Title

stata.com

mdslong — Multidimensional scaling of proximity data in long format

Description

`mdslong` performs multidimensional scaling (MDS) for two-way proximity data in long format with an explicit measure of similarity or dissimilarity between objects. `mdslong` performs classical metric MDS as well as modern metric and nonmetric MDS.

For MDS with two-way proximity data in a matrix, see [MV] **mdsmat**. If you are looking for MDS on a dataset, based on dissimilarities between observations over variables, see [MV] **mds**.

Quick start

Classical multidimensional scaling based on dissimilarities in variable `d` between subjects identified by variables `i` and `j`

```
mdslong d, id(i j)
```

As above, but suppress the MDS configuration plot and use 3 dimensions for the approximating configuration

```
mdslong d, id(i j) noplot dimension(3)
```

Modern multidimensional scaling

```
mdslong d, id(i j) method(modern)
```

As above, but with Sammon mapping loss criterion and Procrustes rotation toward the classical solution

```
mdslong d, id(i j) loss(sammon) normalize(classical)
```

Nonmetric modern multidimensional scaling

```
mdslong d, id(i j) method(nonmetric)
```

Menu

Statistics > Multivariate analysis > Multidimensional scaling (MDS) > MDS of proximity-pair data

Syntax

```
mdslong depvar [if] [in] [weight], id(var1 var2) [options]
```

options	Description
Model	
* `id(`*var*$_1$ *var*$_2$`)`	identify comparison pairs (object$_1$, object$_2$)
`method(`*method*`)`	method for performing MDS
`loss(`*loss*`)`	loss function
`transform(`*tfunction*`)`	permitted transformations of dissimilarities
`normalize(`*norm*`)`	normalization method; default is `normalize(principal)`
`s2d(standard)`	convert similarity to dissimilarity: $\text{dissim}_{ij} = \sqrt{\text{sim}_{ii} + \text{sim}_{jj} - 2\text{sim}_{ij}}$; the default
`s2d(oneminus)`	convert similarity to dissimilarity: $\text{dissim}_{ij} = 1 - \text{sim}_{ij}$
`force`	correct problems in proximity information
`dimension(#)`	configuration dimensions; default is `dimension(2)`
`addconstant`	make distance matrix positive semidefinite (classical MDS only)
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 description below

* `id(`*var*$_1$ *var*$_2$`)` is required.

`by` and `statsby` are allowed; see **[U] 11.1.10 Prefix commands**.

`aweights` and `fweights` are allowed for methods `modern` and `nonmetric`; see **[U] 11.1.6 weight**.

The maximum number of compared objects allowed is the maximum matrix size; see **[R] matsize**.

`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

Options

Model

id(*var*$_1$ *var*$_2$) is required. The pair of variables *var*$_1$ and *var*$_2$ should uniquely identify comparisons. *var*$_1$ and *var*$_2$ are string or numeric variables that identify the objects to be compared. *var*$_1$ and *var*$_2$ should be of the same data type; if they are value labeled, they should be labeled with the same value label. Using value-labeled variables or string variables is generally helpful in identifying the points in plots and tables.

Example data layout for `mdslong proxim, id(i1 i2)`.

proxim	i1	i2
7	1	2
10	1	3
12	1	4
4	2	3
6	2	4
3	3	4

If you have multiple measurements per pair, we suggest that you specify the mean of the measures as the proximity and the inverse of the variance as the weight.

`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.

`s2d(standard`|`oneminus)` specifies how similarities are converted into dissimilarities. By default, the command assumes 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`	$\text{dissim}_{ij} = \sqrt{\text{sim}_{ii} + \text{sim}_{jj} - 2\text{sim}_{ij}} = \sqrt{2(1 - \text{sim}_{ij})}$
`oneminus`	$\text{dissim}_{ij} = 1 - \text{sim}_{ij}$

`s2d(standard)` is the default.

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

`force` corrects problems with the supplied proximity information. In the long format used by `mdslong`, multiple measurements on (i,j) may be available. Including both (i,j) and (j,i) would be treated as multiple measurements. This is an error, even if the measures are identical. Option `force` uses

the mean of the measurements. `force` also resolves problems on the diagonal, that is, comparisons of objects with themselves; these should have zero dissimilarity or unit similarity. `force` does not resolve incomplete data, that is, pairs (i, j) for which no measurement is available. Out-of-range values are also not fixed.

`dimension(#)` specifies the dimension of the approximating configuration. The default is `dimension(2)`, and # should not exceed the number of positive eigenvalues of the centered distance 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. This option is allowed with classical MDS only.

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 `mdslong` 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.

`mdslong` performs MDS on data in long format. *depvar* specifies proximity data in either dissimilarity or similarity form. The comparison pairs are identified by two variables specified in the required option `id()`. Exactly 1 observation with a nonmissing *depvar* should be included for each pair (i, j). Pairs are unordered; you do not include observations for both (i, j) and (j, i). Observations for comparisons of objects with themselves (i, i) are optional. See option `force` if your data violate these assumptions.

When you have multiple independent measurements of the dissimilarities, you may specify the mean of these dissimilarity measurements as the combined measurement and specify $1/(\#\text{of measurements})$ or $1/\text{variance(measurements)}$ as weights. For more discussion of weights in MDS, we refer to Borg and Groenen (2005, sec. 11.3). Weights should be irreducible; that is, it is not possible to split the objects into disjointed groups with all intergroup weights 0.

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 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. `mdslong` 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.

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. `mdslong` provides 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.

Proximity data in long format

One format for proximity data is called the "long format", with an observation recording the dissimilarity d_{ij} of the "objects" i and j. This requires three variables: one variable to record the dissimilarities and two variables to identify the comparison pair. The MDS command `mdslong` requires

- Complete data without duplicates: there is exactly 1 observation for each combination (i, j) or (j, i).
- Optional diagonal: you may, but need not, specify dissimilarities for the reflexive pairs (i, i). If you do, you need not supply values for all (i, i).

▷ Example 1

We illustrate the use of `mdslong` with a popular dataset from the MDS literature. Rothkopf (1957) had 598 subjects listen to pairs of Morse codes for the 10 digits and for the 26 letters, recording for each pair of codes the percentage of subjects who declared the codes to be the same. The data on the 10 digits are reproduced in Mardia, Kent, and Bibby (1979, 395).

```
. use http://www.stata-press.com/data/r14/morse_long
(Morse data (Rothkopf 1957))

. list in 1/10
```

	digit1	digit2	freqsame
1.	2	1	62
2.	3	1	16
3.	3	2	59
4.	4	1	6
5.	4	2	23
6.	4	3	38
7.	5	1	12
8.	5	2	8
9.	5	3	27
10.	5	4	56

Sixty-two percent of the subjects declare that the Morse codes for 1 and 2 are the same, 16% declare that 1 and 3 are the same, 59% declare 2 and 3 to be the same, etc. We may think that these percentages are similarity measures between the Morse codes: the more similar two Morse codes, the higher the percentage is of subjects who do not distinguish them. The reported percentages suggest, for example, that 1 and 2 are similar to approximately the same extent as 2 and 3, whereas 1 and 3 are much less similar. This is the kind of relationship you would expect with data that can be adequately represented with MDS.

We transform our data to a zero-to-one scale.

```
. generate sim = freqsame/100
```

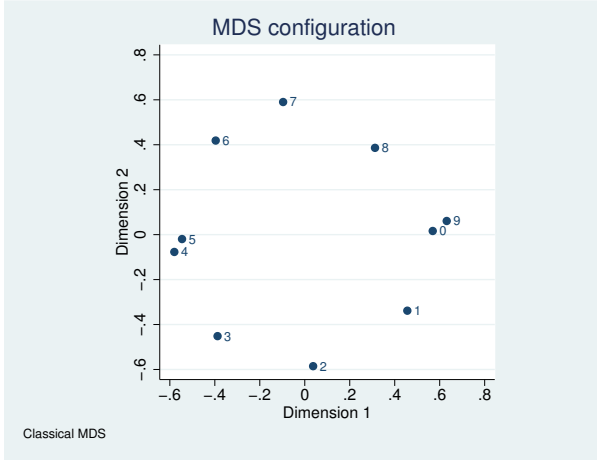
and invoke `mdslong` on `sim`, specifying that the proximity variable `sim` be interpreted as similarities, and we use option `s2d(standard)` to convert to dissimilarities by using the standard conversion.

```
. mdslong sim, id(digit1 digit2) s2d(standard)

Classical metric multidimensional scaling
    similarity variable: sim in long format
          dissimilarity: sqrt(2(1-similarity))

                                        Number of obs        =         10
Eigenvalues > 0      =         9        Mardia fit measure 1 =     0.5086
Retained dimensions  =         2        Mardia fit measure 2 =     0.7227
```

Dimension	Eigenvalue	abs(eigenvalue) Percent	abs(eigenvalue) Cumul.	(eigenvalue)^2 Percent	(eigenvalue)^2 Cumul.
1	1.9800226	30.29	30.29	49.47	49.47
2	1.344165	20.57	50.86	22.80	72.27
3	1.063133	16.27	67.13	14.26	86.54
4	.66893922	10.23	77.36	5.65	92.18
5	.60159396	9.20	86.56	4.57	96.75
6	.42722301	6.54	93.10	2.30	99.06
7	.21220785	3.25	96.35	0.57	99.62
8	.1452025	2.22	98.57	0.27	99.89
9	.09351288	1.43	100.00	0.11	100.00



The two-dimensional representation provides a reasonable, but certainly not impressive, fit to the data. The plot itself is interesting, though, with the digits being roughly 45 degrees apart, except for the pairs (0,9) and (4,5), which are mapped almost at the same locations. Interpretation is certainly helped if you see the circular structure in the Morse codes.

digit	morse code
1	. - - - -
2	. . - - -
3	. . . - -
4	. . . . -
5	. . . . .
6	- . . . .
7	- - . . .
8	- - - . .
9	- - - - .
0	- - - - -

Example 2

You might have your data in wide instead of long format. The Morse code dataset in wide format has 10 observations, 10 data variables `d1`, ..., `d9`, `d0`, and one case identifier.

```
. use http://www.stata-press.com/data/r14/morse_wide, clear
(Morse data (Rothkopf 1957))
. describe
Contains data from http://www.stata-press.com/data/r14/morse_wide.dta
  obs:            10                          Morse data (Rothkopf 1957)
 vars:            11                          14 Feb 2014 20:28
 size:           110                          (_dta has notes)
```

variable name	storage type	display format	value label	variable label
digit	byte	%9.0g		
d1	byte	%9.0g		
d2	byte	%9.0g		
d3	byte	%9.0g		
d4	byte	%9.0g		
d5	byte	%9.0g		
d6	byte	%9.0g		
d7	byte	%9.0g		
d8	byte	%9.0g		
d9	byte	%9.0g		
d0	byte	%9.0g		

```
Sorted by:
. list
```

	digit	d1	d2	d3	d4	d5	d6	d7	d8	d9	d0
1.	1	84	62	16	6	12	12	20	37	57	52
2.	2	62	89	59	23	8	14	25	25	28	18
3.	3	16	59	86	38	27	33	17	16	9	9
4.	4	6	23	38	89	56	34	24	13	7	7
5.	5	12	8	27	56	90	30	18	10	5	5
6.	6	12	14	33	34	30	86	65	22	8	18
7.	7	20	25	17	24	18	65	85	65	31	15
8.	8	37	25	16	13	10	22	65	88	58	39
9.	9	57	28	9	7	5	8	31	58	91	79
10.	0	52	18	9	7	5	18	15	39	79	94

Stata does not provide an MDS command to deal directly with the wide format because it is easy to convert the wide format into the long format with the `reshape` command; see [D] **reshape**.

```
. reshape long d, i(digit) j(other)
(note: j = 0 1 2 3 4 5 6 7 8 9)
Data                               wide   ->   long
-----------------------------------------------------------------------------
Number of obs.                       10   ->      100
Number of variables                  11   ->        3
j variable (10 values)                    ->   other
xij variables:
                           d0 d1 ... d9   ->   d
-----------------------------------------------------------------------------
```

Now our data are in long format, and we can use `mdslong` to obtain a MDS analysis.

```
. generate sim = d/100
. mdslong sim, id(digit other) s2d(standard) noplot
objects should have unit similarity to themselves
r(198);
```

`mdslong` complains. The wide data—and hence also the long data that we now have—also contain the frequencies in which two identical Morse codes were recognized as the same. This is not 100%. Auditive memory is not perfect, posing a problem for the standard MDS model. We can solve this by ignoring the diagonal observations:

```
. mdslong ... if digit != other ...
```

We may also specify the `force` option. The `force` option will take care of a problem that has not yet surfaced, namely, that `mdslong` requires 1 observation for each pair (i, j). In the long data as now created, we have duplicates observations (i, j) and (j, i). `force` will symmetrize the proximity information, but it will not deal with multiple measurements in general; having 2 or more observations for (i, j) is an error. If you have multiple measurements, you may average the measurements and use weights.

```
. mdslong sim, id(digit other) s2d(standard) force noplot
Classical metric multidimensional scaling
    similarity variable: sim in long format
          dissimilarity: sqrt(2(1-similarity))
                                        Number of obs        =         10
Eigenvalues > 0      =          9       Mardia fit measure 1 =     0.5086
Retained dimensions  =          2       Mardia fit measure 2 =     0.7227
```

Dimension	Eigenvalue	abs(eigenvalue) Percent	abs(eigenvalue) Cumul.	(eigenvalue)^2 Percent	(eigenvalue)^2 Cumul.
1	1.9800226	30.29	30.29	49.47	49.47
2	1.344165	20.57	50.86	22.80	72.27
3	1.063133	16.27	67.13	14.26	86.54
4	.66893922	10.23	77.36	5.65	92.18
5	.60159396	9.20	86.56	4.57	96.75
6	.42722301	6.54	93.10	2.30	99.06
7	.21220785	3.25	96.35	0.57	99.62
8	.1452025	2.22	98.57	0.27	99.89
9	.09351288	1.43	100.00	0.11	100.00

The output produced by `mdslong` here is identical to what we saw earlier.

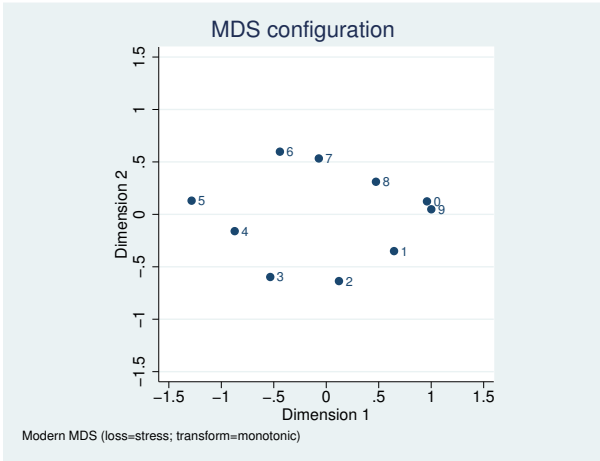
Modern nonmetric MDS

Unlike classical MDS, modern MDS is calculated via the minimization of the loss function. Eigenvalues are no longer calculated. We look at nonmetric MDS, which is a type of modern MDS in which the transformation from distances to disparities is not an identifiable function as in modern metric MDS but is instead a general monotonic function.

Example 3

We return to the Rothkopf (1957) Morse codes in long format. When we specify `method(nonmetric)`, we assume `loss(stress)` and `transform(monotonic)`.

```
. use http://www.stata-press.com/data/r14/morse_long, clear
(Morse data (Rothkopf 1957))
. generate sim = freqsame/100
. mdslong sim, id(digit1 digit2) s2d(standard) meth(nonmetric)
(loss(stress) assumed)
(transform(monotonic) assumed)
Iteration 1t:   stress =  .14719847
Iteration 1c:   stress =  .11378737
  (output omitted)
Iteration 89t:   stress =  .07228281
Iteration 89c:   stress =  .07228281
Modern multidimensional scaling
    similarity variable: sim in long format
          dissimilarity: sqrt(2(1-similarity))
    Loss criterion: stress = raw_stress/norm(distances)
    Transformation: monotonic (nonmetric)
                                                Number of obs    =         10
                                                Dimensions       =          2
    Normalization: principal                    Loss criterion   =     0.0723
```



Each iteration has two steps associated with it. The two parts to each iteration consist of modifying the transformation (the T-step) and modifying the configuration (the C-step). If the `transform(identity)` option had been used, there would not be a T-step. In the iteration log, you see these as *Iteration 1t* and *Iteration 1c*. The rest of the output from modern MDS is explained in [MV] **mds**.

Although there is a resemblance between this graph and the first graph, the points are not as circular or as evenly spaced as they are in the first example, and a great deal more distance is seen between points 4 and 5. Nonmetric MDS depends only on the ordinal properties of the data and admits transformations that may radically change the appearance of the configuration.

After mdslong, 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

mdslong stores the following in e():

Scalars

e(N)	number of underlying 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(npos)	number of pairs with positive weights
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)	mdslong
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(tfunction)	identity, power, or monotonic, transformation function
e(transftitle)	description of transformation
e(id)	two ID variable names identifying compared object pairs
e(idtype)	int or str; type of id() variable
e(duplicates)	1 if duplicates in id(), 0 otherwise
e(labels)	labels for ID categories
e(mxlen)	maximum length of category labels
e(depvar)	dependent variable containing dissimilarities
e(dtype)	similarity or dissimilarity; type of proximity data
e(s2d)	standard or oneminus (when e(dtype) is similarity)
e(wtype)	weight type
e(wexp)	weight expression
e(unique)	1 if eigenvalues are distinct, 0 otherwise
e(init)	initialization method
e(irngstate)	initial random-number state used for init(random)
e(rngstate)	random-number state 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(idcoding)	coding for integer identifier variable
e(norm_stats)	normalization statistics
e(linearf)	two-element vector defining the linear transformation; distance equals first element plus second element times dissimilarity

Functions

e(sample)	marks estimation sample

Methods and formulas

See *Methods and formulas* in [MV] **mdsmat** for information.

References

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

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

Rothkopf, E. Z. 1957. A measure of stimulus similarity and errors in some paired-associate learning tasks. *Journal of Experimental Psychology* 53: 94–101.

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.

See [MV] **mdsmat** for more references.

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] **mdsmat** — Multidimensional scaling of proximity data in a matrix

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

[U] **20 Estimation and postestimation commands**