Title

stata.com

mdsmat — Multidimensional scaling of proximity data in a matrix

Syntax

`mdsmat` *matname* [, *options*]

options	Description
Model	
`method(`*method*`)`	method for performing MDS
`loss(`*loss*`)`	loss function
`transform(`*tfunction*`)`	permitted transformations of dissimilarities
`normalize(`*norm*`)`	normalization method; default is `normalize(principal)`
`names(`*namelist*`)`	variable names corresponding to row and column names of the matrix; required with all but `shape(full)`
`shape(full)`	*matname* is a square symmetric matrix; the default
`shape(lower)`	*matname* is a vector with the rowwise lower triangle (with diagonal)
`shape(llower)`	*matname* is a vector with the rowwise strictly lower triangle (no diagonal)
`shape(upper)`	*matname* is a vector with the rowwise upper triangle (with diagonal)
`shape(uupper)`	*matname* is a vector with the rowwise strictly upper triangle (no diagonal)
`s2d(standard)`	convert similarity to dissimilarity: $d_{ij} = \sqrt{s_{ii} + s_{jj} - 2s_{ij}}$
`s2d(oneminus)`	convert similarity to dissimilarity: $d_{ij} = 1 - s_{ij}$
Model 2	
`dimension(`*#*`)`	configuration dimensions; default is `dimension(2)`
`force`	fix problems in proximity information
`addconstant`	make distance matrix positive semidefinite (classical MDS only)
`weight(`*matname*`)`	specifies a weight matrix with the same shape as the proximity matrix
Reporting	
`neigen(`*#*`)`	maximum number of eigenvalues to display; default is `neigen(10)` (classical MDS only)
`config`	display table with configuration coordinates
`noplot`	suppress configuration plot

Minimization

initialize(*initopt*)	start with configuration given in *initopt*
tolerance(#)	tolerance for configuration matrix; default is tolerance(1e-4)
ltolerance(#)	tolerance for loss criterion; default is ltolerance(1e-8)
iterate(#)	perform maximum # of iterations; default is iterate(1000)
protect(#)	perform # optimizations and report best solution; default is protect(1)
nolog	suppress the iteration log
trace	display current configuration in iteration log
gradient	display current gradient matrix in iteration log
sdprotect(#)	advanced; see *Options* below

sdprotect(#) does not appear in the dialog box.

See **[U] 20 Estimation and postestimation commands** for more capabilities of estimation commands.

method	Description
classical	classical MDS; default if neither loss() nor transform() is specified
modern	modern MDS; default if loss() or transform() is specified; except when loss(stress) and transform(monotonic) are specified
nonmetric	nonmetric (modern) MDS; default when loss(stress) and transform(monotonic) are specified

loss	Description
stress	stress criterion, normalized by distances; the default
nstress	stress criterion, normalized by disparities
sstress	squared stress criterion, normalized by distances
nsstress	squared stress criterion, normalized by disparities
strain	strain criterion (with transform(identity) is equivalent to classical MDS)
sammon	Sammon mapping

tfunction	Description
identity	no transformation; disparity = dissimilarity; the default
power	power α: disparity = dissimilarity$^{\alpha}$
monotonic	weakly monotonic increasing functions (nonmetric scaling); only with loss(stress)

norm	Description
principal	principal orientation; location = 0; the default
classical	Procrustes rotation toward classical solution
target(*matname*) [, copy]	Procrustes rotation toward *matname*; ignore naming conflicts if copy is specified

initopt	Description
classical	start with classical solution; the default
random[(#)]	start at random configuration, setting seed to #
from(*matname*)[, copy]	start from *matname*; ignore naming conflicts if copy is specified

Menu

Statistics > Multivariate analysis > Multidimensional scaling (MDS) > MDS of proximity matrix

Description

mdsmat performs multidimensional scaling (MDS) for two-way proximity data with an explicit measure of similarity or dissimilarity between objects, where the proximities are found in matrix *matname*. mdsmat performs classical metric MDS (Torgerson 1952) as well as modern metric and nonmetric MDS; see the loss() and transform() options.

If your proximities are stored as variables in long format, see [MV] **mdslong**. If you are looking for MDS on a dataset on the basis of dissimilarities between observations over variables, see [MV] **mds**.

Computing the classical solution is straightforward, but with modern MDS the minimization of the loss criteria over configurations is a high-dimensional problem that is easily beset by convergence to local minimums. mds, mdsmat, and mdslong provide options to control the minimization process 1) by allowing the user to select the starting configuration and 2) by selecting the best solution among multiple minimization runs from random starting configurations.

Options

Model

method(*method*) specifies the method for MDS.

method(classical) specifies classical metric scaling, also known as "principal coordinates analysis" when used with Euclidean proximities. Classical MDS obtains equivalent results to modern MDS with loss(strain) and transform(identity) without weights. The calculations for classical MDS are fast; consequently, classical MDS is generally used to obtain starting values for modern MDS. If the options loss() and transform() are not specified, mds computes the classical solution, likewise if method(classical) is specified loss() and transform() are not allowed.

method(modern) specifies modern scaling. If method(modern) is specified but not loss() or transform(), then loss(stress) and transform(identity) are assumed. All values of loss() and transform() are valid with method(modern).

method(nonmetric) specifies nonmetric scaling, which is a type of modern scaling. If method(nonmetric) is specified, loss(stress) and transform(monotonic) are assumed. Other values of loss() and transform() are not allowed.

loss(*loss*) specifies the loss criterion.

loss(stress) specifies that the stress loss function be used, normalized by the squared Euclidean distances. This criterion is often called Kruskal's stress-1. Optimal configurations for loss(stress) and for loss(nstress) are equivalent up to a scale factor, but the iteration paths may differ. loss(stress) is the default.

`loss(nstress)` specifies that the stress loss function be used, normalized by the squared disparities, that is, transformed dissimilarities. Optimal configurations for `loss(stress)` and for `loss(nstress)` are equivalent up to a scale factor, but the iteration paths may differ.

`loss(sstress)` specifies that the squared stress loss function be used, normalized by the fourth power of the Euclidean distances.

`loss(nsstress)` specifies that the squared stress criterion, normalized by the fourth power of the disparities (transformed dissimilarities) be used.

`loss(strain)` specifies the strain loss criterion. Classical scaling is equivalent to `loss(strain)` and `transform(identity)` but is computed by a faster noniterative algorithm. Specifying `loss(strain)` still allows transformations.

`loss(sammon)` specifies the Sammon (1969) loss criterion.

`transform(`*tfunction*`)` specifies the class of allowed transformations of the dissimilarities; transformed dissimilarities are called disparities.

`transform(identity)` specifies that the only allowed transformation is the identity; that is, disparities are equal to dissimilarities. `transform(identity)` is the default.

`transform(power)` specifies that disparities are related to the dissimilarities by a power function,

$$\text{disparity} = \text{dissimilarity}^{\alpha}, \qquad \alpha > 0$$

`transform(monotonic)` specifies that the disparities are a weakly monotonic function of the dissimilarities. This is also known as nonmetric MDS. Tied dissimilarities are handled by the primary method; that is, ties may be broken but are not necessarily broken. `transform(monotonic)` is valid only with `loss(stress)`.

`normalize(`*norm*`)` specifies a normalization method for the configuration. Recall that the location and orientation of an MDS configuration is not defined (“identified”); an isometric transformation (that is, translation, reflection, or orthonormal rotation) of a configuration preserves interpoint Euclidean distances.

`normalize(principal)` performs a principal normalization, in which the configuration columns have zero mean and correspond to the principal components, with positive coefficient for the observation with lowest value of `id()`. `normalize(principal)` is the default.

`normalize(classical)` normalizes by a distance-preserving Procrustean transformation of the configuration toward the classical configuration in principal normalization; see [MV] **procrustes**. `normalize(classical)` is not valid if `method(classical)` is specified.

`normalize(target(`*matname*`)` [`, copy`]`)` normalizes by a distance-preserving Procrustean transformation toward *matname*; see [MV] **procrustes**. *matname* should be an $n \times p$ matrix, where n is the number of observations and p is the number of dimensions, and the rows of *matname* should be ordered with respect to `id()`. The rownames of *matname* should be set correctly but will be ignored if `copy` is also specified.

Note on `normalize(classical)` and `normalize(target())`: the Procrustes transformation comprises any combination of translation, reflection, and orthonormal rotation—these transformations preserve distance. Dilation (uniform scaling) would stretch distances and is not applied. However, the output reports the dilation factor, and the reported Procrustes statistic is for the dilated configuration.

`names(`*namelist*`)` is required with all but `shape(full)`. The number of names should equal the number of rows (and columns) of the full similarity or dissimilarity matrix and should not contain duplicates.

`shape(`*shape*`)` specifies the storage mode of the existing similarity or dissimilarity matrix *matname*. The following storage modes are allowed:

`full` specifies that *matname* is a symmetric $n \times n$ matrix.

`lower` specifies that *matname* is a row or column vector of length $n(n+1)/2$, with the rowwise lower triangle of the similarity or dissimilarity matrix including the diagonal.

$$\mathrm{D}_{11}\ \mathrm{D}_{21}\ \mathrm{D}_{22}\ \mathrm{D}_{31}\ \mathrm{D}_{32}\ \mathrm{D}_{33}\ \ldots\ \mathrm{D}_{n1}\ \mathrm{D}_{n2}\ \ldots\ \mathrm{D}_{nn}$$

`llower` specifies that *matname* is a row or column vector of length $n(n-1)/2$, with the rowwise lower triangle of the similarity or dissimilarity matrix excluding the diagonal.

$$\mathrm{D}_{21}\ \mathrm{D}_{31}\ \mathrm{D}_{32}\ \mathrm{D}_{41}\ \mathrm{D}_{42}\ \mathrm{D}_{43}\ \ldots\ \mathrm{D}_{n1}\ \mathrm{D}_{n2}\ \ldots\ \mathrm{D}_{n,n-1}$$

`upper` specifies that *matname* is a row or column vector of length $n(n+1)/2$, with the rowwise upper triangle of the similarity or dissimilarity matrix including the diagonal.

$$\mathrm{D}_{11}\ \mathrm{D}_{12}\ \ldots\ \mathrm{D}_{1n}\ \mathrm{D}_{22}\ \mathrm{D}_{23}\ \ldots\ \mathrm{D}_{2n}\ \mathrm{D}_{33}\ \mathrm{D}_{34}\ \ldots\ \mathrm{D}_{3n}\ \ldots\ \mathrm{D}_{nn}$$

`uupper` specifies that *matname* is a row or column vector of length $n(n-1)/2$, with the rowwise upper triangle of the similarity or dissimilarity matrix excluding the diagonal.

$$\mathrm{D}_{12}\ \mathrm{D}_{13}\ \ldots\ \mathrm{D}_{1n}\ \mathrm{D}_{23}\ \mathrm{D}_{24}\ \ldots\ \mathrm{D}_{2n}\ \mathrm{D}_{34}\ \mathrm{D}_{35}\ \ldots\ \mathrm{D}_{3n}\ \ldots\ \mathrm{D}_{n-1,n}$$

`s2d(standard | oneminus)` specifies how similarities are converted into dissimilarities. By default, the command dissimilarity data. Specifying `s2d()` indicates that your proximity data are similarities.

Dissimilarity data should have zeros on the diagonal (that is, an object is identical to itself) and nonnegative off-diagonal values. Dissimilarities need not satisfy the triangular inequality, $D(i,j)^2 \leq D(i,h)^2 + D(h,j)^2$. Similarity data should have ones on the diagonal (that is, an object is identical to itself) and have off-diagonal values between zero and one. In either case, proximities should be symmetric. See option `force` if your data violate these assumptions.

The available `s2d()` options, `standard` and `oneminus`, are defined as follows:

`standard`	$\mathrm{dissim}_{ij} = \sqrt{\mathrm{sim}_{ii} + \mathrm{sim}_{jj} - 2\mathrm{sim}_{ij}} = \sqrt{2(1 - \mathrm{sim}_{ij})}$
`oneminus`	$\mathrm{dissim}_{ij} = 1 - \mathrm{sim}_{ij}$

`s2d(standard)` is the default.

`s2d()` should be specified only with measures in similarity form.

Model 2

`dimension(#)` specifies the dimension of the approximating configuration. # defaults to 2 and should not exceed the number of positive eigenvalues of the centered distance matrix.

`force` corrects problems with the supplied proximity information. `force` specifies that the dissimilarity matrix be symmetrized; the mean of D_{ij} and D_{ji} is used. Also, problems on the diagonal (similarities: $D_{ii} \neq 1$; dissimilarities: $D_{ii} \neq 0$) are fixed. `force` does not fix missing values or out-of-range values (that is, $D_{ij} < 0$ or similarities with $D_{ij} > 1$). Analogously, `force` symmetrizes the weight matrix.

`addconstant` specifies that if the double-centered distance matrix is not positive semidefinite (psd), a constant should be added to the squared distances to make it psd and, hence, Euclidean.

`weight(`*matname*`)` specifies a symmetric weight matrix with the same shape as the proximity matrix; that is, if `shape(lower)` is specified, the weight matrix must have this shape. Weights should be nonnegative. Missing weights are assumed to be 0. Weights must also be irreducible; that is, it is not possible to split the objects into disjointed groups with all intergroup weights 0. `weight()` is not allowed with `method(classical)`, but see `loss(strain)`.

Reporting

`neigen(#)` specifies the number of eigenvalues to be included in the table. The default is `neigen(10)`. Specifying `neigen(0)` suppresses the table. This option is allowed with classical MDS only.

`config` displays the table with the coordinates of the approximating configuration. This table may also be displayed using the postestimation command `estat config`; see [MV] **mds postestimation**.

`noplot` suppresses the graph of the approximating configuration. The graph can still be produced later via `mdsconfig`, which also allows the standard graphics options for fine-tuning the plot; see [MV] **mds postestimation plots**.

Minimization

These options are available only with `method(modern)` or `method(nonmetric)`:

`initialize(`*initopt*`)` specifies the initial values of the criterion minimization process.

`initialize(classical)`, the default, uses the solution from classical metric scaling as initial values. With `protect()`, all but the first run start from random perturbations from the classical solution. These random perturbations are independent and normally distributed with standard error equal to the product of `sdprotect(#)` and the standard deviation of the dissimilarities. `initialize(classical)` is the default.

`initialize(random)` starts an optimization process from a random starting configuration. These random configurations are generated from independent normal distributions with standard error equal to the product of `sdprotect(#)` and the standard deviation of the dissimilarities. The means of the configuration are irrelevant in MDS.

`initialize(from(`*matname*`)` [`, copy`]`)` sets the initial value to *matname*. *matname* should be an $n \times p$ matrix, where n is the number of observations and p is the number of dimensions, and the rows of *matname* should be ordered with respect to `id()`. The rownames of *matname* should be set correctly but will be ignored if `copy` is specified. With `protect()`, the second-to-last runs start from random perturbations from *matname*. These random perturbations are independent normal distributed with standard error equal to the product of `sdprotect(#)` and the standard deviation of the dissimilarities.

`tolerance(#)` specifies the tolerance for the configuration matrix. When the relative change in the configuration from one iteration to the next is less than or equal to `tolerance()`, the `tolerance()` convergence criterion is satisfied. The default is `tolerance(1e-4)`.

`ltolerance(#)` specifies the tolerance for the fit criterion. When the relative change in the fit criterion from one iteration to the next is less than or equal to `ltolerance()`, the `ltolerance()` convergence is satisfied. The default is `ltolerance(1e-8)`.

Both the `tolerance()` and `ltolerance()` criteria must be satisfied for convergence.

`iterate(#)` specifies the maximum number of iterations. The default is `iterate(1000)`.

protect(#) requests that # optimizations be performed and that the best of the solutions be reported. The default is protect(1). See option initialize() on starting values of the runs. The output contains a table of the return code, the criterion value reached, and the seed of the random number used to generate the starting value. Specifying a large number, such as protect(50), provides reasonable insight whether the solution found is a global minimum and not just a local minimum.

If any of the options log, trace, or gradient is also specified, iteration reports will be printed for each optimization run. Beware: this option will produce a lot of output.

nolog suppresses the iteration log, showing the progress of the minimization process.

trace displays the configuration matrices in the iteration report. Beware: this option may produce a lot of output.

gradient displays the gradient matrices of the fit criterion in the iteration report. Beware: this option may produce a lot of output.

The following option is available with mdsmat but is not shown in the dialog box:

sdprotect(#) sets a proportionality constant for the standard deviations of random configurations (init(random)) or random perturbations of given starting configurations (init(classical) or init(from())). The default is sdprotect(1).

Remarks and examples stata.com

Remarks are presented under the following headings:

Introduction

Multidimensional scaling (MDS) is a dimension-reduction and visualization technique. Dissimilarities (for instance, Euclidean distances) between observations in a high-dimensional space are represented in a lower-dimensional space (typically two dimensions) so that the Euclidean distance in the lower-dimensional space approximates the dissimilarities in the higher-dimensional space. See Kruskal and Wish (1978) for a brief nontechnical introduction to MDS. Young and Hamer (1987) and Borg and Groenen (2005) are more advanced textbook-sized treatments.

mdsmat performs MDS on a similarity or dissimilarity matrix *matname*. You may enter the matrix as a symmetric square matrix or as a vector (matrix with one row or column) with only the upper or lower triangle; see option shape() for details. *matname* should not contain missing values. The diagonal elements should be 0 (dissimilarities) or 1 (similarities). If you provide a square matrix (that is, shape(full)), names of the objects are obtained from the matrix row and column names. The row names should all be distinct, and the column names should equal the row names. Equation names, if any, are ignored. In any of the vectorized shapes, names are specified with option names(), and the matrix row and column names are ignored.

See option force if your matrix violates these assumptions.

In some applications, the similarity or dissimilarity of objects is defined by the researcher in terms of variables (attributes) measured on the objects. If you need to do MDS of this form, you should continue by reading [MV] **mds**.

Often, however, proximities—that is, similarities or dissimilarities—are measured directly. For instance, psychologists studying the similarities or dissimilarities in a set of stimuli—smells, sounds, faces, concepts, etc.—may have subjects rate the dissimilarity of pairs of stimuli. Linguists have subjects rate the similarity or dissimilarity of pairs of dialects. Political scientists have subjects rate the similarity or dissimilarity of political parties or candidates for political office. In other fields, relational data are studied that may be interpreted as proximities in a more abstract sense. For instance, sociologists study interpersonal contact frequencies in groups ("social networks"); these measures are sometimes interpreted in terms of similarities.

A wide variety of MDS methods have been proposed. `mdsmat` performs classical and modern scaling. Classical scaling has its roots in Young and Householder (1938) and Torgerson (1952). MDS requires complete and symmetric dissimilarity interval-level data. To explore modern scaling, see Borg and Groenen (2005). Classical scaling results in an eigen decomposition, whereas modern scaling is accomplished by the minimization of a loss function. Consequently, eigenvalues are not available after modern MDS.

Proximity data in a Stata matrix

To perform MDS of relational data, you must enter the data in a suitable format. One convenient format is a Stata matrix. You may want to use this format for analyzing data that you obtain from a printed source.

▷ Example 1

Many texts on multidimensional scaling illustrate how locations can be inferred from a table of geographic distances. We will do this too, using an example of distances in miles between 14 locations in Texas, representing both manufactured and natural treasures:

Big Bend	0	523	551	243	322	412	263	596	181	313	553	434	397	426
Corpus Christi	523	0	396	280	705	232	619	226	342	234	30	216	141	205
Dallas	551	396	0	432	643	230	532	243	494	317	426	235	274	151
Del Rio	243	280	432	0	427	209	339	353	62	70	310	231	154	287
El Paso	322	705	643	427	0	528	110	763	365	525	735	550	564	606
Enchanted Rock	412	232	230	209	528	0	398	260	271	69	262	40	91	148
Guadalupe Mnt	263	619	532	339	110	398	0	674	277	280	646	420	475	512
Houston	596	226	243	353	763	260	674	0	415	292	256	202	199	83
Langtry	181	342	494	62	365	271	277	415	0	132	372	293	216	318
Lost Maples	313	234	317	70	525	69	280	292	132	0	264	115	93	202
Padre Island	553	30	426	310	735	262	646	256	372	264	0	246	171	316
Pedernales Falls	434	216	235	231	550	40	420	202	293	115	246	0	75	116
San Antonio	397	141	274	154	564	91	475	199	216	93	171	75	0	154
StataCorp	426	205	151	287	606	148	512	83	318	202	316	116	154	0

Note the inclusion of StataCorp, which is located in the twin cities of Bryan/College Station (BCS). To get the data into Stata, we will enter only the strictly upper triangle as a Stata one-dimensional matrix and collect the names in a `global` macro for later use. We are using the strictly upper triangle (that is, omitting the diagonal) because the diagonal of a dissimilarity matrix contains all zeros—there is no need to enter them.

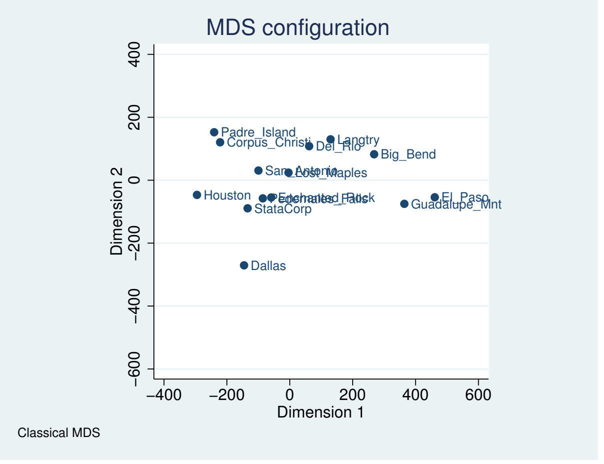
```
. matrix input D = (
>    523  551  243  322  412  263  596  181  313  553  434  397  426
>         396  280  705  232  619  226  342  234   30  216  141  205
>              432  643  230  532  243  494  317  426  235  274  151
>                   427  209  339  353   62   70  310  231  154  287
>                        528  110  763  365  525  735  550  564  606
>                             398  260  271   69  262   40   91  148
>                                  674  277  280  646  420  475  512
>                                       415  292  256  202  199   83
>                                            132  372  293  216  318
>                                                 264  115   93  202
>                                                      246  171  316
>                                                            75  116
>                                                                154 )
. global names
>    Big_Bend      Corpus_Christi  Dallas         Del_Rio
>    El_Paso       Enchanted_Rock  Guadalupe_Mnt  Houston
>    Langtry       Lost_Maples     Padre_Island   Pedernales_Falls
>    San_Antonio   StataCorp
```

The triangular data entry is just typographical and is useful for catching data entry errors. As far as Stata is concerned, we could have typed all the numbers in one long row. We use `matrix input D =` rather than `matrix define D =` or just `matrix D =` so that we do not have to separate entries with commas.

With the data now in Stata, we may use `mdsmat` to infer the locations in Texas and produce a map:

```
. mdsmat D, names($names) shape(uupper)
Classical metric multidimensional scaling
    dissimilarity matrix: D
                                                Number of obs          =       14
    Eigenvalues > 0       =         8           Mardia fit measure 1 =     0.7828
    Retained dimensions   =         2           Mardia fit measure 2 =     0.9823
```

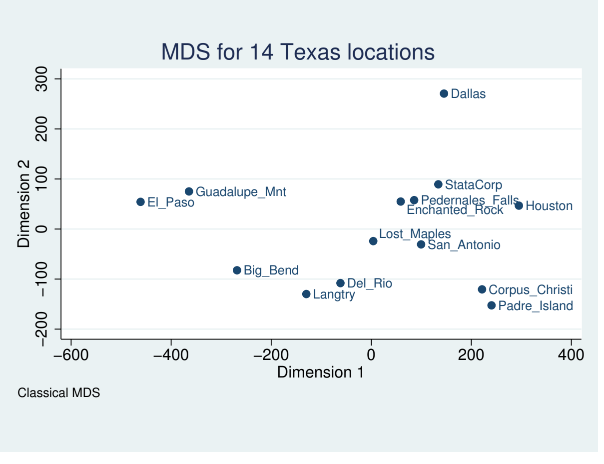
Dimension	Eigenvalue	abs(eigenvalue) Percent	abs(eigenvalue) Cumul.	(eigenvalue)^2 Percent	(eigenvalue)^2 Cumul.
1	691969.62	62.63	62.63	92.45	92.45
2	172983.05	15.66	78.28	5.78	98.23
3	57771.995	5.23	83.51	0.64	98.87
4	38678.916	3.50	87.01	0.29	99.16
5	19262.579	1.74	88.76	0.07	99.23
6	9230.7695	0.84	89.59	0.02	99.25
7	839.70996	0.08	89.67	0.00	99.25
8	44.989372	0.00	89.67	0.00	99.25



The representation of the distances in two dimensions provides a reasonable, but not great, fit; the percentage of eigenvalues accounted for is 78%.

By default, mdsmat produces a configuration plot. Enhancements to the configuration plot are possible using the mdsconfig postestimation graphics command; see [MV] **mds postestimation plots**. We present the configuration plot with the autoaspect option to obtain better use of the available space while preserving the equivalence of distance in the x and y axes. We negate the direction of the x axis with the xnegate option to flip the configuration horizontally and flip the direction of the y axis with the ynegate option. We also change the default title and control the placement of labels.

```
. set obs 14
obs was 0, now 14
. generate pos = 3
. replace pos = 4 in 6
(1 real change made)
. replace pos = 2 in 10
(1 real change made)
. mdsconfig, autoaspect xnegate ynegate mlabvpos(pos)
> title(MDS for 14 Texas locations)
```



Look at the graph produced by `mdsconfig` after `mdsmat`. You will probably recognize a twisted (and slightly distorted) map of Texas. The vertical orientation of the map is not correctly north–south; you would probably want to turn the map some 20 degrees clockwise. Why didn't `mdsmat` get it right? It could not have concluded the correct rotation from the available distance information. Any orthogonal rotation of the map would produce the same distances. The orientation of the map is *not identified*. Finally, the "location" of the map cannot be inferred from the distances. Translating the coordinates does not change the distances. As far as `mdsmat` is concerned, Texas could be part of China.

◁

Modern MDS and local minimums

Modern MDS can converge to a local rather than a global minimum. We give an example where this happens and show how the `protect()` option can guard against this. `protect(#)` performs multiple minimizations and reports the best one. The output is explained in [MV] **mds**.

▷ Example 2

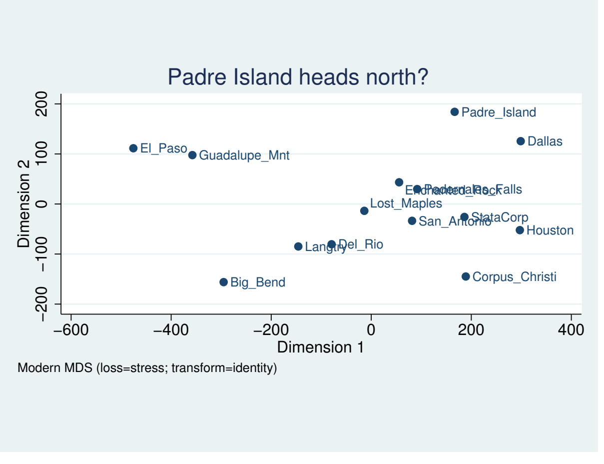
Continuing from where we left off, we perform modern MDS, using an initial random configuration with the `init(random(512308))` option. The number 512,308 sets the seed so that this run may be replicated.

```
. mdsmat D, names($names) shape(uupper) meth(modern) init(random(512308)) nolog
> noplot
(loss(stress) assumed)
(transform(identity) assumed)

Modern multidimensional scaling
    dissimilarity matrix: D

    Loss criterion: stress = raw_stress/norm(distances)
    Transformation: identity (no transformation)
                                              Number of obs    =         14
                                              Dimensions       =          2
    Normalization: principal                  Loss criterion   =     0.1639

. mdsconfig, autoaspect xnegate ynegate mlabvpos(pos)
> title(Padre Island heads north?)
```



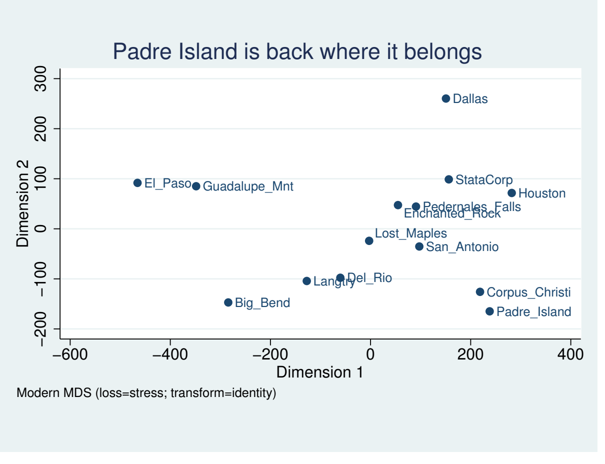
This graph has some resemblance to the one we saw before, but any Texan can assure you that Padre Island should not end up north of Dallas.

We check this result by rerunning with `protect(10)`. This will repeat the minimization and report the best result. Larger values of `protect()` give us more assurance that we have found the global minimum, but here `protect(10)` is sufficient to tell us that our original `mdsmat` found a local, not a global, minimum.

```
. mdsmat D, names($names) shape(uupper) meth(modern) init(random(512308)) nolog
> protect(10) noplot
(loss(stress) assumed)
(transform(identity) assumed)
run  mrc  #iter     lossval    seed random configuration
────────────────────────────────────────────────────────────────────────
  1    0     54   .06180059    X4f2d0d0cc0f1b3343a20359ad2ef90f100013c36
  2    0     50   .06180059    X75445b1482f7cbeca4262f9391b6f5e100013232
  3    0     42    .0618006    X7fa3d0c0ff14ace95d0e18ed6b30fed800013c05
  4    0     54    .0618006    Xbca058982f163fb5735186b9c7f29262000133b9
  5    0     46    .0618006    X7ce217b44e3a967be8b4ef2ca63b100300012ba8
  6    0     47    .0618006    Xcaf38d70eba11618b2169652cd28c9a6000117ca
  7    0     83   .08581202    Xa3aacf488b86c16fde811ec95b22fdf700012b7e
  8    0    111   .08581202    Xaaa0cebc256ebbd6da55f00755942127000138 51
  9    0     74   .08581202    X448b7f64f30e23ca382ef3eeeaec6da4000118b5
 10    0    100    .1639279    X6c6495e8c43f462544a474abacbdd93d00014486
────────────────────────────────────────────────────────────────────────

Modern multidimensional scaling
    dissimilarity matrix: D

    Loss criterion: stress = raw_stress/norm(distances)
    Transformation: identity (no transformation)
                                                  Number of obs    =         14
                                                  Dimensions       =          2
    Normalization: principal                      Loss criterion   =     0.0618
. mdsconfig, autoaspect xnegate ynegate mlabvpos(pos)
> title(Padre Island is back where it belongs)
```



The original run had a loss criterion of 0.1639, but after using the `protect()` option the loss criterion was much lower—0.0618. We also see that Padre Island is back down south where it belongs. It is clear that the original run converged to a local minimum. You can see the original results appear as the final output line of the first table in the output after using `protect(10)`. The seed in the table is a hexadecimal representation of how the seed is stored internally. The number 512,308 in `init(random(512308))` is convenient shorthand for specifying the seed; the two are equivalent. If we wish, we could repeat the command with a larger value of `protect()` to assure ourselves that 0.0618 is indeed the global minimum.

◁

After `mdsmat`, all MDS postestimation tools are available. For instance, you may analyze residuals with `estat quantile`, you may produce a Shepard diagram, etc.; see [MV] **mds postestimation** and [MV] **mds postestimation plots**.

Stored results

`mdsmat` stores the following in `e()`:

Scalars

`e(N)`	number of rows or columns (i.e., number of observations)
`e(p)`	number of dimensions in the approximating configuration
`e(np)`	number of strictly positive eigenvalues
`e(addcons)`	constant added to squared dissimilarities to force positive semidefiniteness
`e(mardia1)`	Mardia measure 1
`e(mardia2)`	Mardia measure 2
`e(critval)`	loss criterion value
`e(wsum)`	sum of weights
`e(alpha)`	parameter of `transform(power)`
`e(ic)`	iteration count
`e(rc)`	return code
`e(converged)`	`1` if converged, `0` otherwise

Macros

`e(cmd)`	`mdsmat`
`e(cmdline)`	command as typed
`e(method)`	`classical` or `modern` MDS method
`e(method2)`	`nonmetric`, if `method(nonmetric)`
`e(loss)`	loss criterion
`e(losstitle)`	description loss criterion
`e(dmatrix)`	name of analyzed matrix
`e(tfunction)`	`identity`, `power`, or `monotonic`, transformation function
`e(transftitle)`	description of transformation
`e(mxlen)`	maximum length of category labels
`e(dtype)`	`similarity` or `dissimilarity`; type of proximity data
`e(s2d)`	`standard` or `oneminus` (when `e(dtype)` is `similarity`)
`e(unique)`	`1` if eigenvalues are distinct, `0` otherwise
`e(init)`	initialization method
`e(iseed)`	seed for `init(random)`
`e(seed)`	seed for solution
`e(norm)`	normalization method
`e(targetmatrix)`	name of target matrix for `normalize(target)`
`e(properties)`	`nob noV` for modern or nonmetric MDS; `nob noV eigen` for classical MDS
`e(estat_cmd)`	program used to implement `estat`
`e(predict)`	program used to implement `predict`
`e(marginsnotok)`	predictions disallowed by `margins`

Matrices

`e(D)`	dissimilarity matrix
`e(Disparities)`	disparity matrix for nonmetric MDS
`e(Y)`	approximating configuration coordinates
`e(Ev)`	eigenvalues
`e(W)`	weight matrix
`e(norm_stats)`	normalization statistics
`e(linearf)`	two element vector defining the linear transformation; distance equals first element plus second element times dissimilarity

Methods and formulas

Methods and formulas are presented under the following headings:

Classical multidimensional scaling

Let $\mathbf{D}$ be an $n \times n$ dissimilarity matrix. The matrix $\mathbf{D}$ is said to be *Euclidean* if there are coordinates $\mathbf{Y}$ so that

$$D_{ij}^2 = (\mathbf{Y}_i - \mathbf{Y}_j)(\mathbf{Y}_i - \mathbf{Y}_j)'$$

Here $\mathbf{Y}_i$ and $\mathbf{Y}_j$ are the ith and jth column vectors extracted from $\mathbf{Y}$. Let $\mathbf{A} = -(1/2)\mathbf{D} \odot \mathbf{D}$, with $\odot$ being the Hadamard or elementwise matrix product, and define $\mathbf{B}$ as the double-centered distance matrix

$$\mathbf{B} = \mathbf{HAH} \qquad \text{with} \qquad \mathbf{H} = \mathbf{I} - \frac{1}{n}\mathbf{1}\mathbf{1}'$$

$\mathbf{D}$ is Euclidean if and only if $\mathbf{B}$ is positive semidefinite. Assume for now that $\mathbf{D}$ is indeed Euclidean. The spectral or eigen decomposition of $\mathbf{B}$ is written as $\mathbf{B} = \mathbf{U}\boldsymbol{\Lambda}\mathbf{U}'$, with $\mathbf{U}$ the orthonormal matrix of eigenvectors normed to 1, and $\boldsymbol{\Lambda}$ a diagonal matrix with nonnegative values (the eigenvalues of $\mathbf{B}$) in decreasing order. The coordinates $\mathbf{Y}$ are defined in terms of the spectral decomposition $\mathbf{Y} = \mathbf{U}\boldsymbol{\Lambda}^{1/2}$. These coordinates are centered $\mathbf{Y}'\mathbf{1} = \mathbf{0}$.

The spectral decomposition can also be used to obtain a low-dimensional configuration $\widetilde{\mathbf{Y}}$, $n \times p$, so that the interrow distances of $\widetilde{\mathbf{Y}}$ approximate $\mathbf{D}$. Mardia, Kent, and Bibby (1979, sec. 14.4) discuss some characterizations under which the leading p columns of $\mathbf{Y}$ are an optimal choice of $\widetilde{\mathbf{Y}}$. These characterizations also apply to the case when $\mathbf{B}$ is not positive semidefinite, so some of the λ's are negative; we require that $\lambda_p > 0$.

Various other approaches have been proposed to deal with the case when the matrix $\mathbf{B}$ is not positive semidefinite, that is, when $\mathbf{B}$ has negative eigenvalues (see Cox and Cox 2001, 45–48). An easy solution is to add a constant to the off-diagonal elements of $\mathbf{D} \odot \mathbf{D}$ to make $\mathbf{B}$ positive semidefinite. The smallest such constant is $-2\lambda_n$, where λ_n is the smallest eigenvalue of $\mathbf{B}$ (Lingoes 1971). See Cailliez (1983) for a solution to the additive constant problem in terms of the dissimilarities instead of the squared dissimilarities.

Goodness-of-fit statistics for a configuration in p dimensions have been proposed by Mardia (1978) in characterizations of optimality properties of the classical solution

$$\text{Mardia}_1 = \frac{\sum_{i=1}^{p} |\lambda_i|}{\sum_{i=1}^{n} |\lambda_i|}$$

and

$$\text{Mardia}_2 = \frac{\sum_{i=1}^{p} \lambda_i^2}{\sum_{i=1}^{n} \lambda_i^2}$$

Modern multidimensional scaling

Let $\mathbf{D}$ be a symmetric $n \times n$ matrix of observed dissimilarities. We assume that proximity data in the form of similarities have already been transformed into dissimilarities. Let $\mathbf{W}$ be an $n \times n$ matrix of nonnegative weights. With unweighted MDS, we define $W_{ij} = 1$. For a configuration of n points in k-dimensional space represented by the $n \times k$ matrix $\mathbf{Y}$, let $\mathbf{B}(\mathbf{Y})$ be the $n \times n$ matrix of Euclidean distances between the rows of $\mathbf{Y}$. We consider $\mathcal{F}$ to be some class of permitted transformations from $n \times n$ real matrices to $n \times n$ real matrices.

Modern metric and nonmetric multidimensional scaling involves the minimization of a loss criterion

$$L\{f(\mathbf{D}), \mathbf{B}(\mathbf{Y}), \mathbf{W}\}$$

over the configurations $\mathbf{Y}$ and transformations $f \in \mathcal{F}$. Whether a scaling method is labeled metric or nonmetric depends on the class $\mathcal{F}$. In nonmetric scaling, $\mathcal{F}$ is taken to be the class of monotonic functions. If $\mathcal{F}$ is a regular parametrized set of functions, one commonly labels the scaling as metric.

$\mathbf{D}$ is the matrix of proximities or dissimilarities, $\mathbf{B}(\mathbf{Y})$ is the matrix of distances, and the result of $f(\mathbf{D}) = \widehat{\mathbf{D}}$ is the matrix of disparities.

The `mdsmat` command supports the following loss criteria:

(1) **stress** specifies Kruskal's stress-1 criterion: the Euclidean norm of the difference between the distances and the disparities, normalized by the Euclidean norm of the distances.

$$\textbf{stress}(\widehat{\mathbf{D}}, \mathbf{B}, \mathbf{W}) = \left\{ \frac{\sum_{ij} W_{ij}(B_{ij} - \widehat{D}_{ij})^2}{\sum_{ij} W_{ij} B_{ij}^2} \right\}^{1/2}$$

(2) **nstress** specifies the square root of the normalized stress criterion: the Euclidean norm of the difference between the distances and the disparities, normalized by the Euclidean norm of the disparities.

$$\textbf{nstress}(\widehat{\mathbf{D}}, \mathbf{B}, \mathbf{W}) = \left\{ \frac{\sum_{ij} W_{ij}(B_{ij} - \widehat{D}_{ij})^2}{\sum_{ij} W_{ij} \widehat{D}_{ij}^2} \right\}^{1/2}$$

nstress normalizes with the disparities, **stress** with the distances.

(3) **sammon** specifies the Sammon mapping criterion (Sammon 1969; Neimann and Weiss 1979): the sum of the scaled, squared differences between the distances and the disparities, normalized by the sum of the disparities.

$$\textbf{sammon}(\widehat{\mathbf{D}}, \mathbf{B}, \mathbf{W}) = \frac{\sum_{ij} W_{ij}(B_{ij} - \widehat{D}_{ij})^2 / \widehat{D}_{ij}}{\sum_{ij} W_{ij} \widehat{D}_{ij}}$$

(4) **sstress** specifies the squared stress criterion: the Euclidean norm of the difference between the squared distances and the squared disparities, normalized by the Euclidean norm of the squared distances.

$$\textbf{sstress}(\widehat{\mathbf{D}}, \mathbf{B}, \mathbf{W}) = \left\{ \frac{\sum_{ij} W_{ij}(B_{ij}^2 - \widehat{D}_{ij}^2)^2}{\sum_{ij} W_{ij} B_{ij}^4} \right\}^{1/2}$$

(5) **nsstress** specifies the normalized squared stress criterion: the Euclidean norm of the difference between the squared distances and the squared disparities, normalized by the Euclidean norm of the squared disparities.

$$\textbf{nsstress}(\widehat{\mathbf{D}}, \mathbf{B}, \mathbf{W}) = \left\{ \frac{\sum_{ij} W_{ij}(B_{ij}^2 - \widehat{D}_{ij}^2)^2}{\sum_{ij} W_{ij}\widehat{D}_{ij}^4} \right\}^{1/2}$$

nsstress normalizes with the disparities, **sstress** with the distances.

(6) **strain** specifies the strain criterion:

$$\textbf{strain}(\widehat{\mathbf{D}}, \mathbf{B}, \mathbf{W}) = \frac{\sqrt{\text{trace}(\mathbf{X}'\mathbf{X})}}{\sum_{ij} W_{ij}}$$

where

$$\mathbf{X} = \mathbf{W} \odot \left\{ \widehat{\mathbf{D}} - \mathbf{B}(\widetilde{\mathbf{Y}}) \right\}$$

where $\widetilde{\mathbf{Y}}$ is the centered configuration of $\mathbf{Y}$. Without weights, $W_{ij} = 1$, and without transformation, that is, $\widehat{\mathbf{D}} = \mathbf{D}$, minimization of the strain criterion is equivalent to classical metric scaling.

The `mdsmat` command supports three classes of permitted transformations, $f \in \mathcal{F}$: (1) the class of all weakly monotonic transformations, (2) the power class of functions where f is defined elementwise on $\mathbf{D}$ as $f(D_{ij}, \alpha) = D_{ij}^{\alpha}$ (Critchley 1978; Cox and Cox 2001), and (3) the trivial identity case of $f(\mathbf{D}) = \mathbf{D}$.

Minimization of a loss criterion with respect to the configuration $\mathbf{Y}$ and the permitted transformation $f \in \mathcal{F}$ is performed with an alternating algorithm in which the configuration $\mathbf{Y}$ is modified (the C-step) and the transformation f is adjusted (the T-step) to reduce loss. Obviously, no T-step is made with the identity transformation. The classical solution is the default starting configuration. Iteration continues until the C-step and T-step reduce loss by less than the tolerance for convergence or the maximum number of iterations is performed. The C-step is taken by steepest descent using analytical gradients and an optimal stepsize computed using Brent's bounded minimization (Brent 1973). The implementation of the T-step varies with the specified class of transformations. In the nonmetric case of monotonic transformations, we use isotonic regression (Kruskal 1964a, 1964b; Cox and Cox 2001), using the primary approach to ties (Borg and Groenen 2005, 40). For power transformations, we again apply Brent's minimization method.

Given enough iterations, convergence is usually not a problem. However, the alternating algorithm may not converge to a global minimum. `mdsmat` provides some protection by repeated runs from different initial configurations. However, as Euclidean distances $\mathbf{B}(\mathbf{Y})$ are invariant with respect to isometric transformations (rotations, translations) of $\mathbf{Y}$, some caution is required to compare different runs and, similarly, to compare the configurations obtained from different scaling methods. `mdsmat` normalizes the optimal configuration by centering and via the orthogonal Procrustean rotation without dilation toward the classical or a user-specified solution; see [MV] **procrustes**.

Conversion of similarities to dissimilarities

If a similarity measure was selected, it is turned into a dissimilarity measure by using one of two methods. The *standard* conversion method is

$$\text{dissim}_{ij} = \sqrt{\text{sim}_{ii} + \text{sim}_{jj} - 2\text{sim}_{ij}}$$

With the similarity of an object to itself being 1, this is equivalent to

$$\text{dissim}_{ij} = \sqrt{2(1 - \text{sim}_{ij})}$$

This conversion method has the attractive property that it transforms a positive-semidefinite similarity matrix into a Euclidean distance matrix (see Mardia, Kent, and Bibby 1979, 402).

We also offer the *one-minus* method

$$\text{dissim}_{ij} = 1 - \text{sim}_{ij}$$

References

Borg, I., and P. J. F. Groenen. 2005. *Modern Multidimensional Scaling: Theory and Applications*. 2nd ed. New York: Springer.

Brent, R. P. 1973. *Algorithms for Minimization without Derivatives*. Englewood Cliffs, NJ: Prentice Hall. (Reprinted in paperback by Dover Publications, Mineola, NY, January 2002).

Cailliez, F. 1983. The analytical solution of the additive constant problem. *Psychometrika* 48: 305–308.

Cox, T. F., and M. A. A. Cox. 2001. *Multidimensional Scaling*. 2nd ed. Boca Raton, FL: Chapman & Hall/CRC.

Critchley, F. 1978. Multidimensional scaling: A short critique and a new method. In *COMPSTAT 1978: Proceedings in Computational Statistics*, ed. L. C. A. Corsten and J. Hermans. Vienna: Physica.

Kruskal, J. B. 1964a. Multidimensional scaling by optimizing goodness of fit to a nonmetric hypothesis. *Psychometrika* 29: 1–27.

——. 1964b. Nonmetric multidimensional scaling: A numerical method. *Psychometrika* 29: 115–129.

Kruskal, J. B., and M. Wish. 1978. *Multidimensional Scaling*. Newbury Park, CA: Sage.

Lingoes, J. C. 1971. Some boundary conditions for a monotone analysis of symmetric matrices. *Psychometrika* 36: 195–203.

Mardia, K. V. 1978. Some properties of classical multidimensional scaling. *Communications in Statistics, Theory and Methods* 7: 1233–1241.

Mardia, K. V., J. T. Kent, and J. M. Bibby. 1979. *Multivariate Analysis*. London: Academic Press.

Neimann, H., and J. Weiss. 1979. A fast-converging algorithm for nonlinear mapping of high-dimensional data to a plane. *IEEE Transactions on Computers* 28: 142–147.

Sammon, J. W., Jr. 1969. A nonlinear mapping for data structure analysis. *IEEE Transactions on Computers* 18: 401–409.

Torgerson, W. S. 1952. Multidimensional scaling: I. Theory and method. *Psychometrika* 17: 401–419.

Young, F. W., and R. M. Hamer. 1987. *Multidimensional Scaling: History, Theory, and Applications*. Hillsdale, NJ: Erlbaum Associates.

Young, G., and A. S. Householder. 1938. Discussion of a set of points in terms of their mutual distances. *Psychometrika* 3: 19–22.

Also see

[MV] **mds postestimation** — Postestimation tools for mds, mdsmat, and mdslong

[MV] **mds postestimation plots** — Postestimation plots for mds, mdsmat, and mdslong

[MV] **biplot** — Biplots

[MV] **ca** — Simple correspondence analysis

[MV] **factor** — Factor analysis

[MV] **mds** — Multidimensional scaling for two-way data

[MV] **mdslong** — Multidimensional scaling of proximity data in long format

[MV] **pca** — Principal component analysis

[U] 20 Estimation and postestimation commands