

Title

mds — Multidimensional scaling for two-way data

Syntax

`mds varlist [if] [in], id(varname) [options]`

<i>options</i>	description
Model	
* <code>id(varname)</code>	identify observations
<code>method(method)</code>	method for performing MDS
<code>loss(loss)</code>	loss function
<code>transform(function)</code>	permitted transformations of dissimilarities
<code>normalize(norm)</code>	normalization method; default is <code>normalize(principal)</code>
<code>dimension(#)</code>	configuration dimensions; default is <code>dimension(2)</code>
<code>addconstant</code>	make distance matrix positive semidefinite
Model 2	
<code>unit[(varlist₂)]</code>	scale variables to min = 0 and max = 1
<code>std[(varlist₃)]</code>	scale variables to mean = 0 and sd = 1
<code>measure(measure)</code>	similarity or dissimilarity measure; default is L2 (Euclidean)
<code>s2d(standard)</code>	convert similarity to dissimilarity: $\text{dissim}_{ij} = \sqrt{\text{sim}_{ii} + \text{sim}_{jj} - 2\text{sim}_{ij}}$; the default
<code>s2d(oneminus)</code>	convert similarity to dissimilarity: $\text{dissim}_{ij} = 1 - \text{sim}_{ij}$
Reporting	
<code>neigen(#)</code>	maximum number of eigenvalues to display; default is <code>neigen(10)</code>
<code>config</code>	display table with configuration coordinates
<code>noplot</code>	suppress configuration plot
Minimization	
<code>initialize(inopt)</code>	start with configuration given in <i>inopt</i>
<code>tolerance(#)</code>	tolerance for configuration matrix; default is <code>tolerance(1e-4)</code>
<code>ltolerance(#)</code>	tolerance for loss criterion; default is <code>ltolerance(1e-8)</code>
<code>iterate(#)</code>	perform maximum # of iterations; default is <code>iterate(1000)</code>
<code>protect(#)</code>	perform # optimizations and report best solution; default is <code>protect(1)</code>
<code>nolog</code>	suppress the iteration log
<code>trace</code>	display current configuration in iteration log
<code>gradient</code>	display current gradient matrix in iteration log
† <code>sdprotect(#)</code>	advanced; see description below

* `id(varname)` is required.

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

`bootstrap`, `by`, `jackknife`, `rolling`, `statsby`, and `xi` are allowed; see [U] **11.1.10 Prefix commands**.

The maximum number of observations allowed in `mds` is the maximum matrix size; see [R] `matsize`.

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

<i>method</i>	description
<u>classical</u>	classical MDS; default if neither <code>loss()</code> nor <code>transform()</code> is specified
<u>modern</u>	modern MDS; default if <code>loss()</code> or <code>transform()</code> is specified; except when <code>loss(stress)</code> and <code>transform(monotonic)</code> are specified
<u>nonmetric</u>	nonmetric (modern) MDS; default when <code>loss(stress)</code> and <code>transform(monotonic)</code> are specified

<i>loss</i>	description
<u>stress</u>	stress criterion, normalized by distances; the default
<u>nstress</u>	stress criterion, normalized by disparities
<u>sstress</u>	squared stress criterion, normalized by distances
<u>nsstress</u>	squared stress criterion, normalized by disparities
<u>strain</u>	strain criterion (with <code>transform(identity)</code> is equivalent to classical MDS)
<u>sammon</u>	Sammon mapping

<i>tfunction</i>	description
<u>identity</u>	no transformation; disparity = dissimilarity; the default
<u>power</u>	power α : disparity = dissimilarity ^{α}
<u>monotonic</u>	weakly monotonic increasing functions (nonmetric scaling); only with <code>loss(stress)</code>

<i>norm</i>	description
<u>principal</u>	principal orientation; location = 0; the default
<u>classical</u>	Procrustes rotation toward classical solution
<u>target(matname) [, copy]</u>	Procrustes rotation toward <i>matname</i> ; ignore naming conflicts if <code>copy</code> is specified

<i>initopt</i>	description
<u>classical</u>	start with classical solution; the default
<u>random [(#)]</u>	start at random configuration, setting seed to #
<u>from(matname) [, copy]</u>	start from <i>matname</i> ; ignore naming conflicts if <code>copy</code> is specified

Description

`mds` performs multidimensional scaling (MDS) for dissimilarities between observations with respect to the variables in *varlist*. A wide selection of similarity and dissimilarity measures is available; see the `measure()` option. `mds` performs classical metric MDS (Torgerson 1952) as well as modern metric and nonmetric MDS; see the `loss()` and `transform()` options.

`mds` computes dissimilarities from the observations; `mdslong` and `mdsmat` are for use when you already have proximity information. `mdslong` and `mdsmat` offer the same statistical features but require different data organizations. `mdslong` expects the proximity information (and, optionally, weights) in a “long format” (pairwise or dyadic form), whereas `mdsmat` performs MDS on symmetric proximity and weight matrices; see [MV] **mdslong** and [MV] **mdsmat**.

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

`id(varname)` is required and specifies a variable that identifies observations. A warning message is displayed if *varname* has duplicate values.

`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, i.e., 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; i.e., disparities are equal to dissimilarities. `transform(identity)` is the default.

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

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

`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; i.e., 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 (i.e., 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.

`dimension(#)` specifies the dimension of the approximating configuration. The default `#` is 2 and should not exceed the number of observations; typically, `#` would be much smaller. With `method(classical)`, it 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. `addconstant` is allowed with classical MDS only.

Model 2

`unit` [*varlist*₂] specifies variables that are transformed to min = 0 and max = 1 before entering in the computation of similarities or dissimilarities. `unit` by itself, without an argument, is a shorthand for `unit(_all)`. Variables in `unit()` should not be included in `std()`.

`std` [*varlist*₃] specifies variables that are transformed to mean = 0 and sd = 1 before entering in the computation of similarities or dissimilarities. `std` by itself, without an argument, is a shorthand for `std(_all)`. Variables in `std()` should not be included in `unit()`.

`measure` (*measure*) specifies the similarity or dissimilarity measure. The default is `measure(L2)`, Euclidean distance. This option is not case sensitive. See [MV] *measure_option* for detailed descriptions of the supported measures.

If a similarity measure is selected, the computed similarities will first be transformed into dissimilarities, before proceeding with the scaling; see the `s2d()` option below.

Classical metric MDS with Euclidean distance is equivalent to principal component analysis (see [MV] `pca`); the MDS configuration coordinates are the principal components.

`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 (i.e., 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 (i.e., 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:

$$\begin{array}{ll} \text{standard} & \text{dissim}_{ij} = \sqrt{\text{sim}_{ii} + \text{sim}_{jj} - 2\text{sim}_{ij}} = \sqrt{2(1 - \text{sim}_{ij})} \\ \text{oneminus} & \text{dissim}_{ij} = 1 - \text{sim}_{ij} \end{array}$$

`s2d(standard)` is the default.

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

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*.

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 `mds` 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

Remarks are presented under the following headings:

- Introduction*
- Euclidean distances*
- Non-Euclidean dissimilarity measures*
- Introduction to modern MDS*
- Protecting from local minimums*

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 (1994) and Borg and Groenen (2005) offer more advanced textbook-sized treatments.

If you already have the similarities or dissimilarities of the n objects, you should continue by reading [MV] **mdsmat**.

In many applications of MDS, however, the similarity or dissimilarity of objects is not measured but rather *defined* by the researcher in terms of variables (“attributes”) x_1, \dots, x_k that are measured on the objects. The pairwise dissimilarity of objects can be expressed using a variety of similarity or dissimilarity measures in the attributes (e.g., Mardia, Kent, and Bibby 1979, sec. 13.4; Cox and Cox 2001, sec. 1.3). A common measure is the Euclidean distance L2 between the attributes of the objects i and j :

$$L2_{ij} = \{(x_{i1} - x_{j1})^2 + (x_{i2} - x_{j2})^2 + \dots + (x_{ik} - x_{jk})^2\}^{1/2}$$

A popular alternative is the L1 distance, also known as the **cityblock** or **Manhattan** distance. In comparison to L2, L1 gives less influence to larger differences in attributes:

$$L1_{ij} = |x_{i1} - x_{j1}| + |x_{i2} - x_{j2}| + \dots + |x_{ik} - x_{jk}|$$

In contrast, we may also define the extent of dissimilarity between 2 observations as the maximum absolute difference in the attributes and thus give a larger influence to larger differences:

$$Linfinity_{ij} = \max(|x_{i1} - x_{j1}|, |x_{i2} - x_{j2}|, \dots, |x_{ik} - x_{jk}|)$$

These three measures are special cases of the Minkowski distance $L(q)$, for $q = 2$ (L2), $q = 1$ (L1), and $q = \infty$ (Linfinity), respectively. Minkowski distances with other values of q may be used as well. Stata supports a wide variety of other similarity and dissimilarity measures, both for continuous variables and for binary variables. See [MV] **measure_option** for details.

Multidimensional scaling constructs approximations for dissimilarities, not for similarities. Thus, if a similarity measure is specified, **mds** first transforms the similarities into dissimilarities. Two methods to do this are available. The default **standard** method,

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

has a useful property: if the similarity matrix is positive semidefinite, a property satisfied by most similarity measures, the standard dissimilarities are Euclidean.

Usually, the number of observations exceeds the number of variables on which the observations are compared, but this is not a requirement for MDS. MDS creates an $n \times n$ dissimilarity matrix **D** from the n observations on k variables. It then constructs an approximation of **D** by the Euclidean distances in a matching configuration **Y** of n points in p -dimensional space:

$$\text{dissimilarity}(x_i, x_j) \approx L2(y_i, y_j) \quad \text{for all } i, j$$

Typically, of course, $p \ll k$, and most often $p = 1, 2$, or 3 .

A wide variety of MDS methods have been proposed. `mds` 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.

Euclidean distances

► Example 1

The most popular dissimilarity measure is Euclidean distance. We illustrate with data from table 7.1 of Yang and Trewn (2004, 182). This dataset consists of eight variables with nutrition data on 25 breakfast cereals.

```
. use http://www.stata-press.com/data/r10/cerealnut
(Cereal Nutrition)
. describe
Contains data from http://www.stata-press.com/data/r10/cerealnut.dta
  obs:                25                Cereal Nutrition
  vars:                9                24 Feb 2007 17:19
  size:               1,150 (99.9% of memory free)  (_dta has notes)
```

variable name	storage type	display format	value label	variable label
brand	str25	%25s		Cereal Brand
calories	int	%9.0g		Calories (Cal/oz)
protein	byte	%9.0g		Protein (g)
fat	byte	%9.0g		Fat (g)
Na	int	%9.0g		Na (mg)
fiber	float	%9.0g		Fiber (g)
carbs	float	%9.0g		Carbs (g)
sugar	byte	%9.0g		Sugar (g)
K	int	%9.0g		K (mg)

Sorted by:

```
. summarize calories-K, sep(4)
```

Variable	Obs	Mean	Std. Dev.	Min	Max
calories	25	109.6	21.30728	50	160
protein	25	2.68	1.314027	1	6
fat	25	.92	.7593857	0	2
Na	25	195.8	71.32204	0	320
fiber	25	1.7	2.056494	0	9
carbs	25	15.3	4.028544	7	22
sugar	25	7.4	4.609772	0	14
K	25	90.6	77.5043	15	320

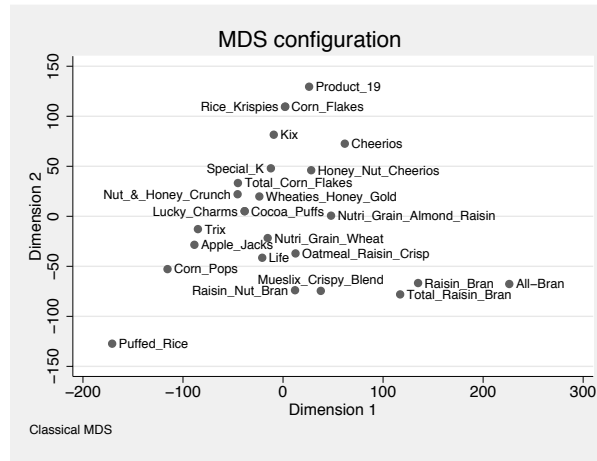
```
. replace brand = substr(brand," ","_",.)
(20 real changes made)
```

We replaced spaces in the cereal brand names with underscores to avoid confusing which words in the brand names are associated with which points in the graphs we are about to produce. Removing spaces is not required.


```

. generate place = 3
. replace place = 9 if inlist(brand,"Rice_Krispies","Nut_&_Honey_Crunch",
> "Special_K","Raisin_Nut_Bran","Lucky_Charms")
(5 real changes made)
. replace place = 12 if inlist(brand,"Mueslix_Crispy_Blend")
(1 real change made)
. mdsconfig, autoaspect mlabvposition(place)

```



The *marker_label_option* `mlabvposition()` allowed fine control over the placement of the cereal brand names. We created a variable called `place` giving clock positions where the cereal names were to appear in relation to the plotted point. We set these to minimize overlap of the names. We also requested the `autoaspect` option to obtain better use of the graphing region while preserving the scale of the x and y axes.

MDS has placed the cereals so that all the brands fall within a triangle defined by Product 19, All-Bran, and Puffed Rice. You can examine the graph to see how close your favorite cereal is to the other cereals.

But, as we saw from the variable summary, three of the eight variables are controlling the distances. If we want to provide for a more equal footing for the eight variables, we can request that `mds` compute the Euclidean distances on standardized variables. Euclidean distance based on standardized variables is also known as the *Karl Pearson distance* (Pearson 1900). We obtain standardized measures with the option `std`.

(Continued on next page)

```
. mds calories-K, id(brand) std noplot
Classical metric multidimensional scaling
dissimilarity: L2, computed on 8 variables
Eigenvalues > 0      =      8      Number of obs      =      25
Retained dimensions =      2      Mardia fit measure 1 = 0.5987
                          Mardia fit measure 2 = 0.7697
```

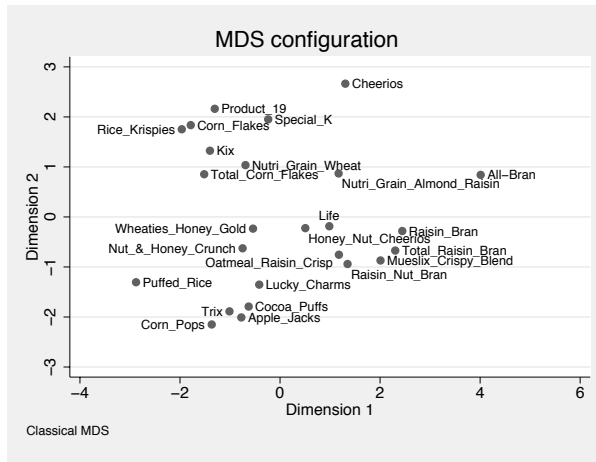
Dimension	Eigenvalue	abs(eigenvalue)		(eigenvalue) ²	
		Percent	Cumul.	Percent	Cumul.
1	65.645395	34.19	34.19	49.21	49.21
2	49.311416	25.68	59.87	27.77	76.97
3	38.826608	20.22	80.10	17.21	94.19
4	17.727805	9.23	89.33	3.59	97.78
5	11.230087	5.85	95.18	1.44	99.22
6	8.2386231	4.29	99.47	0.78	99.99
7	.77953426	0.41	99.87	0.01	100.00
8	.24053137	0.13	100.00	0.00	100.00

In this and the previous example, we did not specify a `method()` for `mds` and got classical metric scaling. Classical scaling is the default when `method()` is omitted and neither the `loss()` nor `transform()` option is specified.

Accounting for more than 99% of the underlying distances now takes more MDS-retained dimensions. For this example, we have still retained only two dimensions. We specified the `noplot` option because we wanted to exercise control over the configuration plot by using the `mdsconfig` command. We generate a variable named `pos` that will help minimize cereal brand name overlap.

```
. generate pos = 3
. replace pos = 5 if inlist(brand,"Honey_Nut_Cheerios","Raisin_Nut_Bran",
> "Nutri_Grain_Almond_Raisin")
(3 real changes made)
. replace pos = 8 if inlist(brand,"Oatmeal_Raisin_Crisp")
(1 real change made)
. replace pos = 9 if inlist(brand,"Corn_Pops","Trix","Nut_&_Honey_Crunch",
> "Rice_Krispies","Wheaties_Honey_Gold")
(5 real changes made)
. replace pos = 12 if inlist(brand,"Life")
(1 real change made)
```

```
. mdsconfig, autoaspect mlabvpos(pos)
```



This configuration plot, based on the standardized variables, better incorporates all the nutrition data. If you are familiar with these cereal brands, spotting groups of similar cereals appearing near each other is easy. The bottom-left corner has several of the most sweetened cereals. The brands containing the word “Bran” all appear to the right of center. Rice Krispies and Puffed Rice are the farthest to the left.

Classical multidimensional scaling based on standardized Euclidean distances is actually equivalent to a principal component analysis of the correlation matrix of the variables. See Mardia, Kent, and Bibby (1979, sec. 14.3) for details.

We now demonstrate this property by doing a principal component analysis extracting the leading two principal components. See [MV] `pca` for details.

```
. pca calories-K, comp(2)
Principal components/correlation          Number of obs   =      25
                                          Number of comp. =       2
                                          Trace           =       8
Rotation: (unrotated = principal)       Rho              =    0.5987
```

Component	Eigenvalue	Difference	Proportion	Cumulative
Comp1	2.73522	.680583	0.3419	0.3419
Comp2	2.05464	.436867	0.2568	0.5987
Comp3	1.61778	.879117	0.2022	0.8010
Comp4	.738659	.270738	0.0923	0.8933
Comp5	.46792	.124644	0.0585	0.9518
Comp6	.343276	.310795	0.0429	0.9947
Comp7	.0324806	.0224585	0.0041	0.9987
Comp8	.0100221	.	0.0013	1.0000

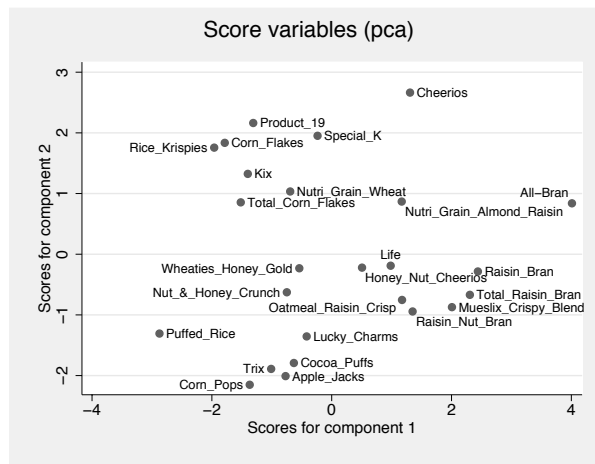
(Continued on next page)

Principal components (eigenvectors)

Variable	Comp1	Comp2	Unexplained
calories	0.1992	-0.0632	.8832
protein	0.3376	0.4203	.3253
fat	0.3811	-0.0667	.5936
Na	0.0962	0.5554	.3408
fiber	0.5146	0.0913	.2586
carbs	-0.2574	0.4492	.4043
sugar	0.2081	-0.5426	.2765
K	0.5635	0.0430	.1278

The proportion and cumulative proportion of the eigenvalues in the PCA match the percentages from MDS. We will ignore the interpretation of the principal components but move directly to the principal coordinates, also known as the scores of the PCA. We make a plot of the first and second scores, using the `scoreplot` command; see [MV] **scoreplot**. We specify the `mlabel()` option to label the cereals and the `mlabelvpos()` option for fine control over placement of the brand names.

```
. replace pos = 11 if inlist(brand,"All-Bran")
(1 real change made)
. scoreplot, mlabel(brand) mlabelvpos(pos)
```



Compare this PCA score plot with the MDS configuration plot. Apart from some differences in how the graphs were rendered, they are the same.

◀

Non-Euclidean dissimilarity measures

With non-Euclidean dissimilarity measures, the parallel between PCA and MDS no longer holds.

▷ Example 2

To illustrate MDS with non-Euclidean distance measures, we will analyze books on multivariate statistics. Gifi (1990) reports on the number of pages devoted to six topics in 20 textbooks on multivariate statistics. We added similar data on five more recent books.

```
. use http://www.stata-press.com/data/r10/mvstatsbooks, clear
. describe
Contains data from http://www.stata-press.com/data/r10/mvstatsbooks.dta
obs:                25
vars:                8                    15 Mar 2007 16:27
size:                825 (99.9% of memory free)  (_dta has notes)
```

variable name	storage type	display format	value label	variable label
author	str17	%17s		
math	int	%9.0g		math other than statistics (e.g., linear algebra)
corr	int	%9.0g		correlation and regression, including linear structural and functional equations
fact	byte	%9.0g		factor analysis and principal component analysis
cano	byte	%9.0g		canonical correlation analysis
disc	int	%9.0g		discriminant analysis, classification, and cluster analysis
stat	int	%9.0g		statistics, incl. dist. theory, hypothesis testing & est.; categorical data
mano	int	%9.0g		manova and the general linear model

Sorted by:

A brief description of the topics is given in the variable labels. For more details, we refer to Gifi (1990, 15). Here are the data:

```
. list, noobs
```

	author	math	corr	fact	cano	disc	stat	mano
	Roy57	31	0	0	0	0	164	11
	Kendall157	0	16	54	18	27	13	14
	Kendall175	0	40	32	10	42	60	0
	Anderson58	19	0	35	19	28	163	52
	CooleyLohnes62	14	7	35	22	17	0	56
	<i>(output omitted)</i>							
	GreenCaroll176	290	10	6	0	8	0	2
	CailliezPages76	184	48	82	42	134	0	0
	Giri77	29	0	0	0	41	211	32
	Gnanadesikan77	0	19	56	0	39	75	0
	Kshirsagar78	0	22	45	42	60	230	59
	Thorndike78	30	128	90	28	48	0	0
	MardiaKentBibby79	34	28	68	19	67	131	55
	Seber84	16	0	59	13	116	129	101
	Stevens96	23	87	67	21	30	43	249
	EverittDunn01	0	54	65	0	56	20	30
	Rencher02	38	0	71	19	105	135	131

For instance, the 1979 book by Mardia, Kent, and Bibby has 34 pages on mathematics (mostly linear algebra); 28 pages on correlation, regression, and related topics (in this particular case, simultaneous

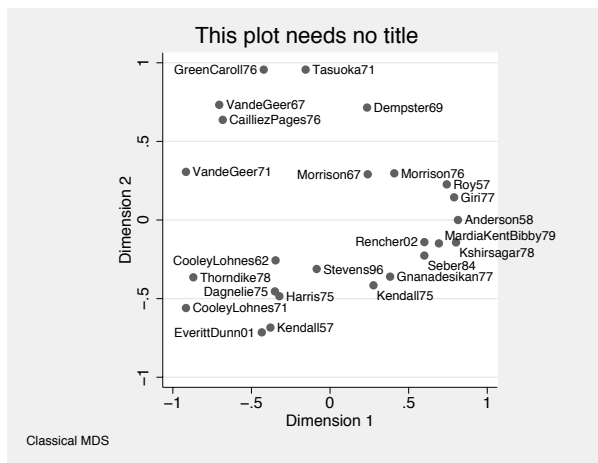
equations); etc. In most of these books, some pages are not classified. Anyway, the number of pages and the amount of information per page vary widely among the books. A Euclidean distance measure is not appropriate here. Standardization does not help us here—the problem is not differences in the scales of the variables but those in the observations. One possibility is to transform the data into *compositional data* by dividing the variables by the total number of classified pages. See Mardia, Kent, and Bibby (1979, 377–380) for a discussion of specialized dissimilarity measures for compositional data. However, we can also use the correlation between observations (not between variables) as the similarity measure. The higher the correlation between the attention given to the various topics, the more similar two textbooks are. We do a classical MDS, suppressing the plot to first assess the quality of a two-dimensional representation.

```
. mds math-mano, id(author) measure(corr) noplot
Classical metric multidimensional scaling
  similarity: correlation, computed on 7 variables
  dissimilarity: sqrt(2(1-similarity))
Eigenvalues > 0      =      6      Number of obs      =      25
Retained dimensions =      2      Mardia fit measure 1 =    0.6680
                          Mardia fit measure 2 =    0.8496
```

Dimension	Eigenvalue	abs(eigenvalue)		(eigenvalue) ²	
		Percent	Cumul.	Percent	Cumul.
1	8.469821	38.92	38.92	56.15	56.15
2	6.0665813	27.88	66.80	28.81	84.96
3	3.8157101	17.53	84.33	11.40	96.35
4	1.6926956	7.78	92.11	2.24	98.60
5	1.2576053	5.78	97.89	1.24	99.83
6	.45929376	2.11	100.00	0.17	100.00

Again the quality of a two-dimensional approximation is somewhat unsatisfactory, with 67% and 85% of the variation accounted for according to the two Mardia criteria. Still, let's look at the plot, using a title that refers to the self-referential aspect of the analysis (Smullyan 1986). We reposition some of the author labels to enhance readability by using the `mLabvpos()` option.

```
. gen spot = 3
. replace spot = 5 if inlist(author,"Seber84","Kshirsagar78","Kendall175")
(3 real changes made)
. replace spot = 2 if author=="MardiaKentBibby79"
(1 real change made)
. replace spot = 9 if inlist(author, "Dagnelie75","Rencher02",
> "GreenCaroll176","EverittDunn01","CooleyLohnes62","Morrison67")
(6 real changes made)
. mdsconfig, mlabvpos(spot) title(This plot needs no title)
```



A striking characteristic of the plot is that the textbooks seem to be located on a circle. This is a phenomenon that is regularly encountered in multidimensional scaling and was labeled the “horseshoe effect” by Kendall (1971, 215–251). This phenomenon seems to occur especially in situations in which a one-dimensional representation of objects needs to be constructed, e.g., in *seriation* applications, from data in which small dissimilarities were measured accurately but moderate and larger dissimilarities are “lumped together”.

◀

□ Technical Note

These data could also be analyzed differently. A particularly interesting method is correspondence analysis (CA), which seeks a simultaneous geometric representation of the rows (textbooks) and columns (topics). We used `camat` to analyze these data. The results for the textbooks were not much different. Textbooks that were mapped as similar using MDS were also mapped this way by CA. The Green and Carroll book that appeared much different from the rest was also displayed away from the rest by CA. In the CA biplot, it was immediately clear that this book was so different because its pages were classified by Gifi (1990) as predominantly mathematical. But CA also located the topics in this space. The pattern was easy to interpret and was expected. The seven topics were mapped in three groups. `math` and `stat` appear as two groups by themselves, and the five applied topics were mapped close together. See [MV] `ca` for information on the `ca` command.

□

Introduction to modern MDS

We return to the data on breakfast cereals explored above to introduce modern MDS. We repeat some steps taken previously and then perform estimation using options `loss(strain)` and `transform(identity)`, which we demonstrate are equivalent to classical MDS.

`mds` is an estimation or `eclass` command, see `program define` in [P] `program`. You can display its saved results using `ereturn list`. The configuration is saved as `e(Y)` and we will compare the configuration obtained from classical MDS with the equivalent one from modern MDS.

▷ Example 3

```

. use http://www.stata-press.com/data/r10/cerealnurt
(Cereal Nutrition)
. replace brand = substr(brand," ","_",.)
(20 real changes made)
. quietly mds calories-K, id(brand) noplot
. mat Yclass = e(Y)
. mds calories-K, id(brand) meth(modern) loss(strain) trans(ident) noplot
Iteration 1:  strain = 594.12657
Iteration 2:  strain = 594.12657
Modern multidimensional scaling
  dissimilarity: L2, computed on 8 variables
  Loss criterion: strain = loss for classical MDS
  Transformation: identity (no transformation)
                                     Number of obs   =       25
                                     Dimensions        =        2
                                     Loss criterion    = 594.1266
  Normalization: principal
. mat Ymod = e(Y)
. assert mreldif(Yclass, Ymod) < 1e-6

```

Note the output differences between modern and classical MDS. In modern MDS we have an iteration log from the minimization of the loss function. The method, measure, observations, dimensions, and number of variables are reported as before, but we do not have or display eigenvalues. The normalization is always reported in modern MDS and with `normalize(target())` for classical MDS. The loss criterion is simply the value of the loss function at the minimum.

◀

Protecting from local minimums

Modern MDS can sometimes converge to a local rather than a global minimum. To protect against this, multiple runs can be made, giving the best of the runs as the final answer. The option for performing this is `protect(#)`, where `#` is the number of runs to be performed. The `nolog` option is of particular use with `protect()`, because the iteration logs from the runs will create a lot of output. Repeating the minimization can take some time, depending on the number of runs selected and the number of iterations it takes to converge.

▷ Example 4

We choose `loss(stress)`, and `transform(identity)` is assumed with modern MDS. We omit the iteration logs to avoid a large amount of output. The number of iterations is available after estimation in `e(ic)`. We first do a run without the `protect()` option, and then we use `protect(50)` and compare our results.

```

. mds calories-K, id(brand) method(modern) loss(stress) nolog noplot
(transform(identity) assumed)
Modern multidimensional scaling
  dissimilarity: L2, computed on 8 variables
  Loss criterion: stress = raw_stress/norm(distances)
  Transformation: identity (no transformation)

                                     Number of obs   =      25
                                     Dimensions        =       2
                                     Loss criterion     =    0.0263

  Normalization: principal
. di e(ic)
45
. mat Ystress = e(Y)
. set seed 123456789
. mds calories-K, id(brand) method(modern) loss(stress) nolog protect(50)
(transform(identity) assumed)
run  mrc  #iter    lossval    seed random configuration
-----
  1    0    74    .02626681  Xdb3578617ea24d1bbd210b2e6541937c4bf2
  2    0   101    .02626681  X630dd08906daab6562fdbc6e21f566d13abb
  3    0    78    .02626681  X73b43d6df67516aea87005b9d405c7020c2c
  4    0    75    .02626681  Xc9913bd7b3258bd7215929fe40ee51c701be
  5    0    75    .02626681  X5fba1fdfb5219b6fd11daa1e739f9a3e0851
  6    0    57    .02626681  X3ea2b7553d633a5418c29f0da0b5cbf433c6
  7    0    84    .02626681  Xeb0a27a9aa7351c262bca0dccbf5dc8e0693
  8    0    75    .02626681  Xcb99c7f17ce97d3e7fa9d27baf16ab60f2a
  9    0    85    .02626681  X8551c69beb028bbd48c91c1a1b6e6f0e2b08
 10    0    60    .02626681  X05d89a191a939001044c3710631948c12c41
 11    0    63    .02626681  X2c6eaeaf4dcd2394c628f466db148b3419c0
 12    0    45    .02626681  <initial nonrandom value>
 13    0    55    .02626681  X167c0bd9c43f462544a474abacbdd93d03c8
 14    0    57    .02626682  X51b9b6c5e05aadd50ca4b924a252124048e1
 15    0    82    .02626682  Xff9280cf913270440939762f65c2b4d622da
 16    0    63    .02626682  X14f4e343d3e32b22014308b4d2407e8949e3
 17    0    63    .02626682  X1b06e2ef52b30203908f0d10327044174a08
 18    0    66    .02626682  X70ceaf639c2f78374fd6a1181468489e3288
 19    0    72    .02626682  X5fa530eb49912716c3b27b00020b158c16cc
 20    0    71    .02626682  Xf2f8a723276c4a1f3c7e5848c07cc438343e
 21    0    52    .02626682  Xd5f821b18557b512b3ebc04c992e06b4115d
 22    0    66    .02626683  Xd0de57fd1b0a448b3450528326fab45d1681
 23    0    61    .02626683  X6d2da541fcdb0d9024a0a92d5d0496231d51
 24    0    59    .02626683  Xb8aae22160bc2fd1beaf5a4b98f46a254a25
 25    0    84    .02626684  Xe2b46cd1199d7ef41b8cd31479b25b274bdb
 26    0   138    .026303    Xb21a3be5f19dad75f7708eb425730c6e45b0
 27    0   100    .026303    X22ef41a50a68221b276cd98ee7dfeef51073
 28    0    74    .026303    X0efbcec71dbb1b3c7111cb0f622502830b54
 29    0    55    .026303    X01c889699f835483fd6182719be301f13e96
 30    0    56    .026303    X2f66c0cb554aca4ff44e5d6a6cd4b6273931
 31    0    67    .026303    X3a5b1f132e05f86d36e01eb46eff578b3d24
 32    0    67    .026303    Xac0226dd85d4f2c440745210acea6ceb12a4
 33    0    75    .026303    X9e59768de92f8d2ab8e9bc0fd0084c7d10ea
 34    0    58    .026303    X333e991d0bf2f212b1e1025e348a6825470b6c
 35    0    60    .026303    Xedef05bfbdbcddd5a2b8abeadcdd5ab74c9c
 36    0    59    .026303    X67e0caf9c38ba588e96cd01d5d908d7f022c
 37    0    53    .026303    X2af205b7aad416610a0dec141b66778a2eee
 38    0    52    .026303    X0b9944753b1c4b3bd3676f624643a915319c
 39    0    87    .026303    Xb175975333f6bdee5bc301e7d30556882042
 40    0    63    .02630301  X7e334ce7d25be1deb7b30539d716026639ef
 41    0    60    .02630301  Xf2e6bfadef621544c441e8363c853045203e
 42    0    60    .02630301  X45c7e0abd63ad668fa94cd4758d974eb2635
 43    0    58    .02630301  X60263a35772a812860431439cad14ad92943

```

```

44 0 66 .02630301 X4bf3debb1c7e07f66b533ec5941e1e07433b
45 0 63 .02630301 X01f186db4f0db540e749c79e59717c18247e
46 0 56 .02630302 X66a301f734b575da6762a4edcf9ac6492715
47 0 53 .02630302 X5c59c9ffd2e9f2e5bd45f3f9aa22b2f027b7
48 0 131 .19899027 Xe2e15b07d97b0bcb086f194a133dd7b23f52
49 0 138 .23020403 X065b9333ce65d69bf4d1596e8e8cc72904ef
50 0 170 .23794378 X075bcd151f123bb5159a55e50022865746ad

```

Modern multidimensional scaling

dissimilarity: L2, computed on 8 variables

Loss criterion: stress = raw_stress/norm(distances)

Transformation: identity (no transformation)

Number of obs = 25

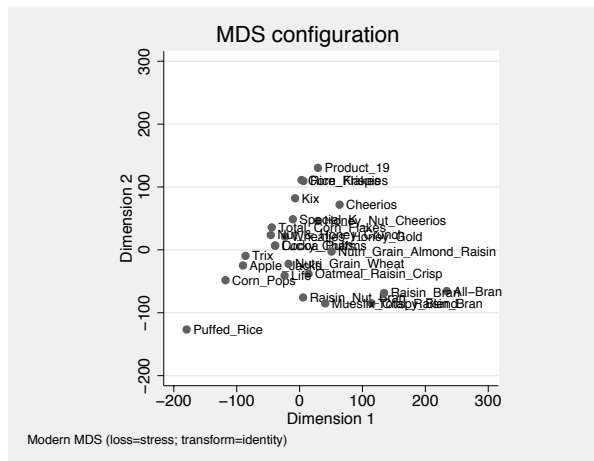
Dimensions = 2

Loss criterion = 0.0263

Normalization: principal

```
. mat YstressP = e(Y)
```

```
. assert mreldif(Ystress, YstressP) < 2e-3
```



The output provided when `protect()` is specified includes a table with information on each run, sorted by the loss criterion value. The first column simply counts the runs. The second column gives the internal return code from modern MDS. This example only has values of 0, which indicate converged results. The column header `mrc` is clickable and opens a help file explaining the various MDS return codes. The number of iterations is in the third column. These runs converged in as few as 47 iterations to as many as 190. The loss criterion values are in the fourth column, and the final column contains the seeds used to calculate the starting values for the runs.

In this example, the results from our original run versus the protected run did not differ by much, approximately $1.3e-3$. However, looking at runs 46–50 we see loss criterion values that are much higher than the rest. The loss criteria for runs 1–45 vary from .02627 to .02630, but these last runs' loss criteria are all more than .198. These runs clearly converged to local, not global, minimums.

The graph from this protected modern MDS run may be compared with the first one produced. There are obvious similarities, though inspection indicates that the two are not the same.

Saved Results

mds saves the following in `e()`:

Scalars

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

Macros

<code>e(cmd)</code>	mds
<code>e(cmdline)</code>	command as typed
<code>e(method)</code>	classical or modern MDS method
<code>e(method2)</code>	nonmetric, if <code>method(nonmetric)</code> specified
<code>e(loss)</code>	loss criterion
<code>e(losstitle)</code>	description loss criterion
<code>e(tfunction)</code>	identity, power, or monotonic, transformation function
<code>e(transftitle)</code>	description of transformation
<code>e(id)</code>	observation identifier variable
<code>e(idtype)</code>	int or str; type of <code>id()</code> variable
<code>e(duplicates)</code>	1 if duplicates in <code>id()</code> , 0 otherwise
<code>e(labels)</code>	labels for ID categories
<code>e(strfmt)</code>	format for category labels
<code>e(mxlen)</code>	maximum length of category labels
<code>e(varlist)</code>	variables used in computing similarities or dissimilarities
<code>e(dname)</code>	similarity or dissimilarity measure name
<code>e(dtype)</code>	similarity or dissimilarity
<code>e(s2d)</code>	standard or oneminus (when <code>e(dtype)</code> is similarity)
<code>e(unique)</code>	1 if eigenvalues are distinct, 0 otherwise
<code>e(init)</code>	initialization method
<code>e(iseed)</code>	seed for <code>init(random)</code>
<code>e(seed)</code>	seed for solution
<code>e(norm)</code>	normalization method
<code>e(targetmatrix)</code>	name of target matrix for <code>normalize(target)</code>
<code>e(properties)</code>	nob noV for modern or nonmetric MDS; nob noV eigen for classical MDS
<code>e(estat_cmd)</code>	program used to implement <code>estat</code>
<code>e(predict)</code>	program used to implement <code>predict</code>

(Continued on next page)

Matrices

e(D)	dissimilarity matrix
e(Disparities)	disparity matrix for nonmetric MDS
e(Y)	approximating configuration coordinates
e(Ev)	eigenvalues
e(idcoding)	coding for integer identifier variable
e(coding)	variable standardization values; first column has value to subtract and second column has divisor
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

mds is implemented as an ado-file.

mds creates a dissimilarity matrix **D** according to the *measure* specified in option `measure()`. See [MV] *measure_option* for descriptions of these measures. Subsequently, mds uses the same subroutines as `mdsmat` to compute the MDS solution for **D**. See [MV] *mdsmat* for information on the methods and formulas.

References

- Borg, I., and P. J. F. Groenen. 2005. *Modern Multidimensional Scaling: Theory and Applications*. 2nd ed. New York: Springer.
- Cox, T. F., and M. A. A. Cox. 2001. *Multidimensional Scaling*. 2nd ed. Boca Raton, FL: Chapman & Hall/CRC.
- Gifi, A. 1990. *Nonlinear Multivariate Analysis*. New York: Wiley.
- Kendall, D. G. 1971. Seriation from abundance matrices. *Mathematics in the Archaeological and Historical Sciences*. Edinburgh: Edinburgh University Press.
- 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*. New York: Academic Press.
- Pearson, K. 1900. Mathematical contributions to the theory of evolution VII: On the correlation of characters not quantitatively measurable. *Philosophical Transactions of the Royal Society. Series A* 195: 1–47.
- Sammon, J. W. 1969. A nonlinear mapping for data structure analysis. *IEEE Transactions on Computers*. 18(5): 401–409.
- Smullyan, R. 1986. *This Book Needs No Title: A Budget of Living Paradoxes*. New York: Touchstone.
- Torgerson, W. S. 1952. Multidimensional scaling: I. Theory and method. *Psychometrika* 17: 401–419.
- Yang, K., and J. Trewn. 2004. *Multivariate Statistical Methods in Quality Management*. New York: McGraw–Hill.
- Young, F. W., and R. M. Hamer. 1994. *Theory and Applications of Multidimensional Scaling*. 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] **biplot** — Biplots

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

[MV] **factor** — Factor analysis

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

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

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

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