user can select the *Erase Highlight* mode to remove the highlighting for points inside the brush.
- Step 3: Choose plot to brush. We provide a menu containing all brushable plots. Select the plot where you want to create the brush and paint the points.
- Step 4: Start brushing. Push the **`Execute`** button the **first** time you start brushing. You also push this if you have deleted the brush or reset the figures. You will get an error if you push this button unnecessarily.

ERROR!!! FIX!!! Not sure why this doesn't work here, but the scatterplot matrix function doesn't work right. The brush goes off the screen like it did before. In any event, the scatterplot brushing function needs to be fixed to be in line with the correct version accessed via B&L GUI.

All figures are created in the same place on the computer screen, so they appear on top of each other. Move the figures so they are not covered. This might not be possible and depends on the number of figures and the size of the computer screen. You can also resize the figure windows. The point is that one needs to be able to see the plot that is being brushed and the corresponding points being highlighted in the linked plots.

Once the user hits the **`Execute`** button, the brushable figure window will pop up, and the cursor will be inside that window. The user draws a brush (a rectangle object) by placing the cursor inside the axes of the plot, holding down the left mouse button and dragging the cursor. One cannot re-size the brush once it is created.

To grab the brush, place the cursor on the brush outline and push the left mouse button. The cursor will show up as a cross. Now one can move the brush by moving the cursor, holding down the mouse button at the same time.

The highlighting behavior is different, depending on the mode that is selected. We repeat a description of the modes here:

- *Transient* is the default mode. In this case, the observations inside the brush are highlighted. The highlightling is removed once the brush passes over them.
- In *lasting* mode, observations inside the brush are highlighted, and the highlighting remains.
- *Erase* mode allows the user to undo the highlighting. Observations inside the brush are returned to the original state.

IMPORTANT: Note that the **`Execute`** button is pushed only once to create a brush and to designate the brushable plot. The user can go back and change the color or the brushing mode and continue brushing without hitting the **`Execute`** button. If you hit the **`Execute`** button, then the plots are reset.

The **`Delete Brush`** button is used when one wants to copy or output the brushed figure for documentation. The button will delete the brush, but will not remove the coloring.

The **`Reset`** button will delete the brush as well as the higlighting. All brushable and linkable plots are reset to their original state.

Color Groups

If the one has group labels for the observations, then this can be used to color the observations in all open brushable and linnkable plots. There are three steps in this section.

- Step 1: Select a color from the dialog box.
- Step 2: Choose a group to color from the list.
- Step 3: Push the button to color the observations

Color Cases

One can use this last section to color specific observations. The selected observations will be highlighted in all open linkable and brushable plots. As with the previous section, there are three steps for coloring individual observations.

- Step 1: Select a color from the dialog box.
- Step 2: Choose an observation to color from the list. One can use the Shift key to highlight a range of observations or the Control key to highlight several observations individually. This is the same capability provided in Windows applications.
- Step 3: Push the button to color the observations

As stated previously, one does not have to load up labels for the observations. The default for labels is simply the observation number.

Note that individual cases are highlighted in this section when one brushes observations using the brushing and linking tool on this GUI. This can be useful in indentifying and removing outliers.

References

Cleveland, W. S. (1993) *Visualizing Data*, Hobart Press.